МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по выполнению лабораторных работ
по дисциплине «Разработка Web-приложений» для студентов направления подготовки 09.03.04 Программная инженерия
Направленность (профиль)
«Разработка и сопровождение программного обеспечения»

Ставрополь 2026

Оглавление

Введение 3

Лабораторная работа № 1. 11

Лабораторная работа № 2. 32

Лабораторная работа № 3. 43

Лабораторная работа № 4. 54

Лабораторная работа № 5. 80

Лабораторная работа № 6. 95

Лабораторная работа № 7 117

Лабораторная работа № 8 130

Лабораторная работа № 9. 142

Лабораторная работа № 10. 156

Лабораторная работа №11. 180

Лабораторная работа № 12 203

Лабораторная работа №13 219

Лабораторная работа № 14. 266

Лабораторная работа №14 (часть2) 317

Лабораторная работа №15. 360

Лабораторная работа №16 380

Лабораторная работа №17 407

Лабораторная работа №18 434

Лабораторная работа №19 462

Лабораторная работа №20 497

Лабораторная работа №21 533

Лабораторная работа №22 571
Лабораторная работа №23 602
Лабораторная работа №24 630
Лабораторная работа №25 649
Лабораторная работа №26 676
Лабораторная работа №27 713
Лабораторная работа №28 743
Лабораторная работа №30 797
Лабораторная работа №31 827
Лабораторная работа №32 854
Лабораторная работа №33 884
Лабораторная работа №34 920
Лабораторная работа №35 959
Лабораторная работа №36 985

Введение

Общая характеристика лабораторного практикума

Лабораторный практикум по дисциплине «Веб-технологии» представляет собой комплекс из 36 лабораторных работ, охватывающих полный цикл разработки современных веб-приложений - от создания виртуального окружения и первых строк кода на Django до контейнеризации, CI/CD, мониторинга и интеграции искусственного интеллекта.

Практикум построен по принципу «от простого к сложному» и разбит на три логических модуля, каждый из которых формирует определённый набор профессиональных компетенций.

Структура лабораторного практикума

Модуль 1. Бэкенд-разработка на Django (лабораторные работы 1–14)

Данный модуль закладывает фундамент для создания серверной части веб-приложений. Студенты последовательно осваивают:

1. Создание виртуального окружения, настройку проекта и маршрутизацию (ЛР №1)
2. Работу с моделями, миграциями и базой данных PostgreSQL (ЛР №2, №5)
3. Основы CRUD-операций и работу с ORM (ЛР №3)
4. Создание шаблонов, наследование, фильтры и теги (ЛР №4, №6, №8)
5. Параметризацию URL-адресов и именование маршрутов (ЛР №7)
6. Кастомные теги шаблонов (ЛР №9)
7. Обратное разрешение адресов и работу со статическими файлами (ЛР №10)
8. Работу с формами (ЛР №11)
9. Классовые представления (ListView, DetailView, CreateView) (ЛР №12)
10. Продвинутую работу с ORM: агрегацию, аннотацию, Q и F-объекты (ЛР №13)
11. Оптимизацию запросов, миксины, пагинацию и Django Debug Toolbar (ЛР №14)

Модуль 2. Фронтенд-разработка с React (лабораторные работы 15–24)

Второй модуль посвящён созданию современных одностраничных приложений (SPA) на React. Студенты изучают:

1. Создание проекта через Vite и настройку маршрутизации (ЛР №15)
2. Управление состоянием с Context API и Redux Toolkit (ЛР №16, №17)
3. Работу с API на клиенте с использованием React Query (ЛР №18)
4. Аутентификацию и защиту маршрутов (ЛР №19)
5. Создание и валидацию форм с Formik и Yup (ЛР №20)
6. Использование UI-библиотеки Material-UI для быстрого интерфейса (ЛР №21)
7. Тестирование React-компонентов (ЛР №22)
8. Оптимизацию производительности (ЛР №23)
9. Сборку и деплой статического фронтенда (ЛР №24)

Модуль 3. Интеграция, тестирование, DevOps и ИИ (лабораторные работы 25–36)

1. Заключительный модуль объединяет бэкенд и фронтенд в единое production-решение:
2. Интеграцию React с Django API и настройку CORS (ЛР №25)
3. Тестирование бэкенда с pytest (ЛР №26)
4. Сквозное тестирование с Cypress (ЛР №27)
5. Контейнеризацию с Docker (ЛР №28) и оркестрацию с Docker Compose (ЛР №29)
6. Production-развертывание с Nginx и Gunicorn (ЛР №30)
7. Развертывание на VPS (ЛР №31)
8. Настройку CI/CD с GitHub Actions (ЛР №32)
9. Автоматизацию инфраструктуры с Ansible (ЛР №33)
10. Централизованное логирование и мониторинг (Loki, Prometheus, Grafana) (ЛР №34)

11. Резервное копирование и отказоустойчивость (ЛР №35)
12. Интеграцию ИИ-модели в веб-приложение (ЛР №36)

Цели и задачи лабораторного практикума

Цель практикума

Сформировать у студентов практические навыки разработки, тестирования, развертывания и сопровождения промышленных веб-приложений с использованием современных технологий (Django, React, Docker, CI/CD, мониторинг, ИИ).

Задачи практикума

1. Бэкенд-разработка
2. Освоить создание REST API на Django REST Framework
3. Научиться проектировать базы данных и работать с ORM
4. Освоить аутентификацию, авторизацию и защиту API
5. Фронтенд-разработка
6. Научиться создавать SPA на React с маршрутизацией и управлением состоянием
7. Освоить работу с формами, UI-библиотеками и тестирование компонентов
8. Интеграция и DevOps
9. Научиться связывать фронтенд и бэкенд через API
10. Освоить контейнеризацию и оркестрацию приложений
11. Настроить CI/CD для автоматического тестирования и деплоя
12. Освоить мониторинг, логирование и резервное копирование
13. Интеграция ИИ
14. Научиться внедрять готовые модели машинного обучения в веб-приложения
15. Освоить асинхронную обработку и кэширование результатов

Организация выполнения лабораторных работ

Каждая лабораторная работа содержит следующие разделы:

Раздел	**Содержание**
Цель работы	Формулировка того, что студент должен освоить
Теоретическое обоснование	Краткое изложение ключевых понятий и концепций
Практический пример	Пошаговое руководство с пояснениями и кодом
Индивидуальные задания	15 вариантов для каждого из трёх уровней сложности
Контрольные вопросы	Вопросы для самопроверки и защиты работы
Требования к отчёту	Структура и содержание отчёта
Критерии оценивания	Таблица с баллами и шкала перевода в оценку

Уровни сложности заданий

Для каждой лабораторной работы предусмотрены три уровня сложности индивидуальных заданий:

Базовый уровень - минимально необходимый набор операций для закрепления базовых навыков

Средний уровень - задания с дополнительными требованиями и более сложной логикой

Продвинутый уровень - комплексные задания, требующие самостоятельного проектирования и интеграции нескольких технологий

Рекомендации по выполнению

Последовательность - работы следует выполнять в порядке возрастания номеров, так как каждая последующая работа опирается на знания и код, полученные в предыдущих.

Активное использование документации - в каждой работе приведены ссылки на официальную документацию; настоятельно рекомендуется обращаться к ней для углублённого изучения.

Ведение отчёта - результаты выполнения каждой работы должны быть оформлены в виде отчёта, содержащего код, скриншоты, выводы и ответы на контрольные вопросы.

Самостоятельность - индивидуальные задания предназначены для самостоятельного выполнения; копирование кода не способствует формированию компетенций.

Контроль времени - оценка выставляется с учётом сроков сдачи; работы, сданные с задержкой, оцениваются со штрафными коэффициентами.

Формируемые компетенции

В результате выполнения лабораторного практикума студенты овладевают следующими профессиональными компетенциями:

Компетенция	Содержание
ПК-2	Способность выполнять проектирование и разработку архитектуры программных систем среднего и крупного масштаба с применением

Компетенция	Содержание
	современных паттернов и методов проектирования
ПК-3	Владение навыками разработки программного обеспечения: способность реализовывать алгоритмы, писать код на современных языках программирования, создавать веб- и desktop-приложения, работать с базами данных
ПК-8	Способность адаптировать и применять методы искусственного интеллекта и машинного обучения для решения профессиональных задач, включая интеграцию готовых моделей в программные продукты

Структура отчёта по лабораторной работе

Отчёт должен содержать:

Титульный лист с названием работы, ФИО студента, номером группы и вариантом

Цель работы

Теоретическое обоснование (своими словами, 2–3 абзаца)

Выполнение практического примера (код, скриншоты, пояснения)

Выполнение индивидуального задания (код, скриншоты, пояснения)

Ответы на контрольные вопросы (не менее 5 по своему уровню)

Заключение (вывод о проделанной работе, приобретённых навыках)

Критерии оценки (общие для всех работ)

Оценка	Характеристика
5 (отлично)	Задание выполнено полностью, код работает без ошибок, даны полные ответы на контрольные вопросы, отчёт оформлен аккуратно, присутствует анализ проделанной работы
4 (хорошо)	Задание выполнено полностью, но есть незначительные недочёты (стиль кода, неполные ответы на вопросы, недостаточно подробный вывод)
3 (удовлетворительно)	Задание выполнено частично, есть ошибки, код работает с недочётами, ответы на вопросы неполные
2 (неудовлетворительно)	Задание не выполнено, код отсутствует или не работает, нет ответов на вопросы

Данный лабораторный практикум представляет собой полный курс подготовки разработчика веб-приложений - от первого «Hello World» до production-развертывания с мониторингом и интеграцией искусственного интеллекта. Успешное выполнение всех 36 работ гарантирует формирование компетенций, достаточных для самостоятельной разработки промышленных веб-приложений в команде или в качестве индивидуального разработчика.

Лабораторная работа № 1.

Создание виртуального окружения. Контроллеры и маршруты

Цель работы: настроить виртуальное пространство. Разработать контроллер приложения и организовать маршруты

Методика выполнения

1. Создание виртуального окружения

1 способ

Создадим папку MyProject, и в этой папке создадим проект с названием вашего проекта, например MySite и создадим виртуальное окружение.

Установим виртуальное окружение с помощью модуля vnv, который установлен в python и вводим команду **python -m venv venv**, где название виртуального окружения venv, через терминал проекта.

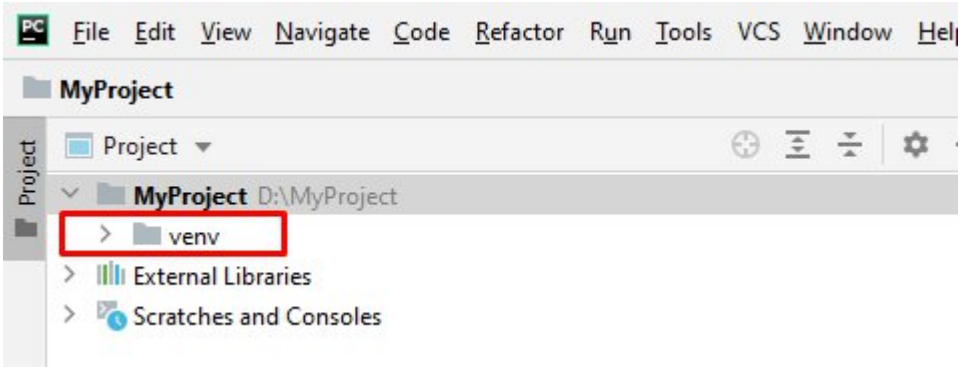


Рисунок 1

В папке MyProject появится папка **venv** вашего виртуального окружения.

Введите команду pip list. Выведится список глобально установленных библиотек.

Возможно выведится строка с рекомендацией обновления pip (зеленая строка на рисунке 2)

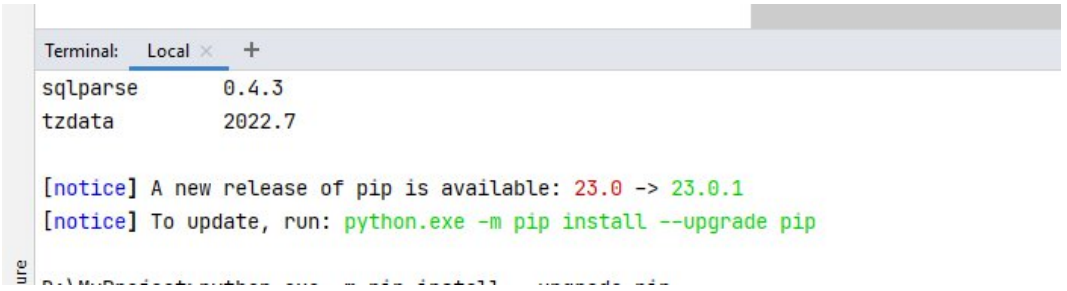


Рисунок 2

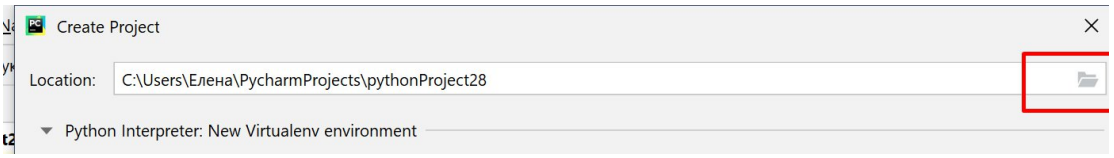


Рисунок 1

Если просмотрим вложенные файлы, то увидим те пакеты, которые установлены на данный момент.

Например, в папке Scripts находится python.

2 способ

Второй способ основан на автоматическом создании средствами системы программирования PyCharm, которое создается при создании проекта.

После создания виртуального окружения, необходимо его активировать. Для этого необходимо перейти в Терминал PyCharm и набрать команду venv\Scripts\activate

Файл activate, который находится в папке Scripts активирует виртуальное пространство (рисунок 3).

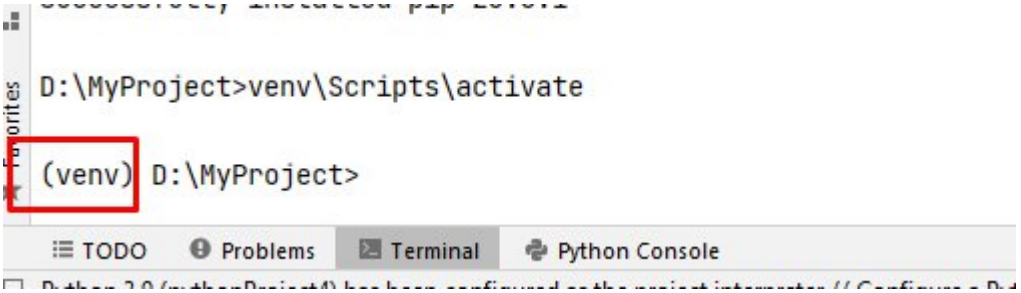


Рисунок 3

Как вы видите (рисунок 3), в командной строке стоит не только путь нахождения папки проекта, но и имя нашего виртуального окружения.

Запустим команду pip list

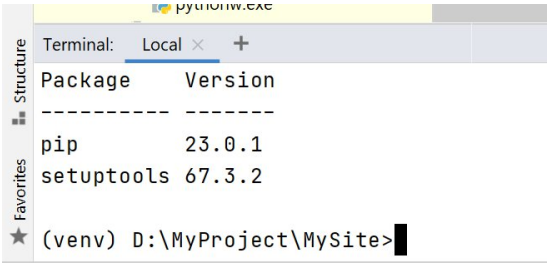


Рисунок 4

Данная команда указала на наличие двух пакетов.

```
(venv) D:\MyProject>pip list
Package    Version
---------- -------
pip        20.2.3
setuptools 49.2.1
WARNING: You are using pip version 20.2.3; however, version 23.0.1 is available.
You should consider upgrading via the 'd:\myproject\venv\scripts\python.exe -m pip install --upgrade pip' command.

(venv) D:\MyProject>
```

Рисунок 5

Следующим шагом является установка Django

Установочный пакет Django находится по адресу pypi.org (рисунок 6)

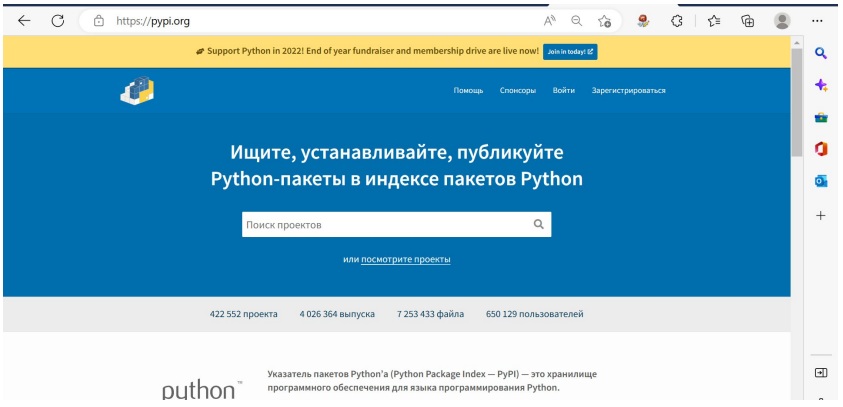


Рисунок 6

В поисковой строке введите Django и в списке выберете последнюю версию (рисунок 7)

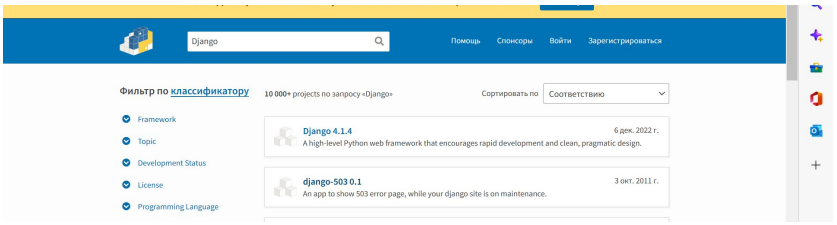


Рисунок 7

Например, Django 4.1.4.

Добавим Django в наш проект в виртуальном окружении, созданном ранее. Открываем через PyCharm. И используем команду pip install Django.

Снова выполним команду pip list

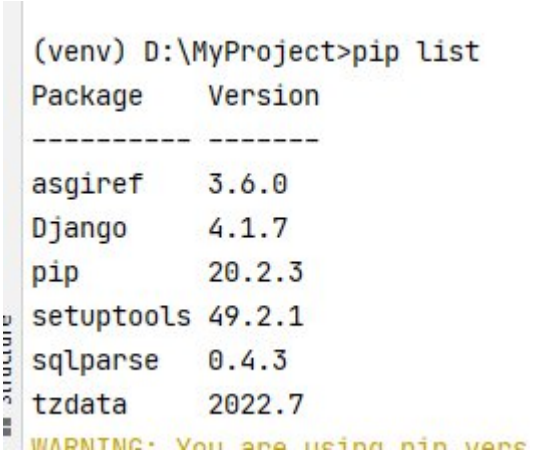


Рисунок 8

Мы видим обновленный список пакетов, в том числе, Django

После установки можно приступать к созданию проекта. Но прежде всего давайте разберем команды, которые нам могут пригодиться.

1. django-admin

Это команда получения списка доступных подкоманд

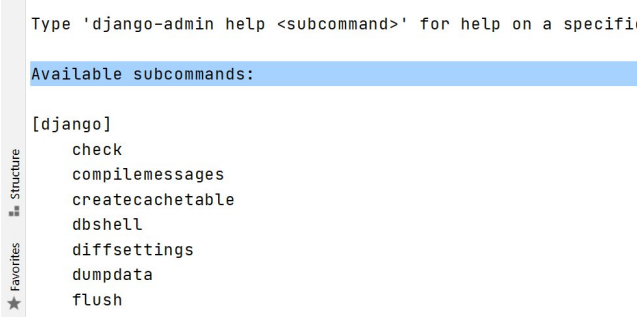


Рисунок 9

Многие команды из этого списка мы будем с вами использовать при работе над проектом.

Начнем работу с команды старта проекта – startproject.

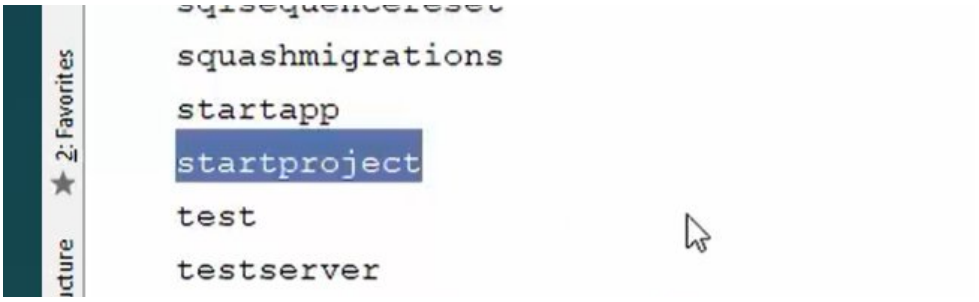


Рисунок 10

В терминале введем команду

django-admin startproject MySite,

где MySite – название вашего проекта

После ввода этой команды, в дереве вашего проекта появится папка с названием MySite.

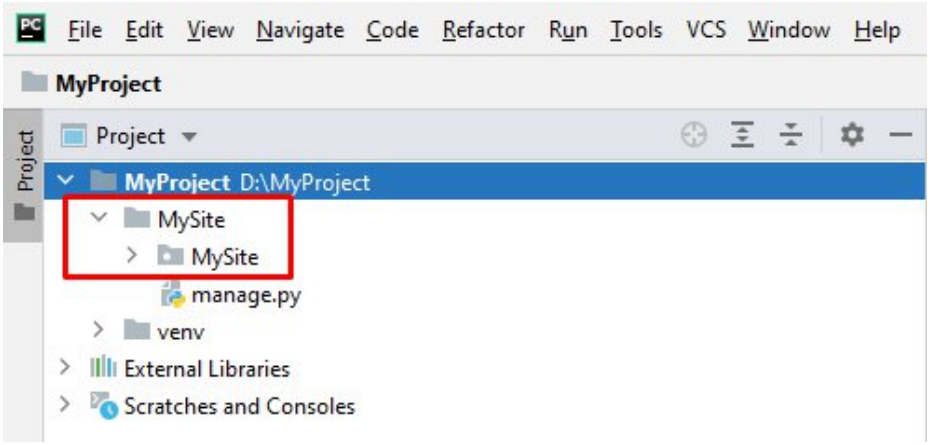


Рисунок 11

Внутри этой папки создана одноименная папка, в которой сформирован пакет, задающий конфигурацию пакета. О том, что это пакет, говорит нам папка _init_py.

Одноименную папку называют пакетом конфигурации.

Проект готов, он, конечно, еще пустой, но мы можем его попробовать запустить. Нужно запустить сервер и для этого мы обращаемся к manage.py.

Для этого перейдем в папку проекта с помощью команды

cd MySite

и здесь уже команду (рисунок 13)

python manage.py runserver



Рисунок 13

Команда runserver запускает тот самый отладочный веб сервер который поставляется вместе с Django, т.е. дополнительно устанавливать не надо. По умолчанию он запускается по адресу http://127.0.0.1:8000/

Если перейдем по этому адресу, то уже запустится шаблон нашего будущего сайта (рисунок 14).

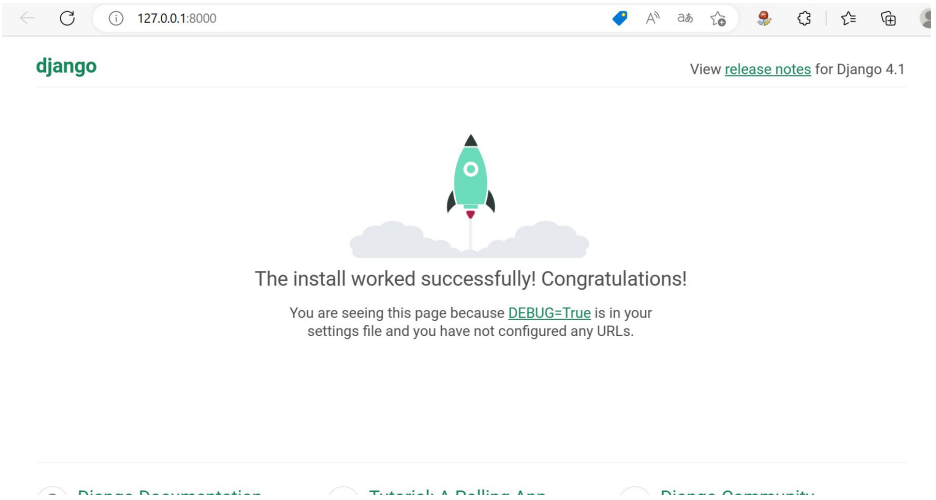
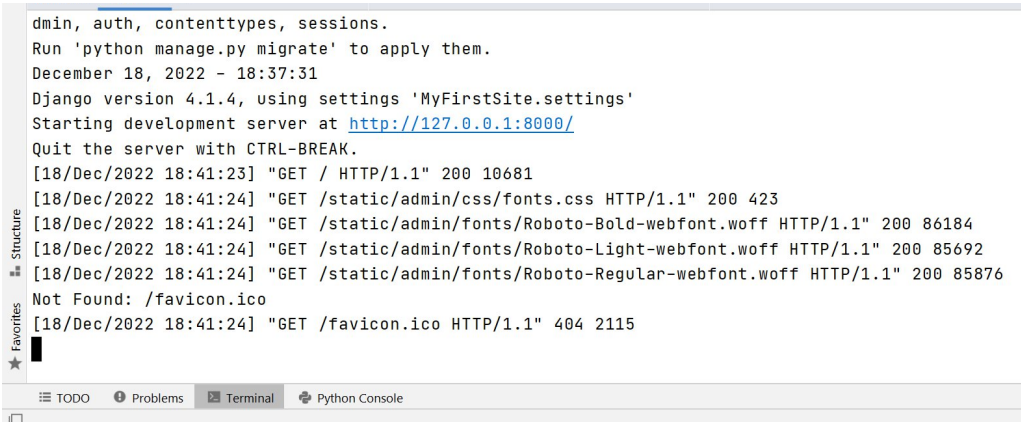


Рисунок 14

Здесь есть http запросы



Все то, что мы запрашиваем в Django, он отображает все запросы здесь с терминале (рисунок 15).

Прелесть отладочного сервера, что он проверяет изменения в проекте и если вы что-то поменяли в существующем коде, он перезапустит сервер и подхватит эти изменения, но не поддерживает проверку в новых файлах.

Для перезагрузки отключить manager.pty с с помощью комбинации клавиш Ctrl+C.

После запуска проекта появился новый файл (рисунок 16).

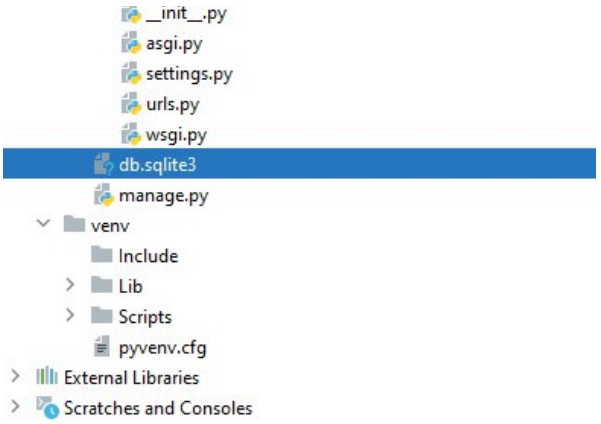


Рисунок 16

Это файл базы данных sqlite, которая поддерживается Django по умолчанию. Но Django поддерживает и другие БД.

Проект пустой пока, нет никакого функционала. Все что мы видим сейчас это пакет конфигурации проекта. Чтобы создать что-нибудь,

необходимо создать хотя бы одно приложение. И весь сайт состоит из приложения.

Создадим новостной сайт. Можно открыть вторую вкладку терминала и в командной строке терминала перейдем в папку MySite, где находится manage.py и введем команду cd MytSite

И введем команду создания приложения

python manage.py startapp news

где news – это имя создаваемого приложения

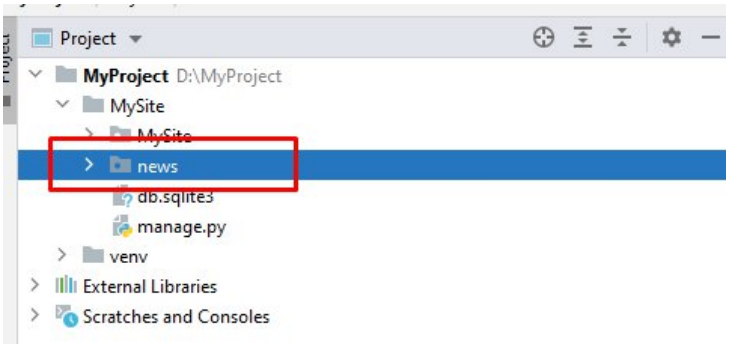


Рисунок 17

После создания приложения, чтобы Django корректно работало, это приложение необходимо зарегистрировать. Регистрируется в конфигурации проекта в файле settings.py

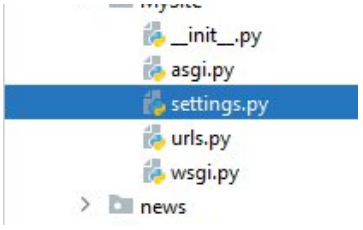


Рисунок 18

В этом файле хранятся всевозможные настройки и регистрируют в части INSTALLED_APPS

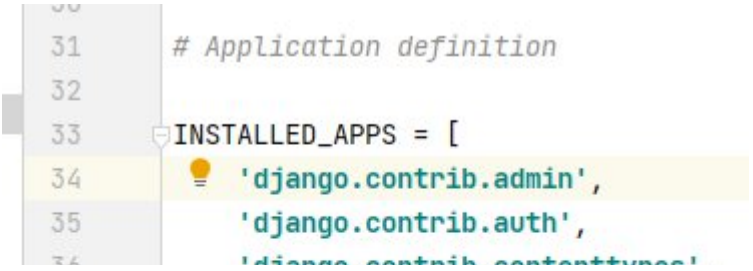


Рисунок 19

Здесь необходимо указать путь к пакету файла конфигурации app.py и указать класс

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'news.apps.NewsConfig',
]
```

Рисунок 20

Для создания контроллеров используется файл views.py

Напишем первую контроллер (функцию), которую назовем index. На вход эта функция принимает обязательный аргумент – экземпляр класса HTTP request и называется он request. В нем хранятся данные о полученном запросе, о его клиенте, его браузере, хранятся куки, сессии и т.д. Когда мы получили запрос, то нам нужно вернуть какой-то ответ.

Возвращать будем HTTP ответ, который нам позволит сформировать класс HTTP respons. Его необходимо импортировать

```
from django.http import HttpResponse
```

и будем его же возвращать и в качестве аргумента передадим строку.

```
from django.shortcuts import render
from django.http import HttpResponse


def index(request):
    print(request)
    return HttpResponse('Hello world')
```

Рисунок 21

Если попытаемся набрать в адресной строке браузера адрес нашего приложения news, то мы получим ошибку, поскольку маршрутизатор не знает такой адрес. Page not found at /news/ , то получим ошибку 404 (рисунок 22)

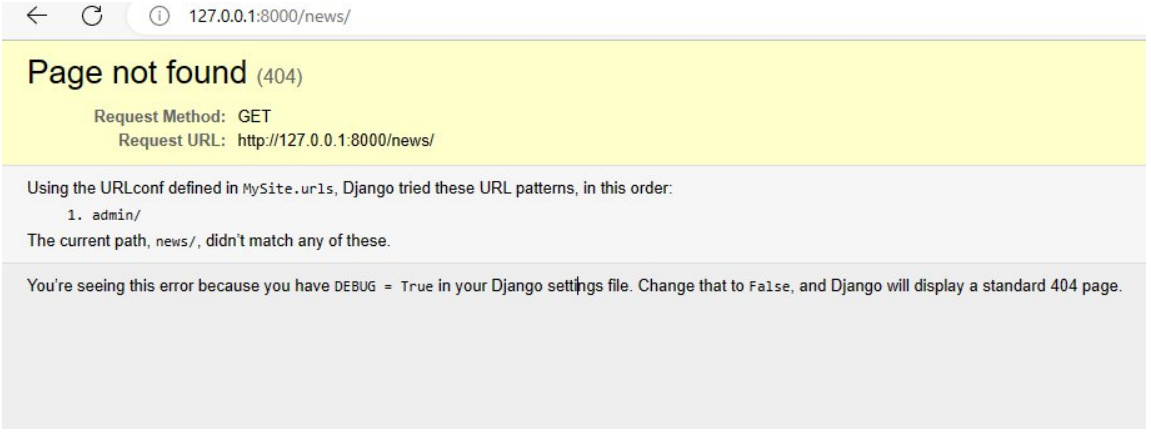


Рисунок 22

Django должен понимать, что при запросе того или иного адреса он должен обратиться к данному приложению, именно к данному контроллеру функции.

Здесь на помощь приходит маршрутизатор, который принимает первичный запрос клиента и сравнивает его со списками записанных маршрутов, адресов и говорит, есть ли у него такой адрес или нет. Если есть, то он вызывает этот адрес и говорит клиенту, куда ему надо отправиться.

Список адресов находится в пакете конфигурации в файле urls.py

Здесь уже есть один адрес админки

```
import ...

urlpatterns = [
    path('admin/', admin.site.urls),
]
```

Рисунок 23

Если мы в адресной строке браузера допишем admin (http://127.0.0.1:8000/admin), то получим страницу входа в админку

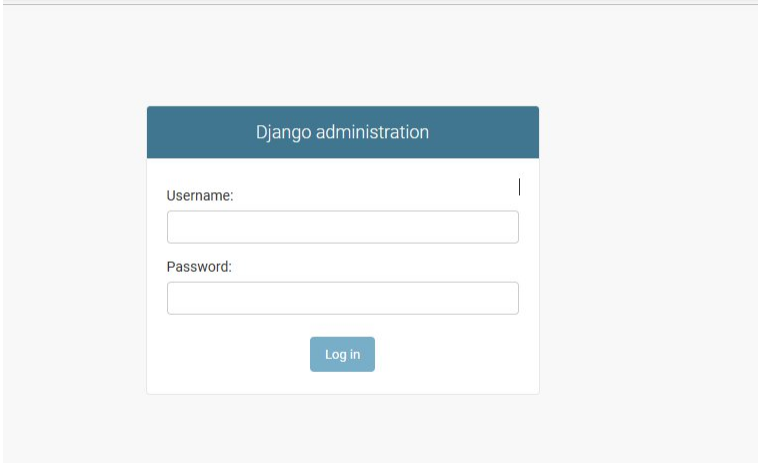


Рисунок 24

Для того, чтобы прописать маршрутизатору путь необходимо импортировать функцию index контроллера views

```
from news.views import index
```

При этом, как мы видим, на рисунке красным цветом подсвечиваются ошибки. Это означает, что не знает такого пути, потому что корневой директорией для нашего проекта является верхняя папка MySite, в которой ищется папка views и не находит. Но эта папка есть во вложенной папке MySite и поэтому эту папку надо указать как корневую для нашего проекта.

```
from news.views import index

urlpatterns = [
    path('admin/', admin.site.urls),
]
```

Рисунок 25

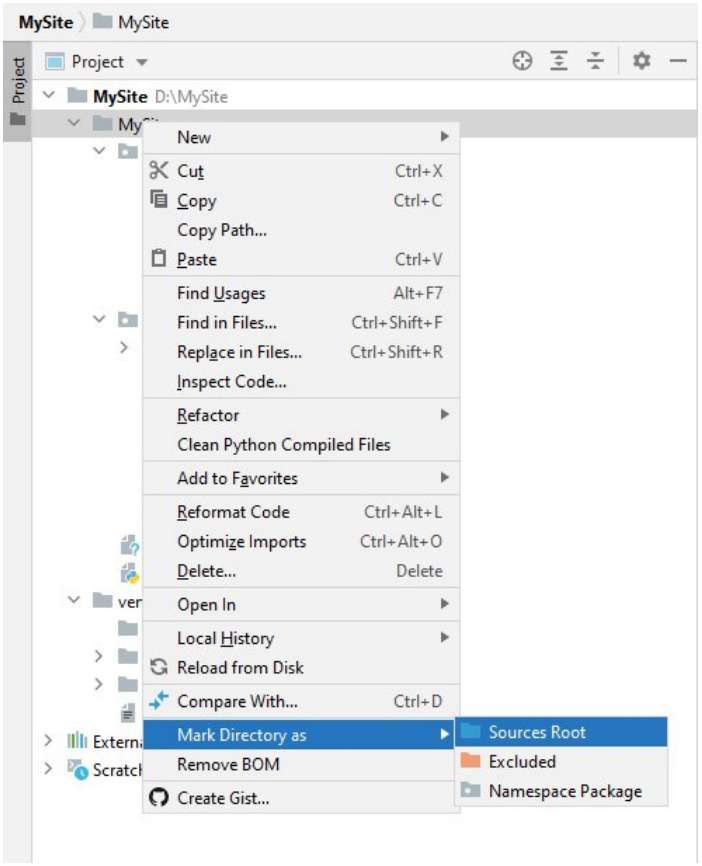


Рисунок 26

Для этого правой кнопкой мыши по этой папке щелкаем и выбираем Mark Directory as/Sources Root.

Мы импортировали нашу функцию index из модуля news в модуль views. Теперь можем вызвать функцию path с помощью этой функции мы можем указать новый маршрут. Таких маршрутов может быть много. Но нас пока интересует один. Что делает маршрутизатор, получив этот адрес? Он проходит по списку маршрутов и ищет совпадения. Первое найденное совпадение его устраивает, и он вызывает нужную функцию. В данном случае функция index, которую только что импортировали.

После импортирования вводим функцию path, которая указывает маршрут.

path('**news/**', index)

Маршрутизатор отбрасывает весь путь кроме имени нашего приложения **news**.

В терминале можно увидеть, выведенный нами, запрос.

Если введем строку print(dir(request)) и после обновления страницы index просмотрим в Терминале все методы этого запроса

```
def index(request):
    print(dir(request))
    return HttpResponse('Hello wordl')
```

Рисунок 27

```
Django version 4.1.7, using settings 'MyFirstSite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
['COOKIES', 'FILES', 'GET', 'META', 'POST', '__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash
, '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', '_current_s
get_raw_host', '_get_scheme', '_initialize_handlers', '_load_post_and_files', '_mark_post_parse_error', '_messages', '_read_started', '_set_content_type_params', '_set_post', '_
 'body', 'build_absolute_uri', 'close', 'content_params', 'content_type', 'csrf_processing_done', 'encoding', 'environ', 'get_full_path', 'get_full_path_info', 'get_host', 'get_
', 'parse_file_upload', 'path', 'path_info', 'read', 'readline', 'readlines', 'resolver_match', 'scheme', 'session', 'upload_handlers', 'user']
[16/Feb/2023 23:14:13] "GET /news/ HTTP/1.1" 200 11
```

Рисунок 28

Давайте создадим еще одну функцию test

```
def index(request):
    # print(dir(request))
    return HttpResponse('Hello wordl')

def test(request):

    return HttpResponse('<h1>Тестовая страница<\h1>')
```

Рисунок 29

И тоже для него зададим маршрут.

```
15  """
16  from django.contrib import admin
17  from django.urls import path
18  from news.views import *
19
20  urlpatterns = [
21      path('admin/', admin.site.urls),
22      path('news/', index),
23      path('test/', test),
24  ]
```

Рисунок 30

Посмотрим на нашу страницу. И распечатаем наш запрос

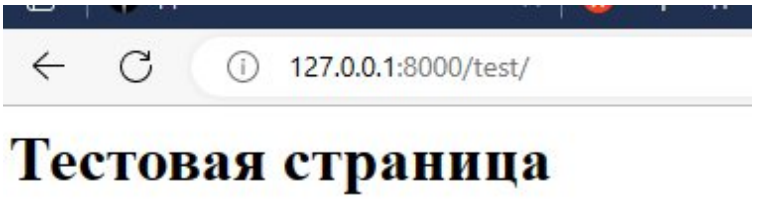


Рисунок 31

Страница test должна быть вложенной для приложения news.

Перепишем маршрут следующим образом:

Для этого нужно хранить все маршруты в нашем приложении news. Достигается путем импорта этих маршрутов. Для этого импортировать функцию include



Рисунок 32

Этого файла в нашем приложении нет, поэтому мы его должны создать .

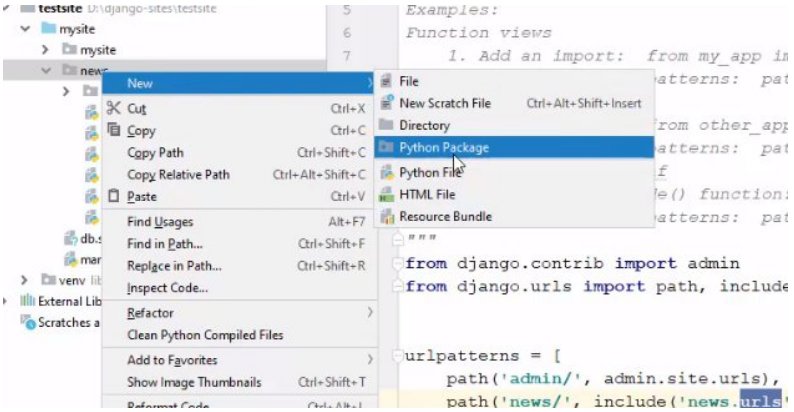


Рисунок 33

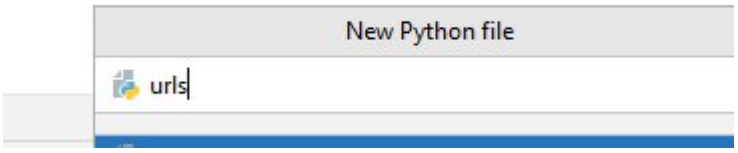


Рисунок 34

Импортируем функцию path



Рисунок 35

И создаем переменную urlpatterns со списком адресов.



Рисунок 36

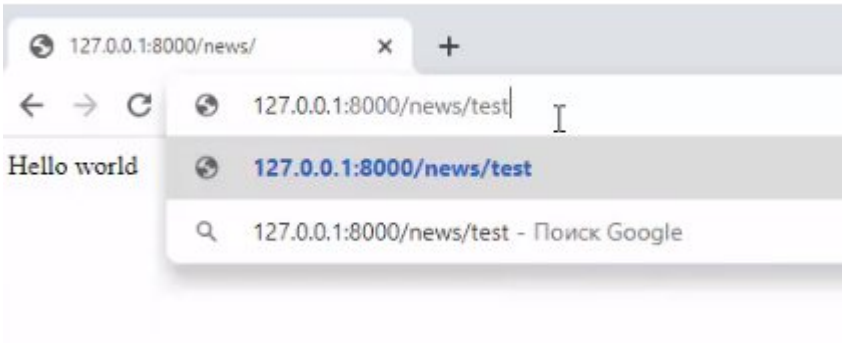


Рисунок 37

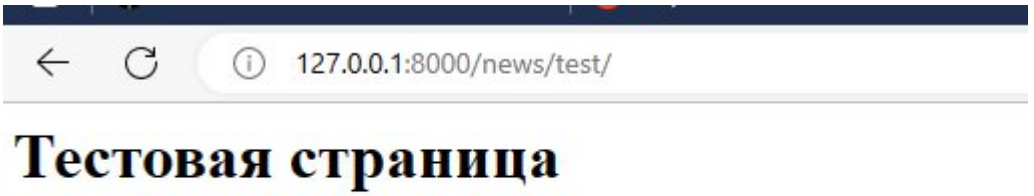


Рисунок 38

Практические задания для индивидуального выполнения

Задание 1.

Создайте виртуальное окружение для нового проекта "BlogProject", установите Django и сохраните список всех установленных пакетов в файл requirements.txt. Продемонстрируйте процесс активации окружения.

Задание 2.

Создайте Django-проект с названием "MySite". Запустите сервер и убедитесь, что страница приветствия Django отображается корректно. Измените порт запуска на 8080.

Задание 3.

В созданном проекте разработайте приложение "articles". Зарегистрируйте его в настройках проекта.

Задание 4.

В приложении "articles" создайте три view-функции:

- home – возвращает текст "Добро пожаловать на сайт!"
- about – возвращает текст "Информация о проекте"
- contacts – возвращает текст "Наши контакты: email@example.com"

Задание 5.

Создайте маршруты для созданных функций:

- На уровне проекта подключите маршруты приложения
- В самом приложении создайте urls.py и пропишите все три маршрута
- Убедитесь, что каждый URL корректно обрабатывается

Задание 6.

Модифицируйте функцию home так, чтобы она выводила в терминал информацию о:

- методе HTTP-запроса

- пути запроса
- user-agent клиента

Задание 7.

Создайте вложенную структуру URL:

- Основной маршрут articles/ – ведет на функцию article_list
- Вложенный маршрут articles/archive/ – ведет на функцию archive
- Реализуйте обе функции с выводом произвольного текста

Задание 8.

Предложите и реализуйте структуру URL для интернет-магазина с разделами: каталог товаров, корзина, личный кабинет, оформление заказа. Достаточно создать только маршруты и заглушки view-функций.

Задание 9.

Объясните и покажите на практике разницу между размещением маршрутов в глобальном urls.py и в файле маршрутов приложения. В каком случае какой подход предпочтительнее?

Задание 10.

Создайте ситуацию с ошибкой импорта view-функции в urls.py и продемонстрируйте, как её исправить с помощью настройки корневой директории в PyCharm (Sources Root).

Контрольные вопросы и задания для защиты лабораторной работы №1

Контрольные вопросы:

1. Виртуальное окружение:

- Что такое виртуальное окружение и для чего оно используется в Python-проектах?

- Какие существуют способы создания виртуального окружения? Опишите их преимущества и недостатки.

- Какие файлы и папки создаются при инициализации виртуального окружения? Объясните назначение папки Scripts (или bin).

- Как активировать и деактивировать виртуальное окружение? Как понять, что оно активно?

2. Управление пакетами:

- Для чего используется команда pip list? Какую информацию она отображает?

- Как установить Django в виртуальное окружение?

- Где хранятся установленные пакеты в виртуальном окружении?

- Как создать файл зависимостей (requirements.txt) и восстановить окружение из него?

3. Django-проект:

- Какая команда используется для создания нового Django-проекта? Какова её структура?

- Что такое manage.py и какие функции он выполняет?

- Объясните назначение файлов и папок, создаваемых в пакете конфигурации проекта.

- Как запустить встроенный веб-сервер Django? Какой адрес используется по умолчанию?

4. Приложения в Django:

- Чем отличается проект от приложения в Django?

- Какая команда создает новое приложение? Приведите пример.

- Где и как необходимо регистрировать созданное приложение? Почему это важно?

- Какие файлы создаются в структуре приложения? Объясните назначение каждого.

5. Контроллеры (представления):

- В каком файле создаются контроллеры-функции?

- Какие обязательные параметры должна принимать view-функция? Что они означают?

- Что такое HttpResponse и как его использовать?

- Как вывести отладочную информацию в терминал при обработке запроса?

6. Маршрутизация:

- Где хранятся маршруты уровня проекта? Как они выглядят?

- Как подключить маршруты приложения к проекту? Какую функцию для этого используют?

- Зачем создавать отдельный urls.py внутри приложения?

- Как работает процесс сопоставления URL с view-функцией?

7. Ошибки и отладка:

- Почему возникает ошибка "Page not found (404)" при попытке обратиться к приложению?

- Какую функцию выполняет отладочный сервер? Какие у него особенности?

- Как корректно остановить работу сервера?

Лабораторная работа № 2.

Модели. Шаблоны.

Цель работы: разработка моделей, научиться проводить миграцию, данных и создавать шаблоны

Методика выполнения

Модели хранят данные не у себя в файлах, а хранят в базах данных (БД). Django работает sqllite3. И для управления понадобится любой менеджер БД. Воспользуемся менеджером sqlitestudio_x64-3.4.3, портативной версией, которую вы можете скачать и без установки использовать в проекте.

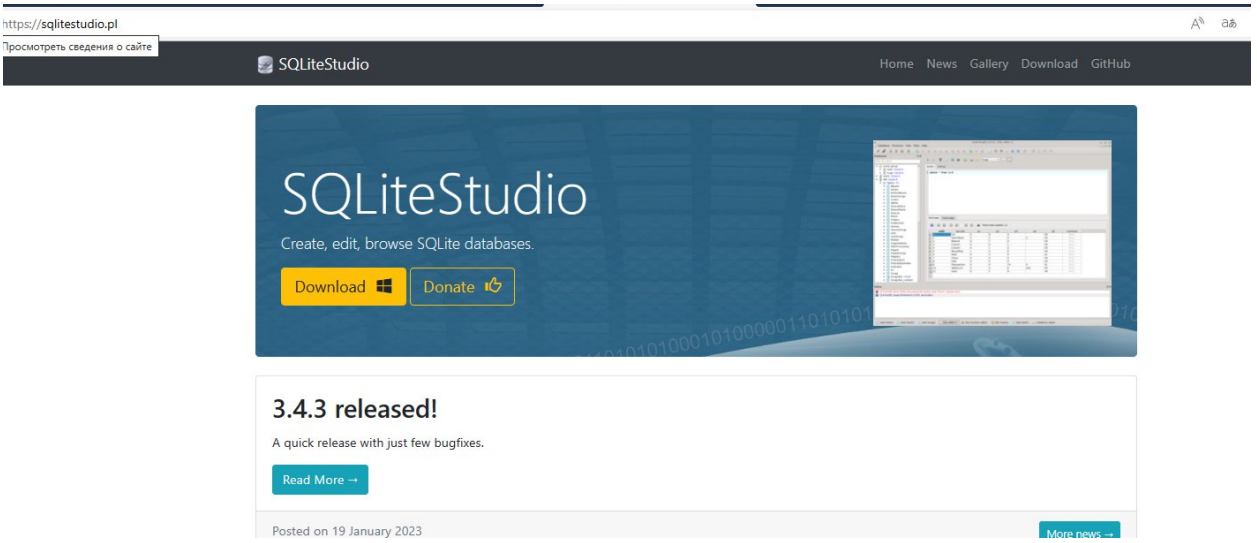


Рисунок 1

Выбрать версию менеджера и установить на своем компьютере с помощью файла SQLiteStuduo.exe (рисунок 3).

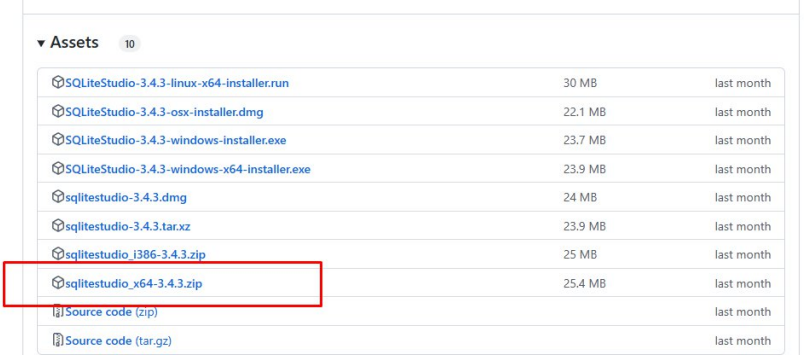


Рисунок 2

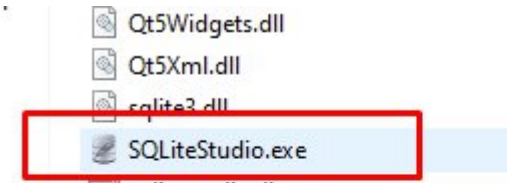


Рисунок 3

Добавляем БД (рисунок 4). В поле файл необходимо указать путь файла БД нашего проекта db.sqlite3.

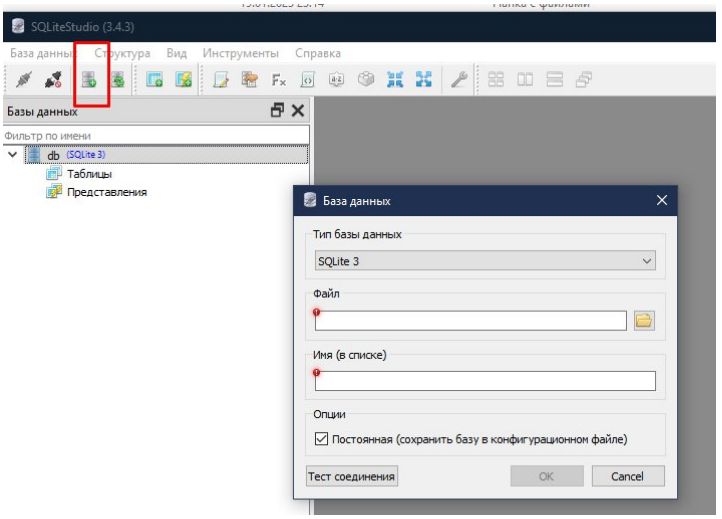


Рисунок 4

Подключаем БД (рисунок 5)

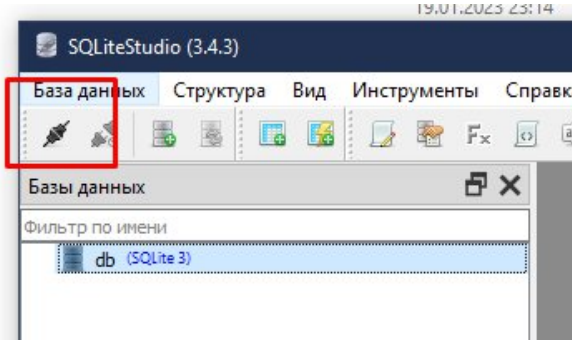


Рисунок 5

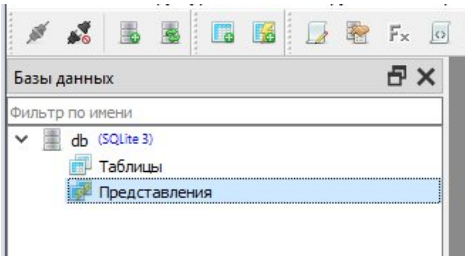


Рисунок 6

Пока база данных у нас пустая. Модели в Django описываются классом внутри соответствующего файла models в приложении news.

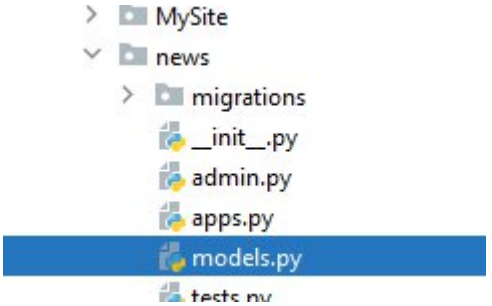


Рисунок 7

Здесь будем писать свой класс, который является подклассом

django.db import models и получать все методы для работы с данными.

В этом классе мы будем описывать атрибуты (свойства) класса в модели, которые являеются обязательным. Атрибуты – это соответствующие поля таблицы БД. В нашем случае это таблица news.

Давайте определим поля нашей таблицы.

Название поля	**Атрибут**	**Назначение поля**
title	Текстовое	Название статьи
content	Текстовое	Содержание статьи
create_at	Дата/Время	Дата и время создания статьи
updated	Дата/Время	Дата и время обновления статьи
photo	Тип картинка	
is_published	логическое	Опубликована статья/ не опубликована статья

При выборе названий полей, то нельзя использовать зарезервированные слова.

Каждый атрибут должен быть экземпляром соответствующего класса Field

Класс создается с помощью class имя(models.Model): (описание полей с типами)

Создаем модель

```
class News(models.Model):
    title=models.CharField(max_length=150)
    content=models.TextField(blank=True)
    created_at=models.DateTimeField(auto_now_add=True)
    updated_at=models.DateTimeField(auto_now=True)
    photo=models.ImageField(upload_to='photos/%Y/%m/%d')
    is_published=models.BooleanField(default=True)
```

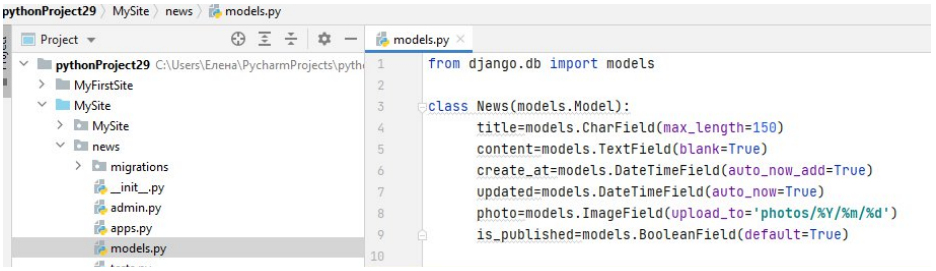


Рисунок 8

ID мы не будем описывать. Он создается автоматически Django.

Опишем параметры соответствующих полей:

auto_now_add - автоматически установите для поля значение «Сейчас» при первом создании объекта.

auto_now - автоматически устанавливайте для поля значение «Сейчас» при каждом сохранении объекта

upload_to - является форматирование strftime(), четырехзначный год, двузначный месяц и равен двузначный день. Если вы загрузите файл 15 января 2023 г., он будет сохранен в каталог ./home/media"photos/%Y/%m/%d"%Y/%m/%d"%Y"%m"%d'/home/media/photos/2023/01/15

Blank=True - означает, что это поле обязательное для заполнения.

После создания модели необходимо перейти к созданию файла миграции (рисунок 9).

Миграции - это способ, которым Django распространяет изменения, которые вы вносите в модели (добавление поля, удаление модели и т. Д.) В схеме базы данных. Они разработаны так, чтобы быть почти автоматическими, но вам нужно знать, когда создавать миграции, когда их запускать, а также общие проблемы, с которыми вы можете столкнуться.

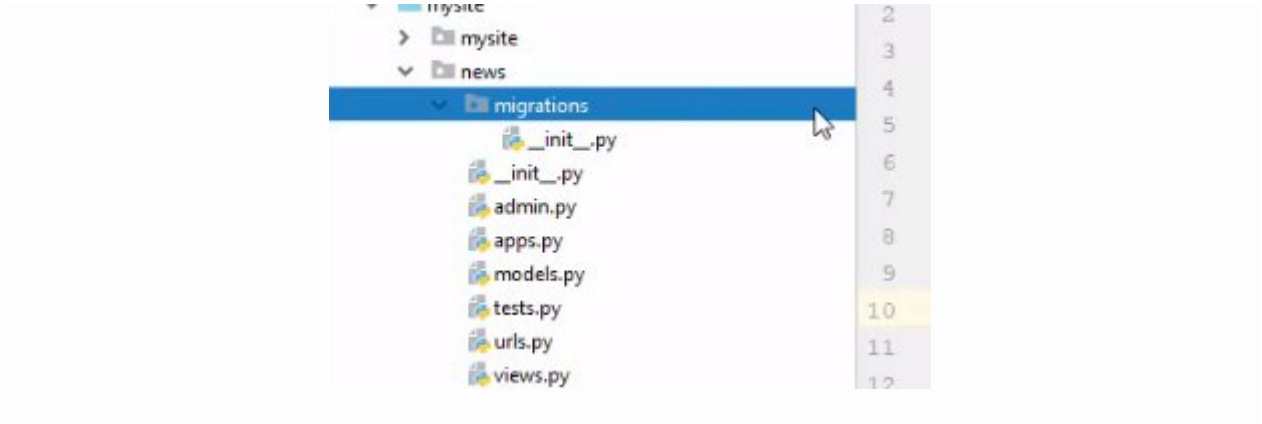


Рисунок 9

Создаются с помощью команды makemigrations, которую выполняем в терминале. Но прежде чем проводить миграцию, необходимо убедиться, что все поля заданы корректно и подключены все библиотеки. Например, если будет обнаружена ошибка такого рода как на рисунке 10, то прежде перед создание миграции необходимо установить библиотеку Pillow для работы с изображениями. И пока не будут установлены нужные библиотеки миграцию провести не возможно.

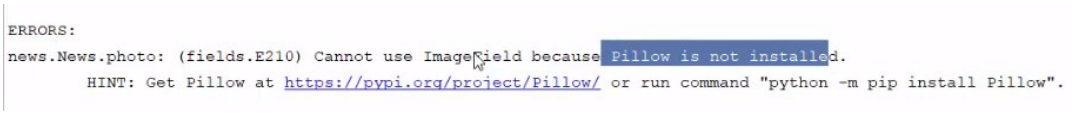


Рисунок 10

Для установки этой библиотеки воспользуемся командой pip install Pillow (рисунок 11)



Рисунок 12

Перезапустите сервер.

В терминале введем команду создания миграции makemigrations, которая отслеживает все изменения в моделях и создает миграции для всех этих изменений. Строка команды следующая (рисунок 13)

python manage.py makemigrations

```
venv) C:\Users\Елена\PycharmProjects\pythonProject29\MySite>python manage.py makemigrations
igrations for 'news':
 news\migrations\0001_initial.py
   - Create model News

venv) C:\Users\Елена\PycharmProjects\pythonProject29\MySite>
```

Рисунок 13

В результате буден создан файл 0001_initial.py (рисунок 14)

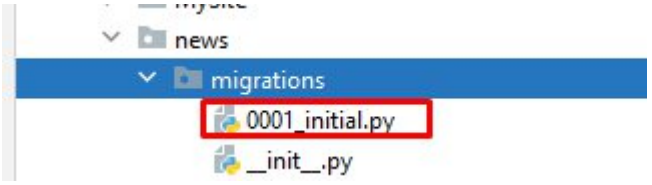


Рисунок 14

А также создана модель News (рисунок 15)

```
igrations for 'news':
  news\migrations\0001_initial.py
    - Create model News
```

Рисунок 15

В файле 0001_initial.py мы можем наблюдать эту миграцию (рисунок 16).



Рисунок 16

Как вы видите, id, который мы не указывали в модели был создан автоматически.

Посмотрим SQL запрос, который может быть выполнен при создании таблица в БД.

В терминале выполним следующий запрос (рисунок 17)

python manage.py sqlmigrate news 0001



Рисунок 17

В пакете конфигурации в файле settings можно увидеть какая БД в настоящий момент подключена (рисунок 18) в виде словаря DATABASES .



Рисунок 18

Как мы видим, в настоящем проекте подключена БД sqlite3.

В словаре используется константа BASE_DIR, которая прописывается в этом файле и вычисляется местонахождение вашей БД, указывает на корень нашего проекта (рисунок 19).

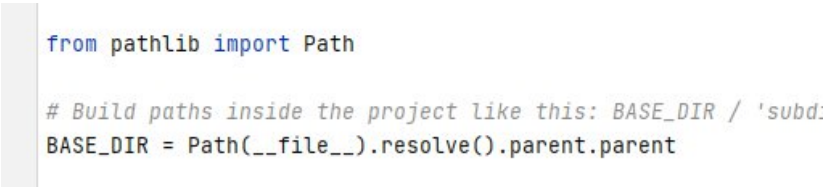


Рисунок 19

Теперь перейдем к выполнению миграций. При старте сервера в терминале можно увидеть сообщение (рисунок 20), в котором говориться, что у нас не применено 18 миграций. Это миграции, которые есть в проекте Django за исключением нашей и которые отвечают за админку, сессии и прочее.

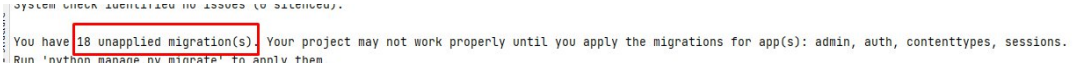


Рисунок 20

Для того, чтобы выполнить миграции используется команда

python manage.py migrate

Выполняем ее (рисунок 21)



Рисунок 21

OK в конце каждой строки указывает, что все миграции выполнены успешно в том числе и миграция нашей модели, которая выполнилась последней.

Обновим структуру нашей БД и увидим, что здесь появились таблицы (рисунок 22) среди которых мы видим таблицу news_news.

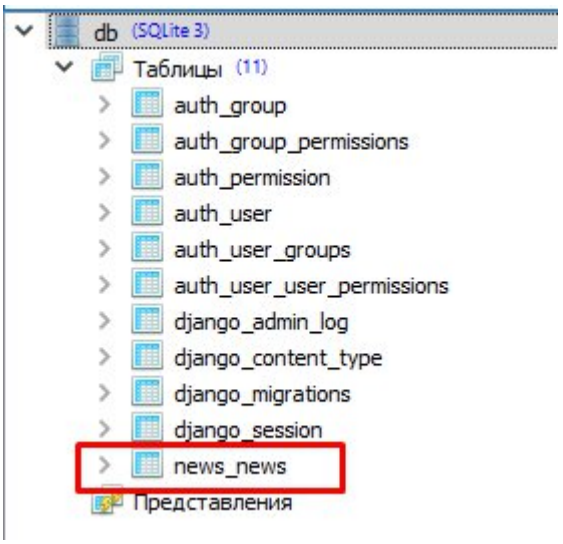


Рисунок 22

Выбрав в дереве таблиц таблицу news_news, можно просмотреть поля таблицы, соответствующие полям нашей модели (рисунок 23).

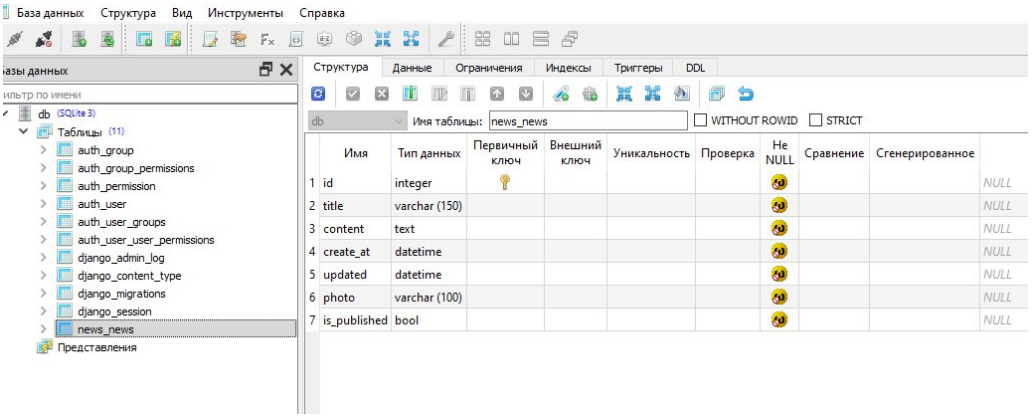


Рисунок 23

А также SQL запрос, который был использован для этой таблицы (рисунок 24)

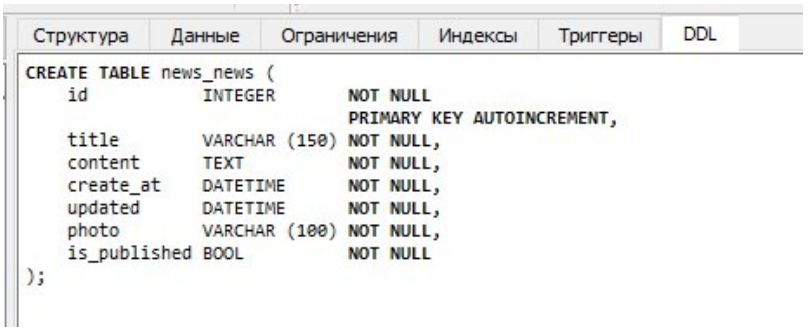


Рисунок 24

Необходимо обратить внимание на поле photo, которое имеет тип varchar и изображения, которого нам необходимо куда-то сохранять.

Для настройки этого поля необходимо настроить константы MEDIA_ROOT и MEDIA_URL. Эти файлы нужны для того, чтобы указать, куда загружать файлы.

Переходим в файл настроек settings и в конце файла дописываем константы (рисунок 25)

```
MEDIA_ROOT=os.path.join(BASE_DIR, 'media')
MEDIA_URL= '/media/'
```

```
DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

MEDIA_ROOT=os.path.join(BASE_DIR, 'media')
MEDIA_URL= '/media/'
```

Рисунок 25

Для того, чтобы файлы-картинки загружались на странице сайте (отображались), то необходимо создать соответствующий маршрут, который будет указывать на загруженные файлы. Для этого используется функция static из модуля Django. Эта функция создаст маршрут, по которому маршрутизатор передаст при запросе файлов управление специальному контроллеру Django, который отдаст файл-фото для просмотра. Но этот маршрут необходим только в отладочном режиме, то есть только при разработке сайта. При переводе на реальный сервер этот маршрут нам уже будет не нужен и файлы будут обслуживаться самим веб-сервером.

В файле settings есть логическая константа DEBUG = True, которой присвоено значение True. Это означает, что ведется разработка. После окончания разработки True необходимо заменить на False.

Переходим в корневой файл urls.

Исходя из константы DEBUG мы обращаемся к модулю settings для того, чтобы указать, путь для загрузки медиа в отладочном режиме через условие. При этом потребуется подключение библиотеки, которую можно

импортировать вручную или нажать Alt+Enter и выбрать импортируемую библиотеку (рисунок 25).

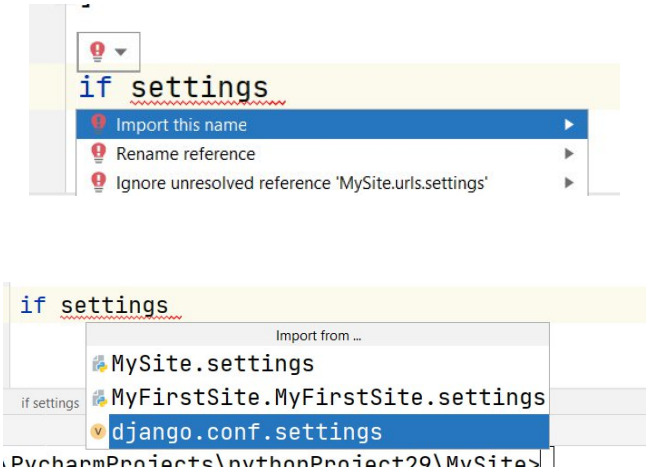


Рисунок 25

И дописываем собственный маршрут, который сформирует функция static, которую тоже нужно импортировать (рисунок 26).

uments

```
from django.conf import settings
from django.contrib import admin
from django.urls import path, include
from django.conf.urls.static import static
from news.views import *


urlpatterns = [
    path('admin/', admin.site.urls),
    path('news/',include('news.urls')),

]

if settings.DEBUG:
    urlpatterns+=static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Рисунок 26

По этому маршруту будем получать медиа-файлы.

Лабораторная работа № 3.

Основы работы с моделями. CRUD

Цель работы: разработка моделей, научиться проводить миграцию, данных и создавать шаблоны

Методика выполнения

Для заполнения базы данных, которая была создана в предыдущей работе (рисунок 1), необходимо добавить данные через админку.

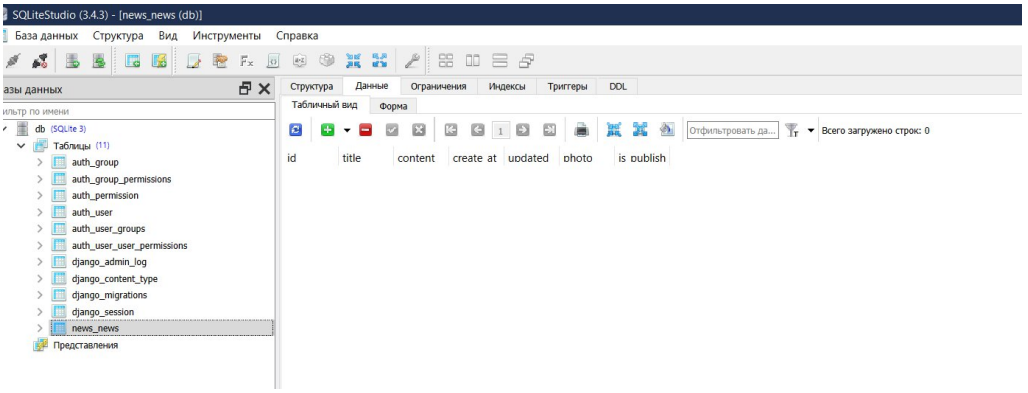


Рисунок 1

И для этого необходимо создать суперпользователя для админки, который обладает всеми правами и может работать в админке. Но пока научимся добавлять данные из консоли Django для ознакомления работы с API для работы с моделями.

У Django есть свое API для работы с моделями, есть ORM.

Чтобы протестировать модель, воспользуемся консолью Django. Консоль Django – это таже самая интерактивная консоль Python, где можно писать код. Откроем ее. Для этого необходимо перейти в папку нашего проекта в Pycharm и выполнить команду shell через обращение к manage.py (рисунок 2)

python manage.py shell



Рисунок 2

Как видим, после ввода становится активна интерактивная консоль InteractiveConsole.

Для того чтобы работать с моделью нам нужно ее импортировать из приложения, где она находится. Наша модель находиться в приложении news. Обращайте внимание на регистры. Модели принято называть в верхнем регистре (рисунок 3).

```
>>> from news.models import News
>>>
```

Рисунок 3

Запишем данные в таблицу. Запись модели создаются вызовом ее конструктора news и передаче в этот самый конструктор именованных параметров с соответствующими атрибутами модели. Просмотрим, какие параметры нужно передавать. Предполагается поле title, которое должно быть обязательным, поле Content с атрибутом blank=True, который указывает, что данное поле является не обязательным для заполнения. Поле created_add мы не трогаем. При единоразовой записи, это поле будет заполнено текущим временем. Поле updated_at будет заполнено при создании записи и при каждом ее обновлении. Здесь также будет заполнено текущим временем. Поле photo используется для загрузки картинки, конечно мы ее загружать из консоли не будем и пока тоже не будем трогать это поле и не смотря, что здесь не поставлен атрибут blank=True, это поле является обязательным для заполнения, но если мы работаем напрямую через консоль, то можем тоже пока не заполнять и Django сам будет формировать SQL-запрос и появление пустой строки, тоже будет являться значением. Поле is_published является

логическим полем и при создании будет проставляться 1. Итак, по факту будут заполнены только два поля title, Content.

Вызовем конструктор нашей модели

News(title='Новость 1', content='Контент новости 1')

```
>>> from news.models import News
>>> News(title='Новость 1', content='Контент новости 1')
<News: News object (None)>
>>>
```

Рисунок 4

В результате получим <News: News object (None)> (рисунок 4)

Давайте посмотрим таблицу в БД и после обновления кнопкой, то увидим, что данные не добавились. Это происходит так, потому что эти данные нужно сохранить. На данный момент мы просто заполнили конструктор данными, но не выгрузили в таблицу. За сохранение данных отвечает специальный метод модели, это метод save. И здесь стоит обратить внимание на то, что модели по умолчанию в Django являются «ленивыми». Это означает, что мы можем формировать sql запрос с помощью API модели, можем делать в нескольких строках кода, вызывая методы и с каждой строкой ни чего происходит не будет до тех пор пока не запустим запрос на исполнение. И в случае записи данных делается это с помощью метода save, в случае чтения данных это делается в момент запроса этих самых данных. Т.е. сами запросы сами по себе не выполняются и выполняются только тогда, когда нужен результат этого запроса.

В нашем случае нужно сохранить эти данные, поэтому давайте создадим некоторую переменную news1 (news1=_) и присвоим ей результат предыдущего выражения в консоли с помощью псевдопеременной с нижним подчеркиванием, в которой хранится предыдущий результат.

```
>>> news1=_
>>>
```

Рисунок 5

И если выведем эту переменную, то в ней буде объект ранее полученный

```
>>> news1
<News: News object (None)>
>>>
```

Рисунок 6

Non – это номер объекта, т.е. id, который пока что у нас его нет и не будет до тех пор пока запись не будет сохранена. Id назначается автоматически.

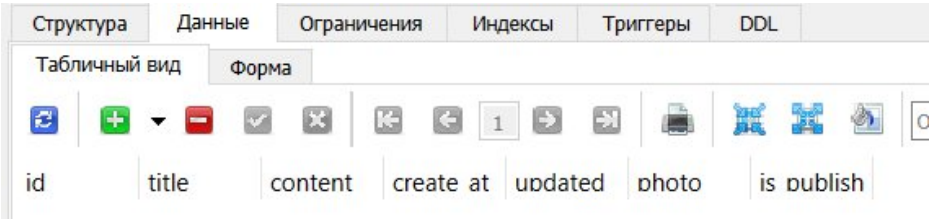


Рисунок 7

Но все свойства, например, news1.title у нас есть (рисунок 7)

```
>>> news1.title
'Новость 1'
>>>
```

Рисунок 8

Или news1.content

```
>>> news1.content
'Контент новости 1'
```

Рисунок 9

Если обратимся к полю id, то увидим пустую строку, после ввода команды (рисунок 10)

```
>>> news1.id
>>>
```

Рисунок 10

Сохраним эту запись с помощью метода save(): news1.save().

```
>>> news1.save()
>>> news1.id
1
```

Рисунок 11

После ввода этой команды сохранения и вывода, например, id, мы увидим, что поле id уже имеет свое уникальное значение «1». Это означает, что Django сохранил эту запись.

Если мы обновим запись в SQLiteStudio, то увидим уже эту запись в БД (рисунок 12)



Рисунок 12

Поле title и content заполнились нашими данными, поле create_at и update_at заполнились текущими данными, поле is_publish – заполнилось логической 1, признаком публикации, поле photo – пустое.

Можно просмотреть выполненные запросы. Для этого на нужно из модуля Django.db импортировать connection и просмотреть его свойства queries

from django.db import connection

connection.queries



Рисунок 13

В результате мы получаем список словарей с двумя ключами ‘sql’ – это запрос и ‘time’ – это время, которое потребовалось на его выполнение.

У news появилось такое свойство как pk, которое равно свойству id.

```
>>> news1.pk
1
>>>
```

Рисунок 14

В Django многое работает на уровне соглашения, т.е. так называемые рекомендации как именовать то или иное выражение и в том числе, есть такое свойство pk (сокращение от слов primary key) и это то, как мы можем обращаться к полю id через универсальное свойство pk и хранить в себе ключевое значение записи.

Создадим и сохраним еще одну запись:

news2=News(title='Новость 2', content='Контент новости 2')

```
>>> news2=News(title='Новость 2', content='Контент новости 2')
>>> news2.save()
>>>
```

Рисунок 15

```
>>> connection.queries
[{'sql': 'INSERT INTO "news_news" ("title", "content", "create_at", "updated", "photo", "is_published") VALUES (\'Новость 1\', \'Контент новости 1\', \'2023-03-12 21:58:29.878317\', \'2023-03-12 21:58:29.878317\', \'\', 1) RETURNING "news_news"."id"', 'time': '0.000'}, {'sql': 'INSERT INTO "news_news" ("title", "content", "create_at", "updated", "photo", "is_published") VALUES (\'Новость 2\', \'Контент новости 2\', \'2023-03-12 22:32:38.842812\', \'2023-03-12 22:32:38.842812\', \'\', 1) RETURNING "news_news"."id"', 'time': '0.016'}]
>>>
```

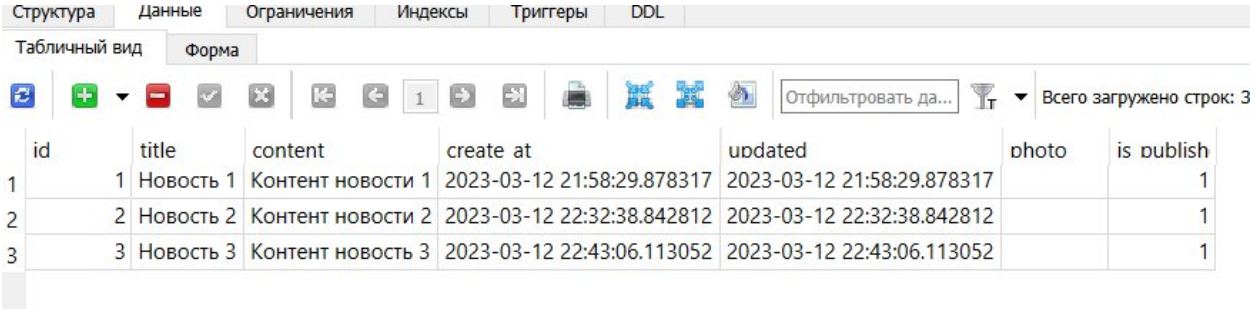
Рисунок 16

Для очистки консоли можно воспользоваться командой Ctrl+L.

Рассмотрим еще один вариант добавления записи. Можно вызвать модель без параметров и дальше заполнить ее параметрами. (рисунок 17)

```
Terminal:  Local  Local (2)  +
>>> news3=News()
>>> news3.title='Новость 3'
>>> news3.content='Контент новость 3'
>>> news3.save()
```

Рисунок 17


Структура Данные Ограничения Индексы Триггеры DDL

Табличный вид Форма

Отфильтровать да... Всего загружено строк: 3

	id	title	content	create at	updated	photo	is publish
1	1	Новость 1	Контент новости 1	2023-03-12 21:58:29.878317	2023-03-12 21:58:29.878317		1
2	2	Новость 2	Контент новости 2	2023-03-12 22:32:38.842812	2023-03-12 22:32:38.842812		1
3	3	Новость 3	Контент новость 3	2023-03-12 22:43:06.113052	2023-03-12 22:43:06.113052		1

Рисунок 18

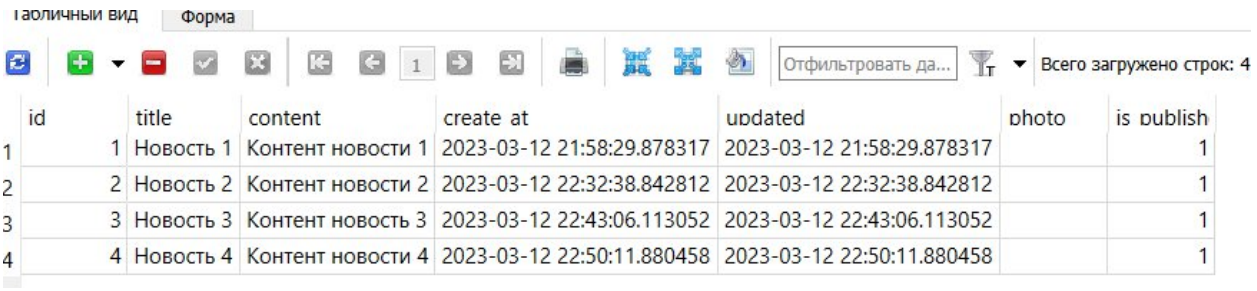
После обновления мы увидим в таблице уже три записи.

Рассмотри третий вариант добавления записи. Для этого воспользуемся атрибутом класса Objects. Данный атрибут хранит диспетчер записей и позволяет манипулировать записями модели. Через этот диспетчер мы можем не только сохранять записи, но и получать их от туда. Данный диспетчер является экземпляром специального класса manader.

news4=News.objects.create(title='Новость 4', content='Контент новости 4')

После нажатия Enter, запись будет автоматически сохранена и не нужно будет вызывать метод save(). Убедимся, если обратимся к свойству pk

```
>>> news4=News.objects.create(title='Новость 4', content='Контент новости 4')
>>> news4.pk
4
>>> ■
```


Табличный вид Форма

Отфильтровать да... Всего загружено строк: 4

	id	title	content	create at	updated	photo	is publish
1	1	Новость 1	Контент новости 1	2023-03-12 21:58:29.878317	2023-03-12 21:58:29.878317		1
2	2	Новость 2	Контент новости 2	2023-03-12 22:32:38.842812	2023-03-12 22:32:38.842812		1
3	3	Новость 3	Контент новость 3	2023-03-12 22:43:06.113052	2023-03-12 22:43:06.113052		1
4	4	Новость 4	Контент новости 4	2023-03-12 22:50:11.880458	2023-03-12 22:50:11.880458		1

Рисунок 19

Метод create() создает запись и сразу ее возвращает.

Давайте создадим еще одну запись и посмотрим как возвращает функция create(). Создадим новую запись следующего вида

News.objects.create(title='Новость 5', content='Контент новости 5')

И после нажатия Enter, мы видим <News: News object (5)>

```
>>> News.objects.create(title='Новость 5', content='Контент новости 5')
<News: News object (5)>
>>>
```

Рисунок 20

Как на рисунке 20 мы видим, что нам вернулось не None, а значение 5.

Таким образом, мы рассмотрели операцию create технологии CRUD. Перейдем к рассмотрению операции read, т.е. получение данных как одна из наиболее частых операций.

Для получения всех записей используется метод all(). Это диспетчер записи атрибута Objects, который вернет набор записей, являющихся экземпляром класса QuerySet.

При этом не забудьте активировать интерактивную консоль InteractiveConsole.

Попробуем получить все записи из модели News.

News.objects.all()

Модель вернет все наши записи (рисунок 21)

```
>>> News.objects.all()
<QuerySet [<News: News object (1)>, <News: News object (2)>, <News: News object (3)>, <News: News object (4)>, <News: News object (5)>]>
>>>
```

Рисунок 21

В таком формате по Django мы видим только название объекта и его id. Необходимо посмотреть с значения и здесь будем использовать метод, который возвращает строковое представление объекта. Т.е. выведем записи, в нашем случае, с названиями новостей.

В файле molels.py опишем этом метод с помощью функции (рисунок 22)

```
    def __str__(self):
return self.title
```



Рисунок 22

И теперь при распечатке данного объекта будет работать метод __str__(), которое возвращает строковое представление объекта, в нашем примере, это название новости.

И попробуем еще раз выполнить команду News.objects.all(), но при этом изменения вступят в силу нужно выйти из консоли Django с помощью команды exit() (рисунок 23)

```
<QuerySet [<News: News object (1)>, <News: News object (2)>, <News: News objec
>>> exit()

(venv) C:\Users\Елена\PycharmProjects\pythonProject29\MySite>
```

Рисунок 23

И еще раз зайти python manage.py shell

После перезагрузки снова импортируем модель

from news.models import News

и записываем News.objects.all() (рисунок 24)

```
>>> News.objects.all()
<QuerySet [<News: Новость 1>, <News: Новость 2>, <News: Новось 3>, <News: Новость 4>, <News: Новость 5>]>
>>>
```

Рисунок 24

Как видим, вывелись записи с названиями новостей.

Поскольку был получен набор значений то его можно обработать в цикле. Сохраним этот набор в переменную news=_

```
>>> news=_
>>> news
<QuerySet [<News: Новость 1>, <News: Новость 2>, <News: Новось 3>, <News: Новость 4>, <News: Новость 5>]>
>>>
```

Рисунок 25

Например, нам необходимо вывести список новостей с признаком опубликована новость или нет (рисунок 26).

```
for item in news:
    print(item.title, item.is_published)
```

```
>>> for item in news:
...     print(item.title, item.is_published)
...
Новость 1 True
Новость 2 True
Новось 3 True
Новость 4 True
Новость 5 True
>>>
```

Рисунок 26

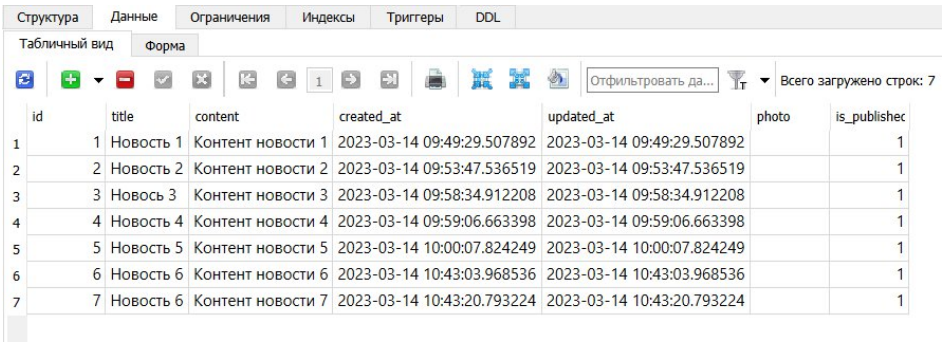
Но чаще всего весь список может быть и не нужен, а нужны по какому-нибудь критерию записи. Для этого используется метод filter().

Предварительно добавим пару записей с одинаковыми заголовками, но с разным контентом (рисунок 27).

```
>>> News.objects.create(title='Новость 6', content='Контент новости 6')
<News: Новость 6>
>>> News.objects.create(title='Новость 6', content='Контент новости 7')
<News: Новость 6>
>>>
```

Рисунок 27

И обновим БД (рисунок 28)



Структура | Данные | Ограничения | Индексы | Триггеры | DDL

Табличный вид | Форма

Отфильтровать да... | Всего загружено строк: 7

	id	title	content	created_at	updated_at	photo	is_publishec
1	1	Новость 1	Контент новости 1	2023-03-14 09:49:29.507892	2023-03-14 09:49:29.507892		1
2	2	Новость 2	Контент новости 2	2023-03-14 09:53:47.536519	2023-03-14 09:53:47.536519		1
3	3	Новось 3	Контент новости 3	2023-03-14 09:58:34.912208	2023-03-14 09:58:34.912208		1
4	4	Новость 4	Контент новости 4	2023-03-14 09:59:06.663398	2023-03-14 09:59:06.663398		1
5	5	Новость 5	Контент новости 5	2023-03-14 10:00:07.824249	2023-03-14 10:00:07.824249		1
6	6	Новость 6	Контент новости 6	2023-03-14 10:43:03.968536	2023-03-14 10:43:03.968536		1
7	7	Новость 6	Контент новости 7	2023-03-14 10:43:20.793224	2023-03-14 10:43:20.793224		1

Рисунок 28

И теперь вернем список записей с названием Новость 6 с использованием метода filter() (рисунок 29).

```
>>> News.objects.filter(title='Новость 6')
<QuerySet [<News: Новость 6>, <News: Новость 6>]>
>>>
```

Рисунок 29

И теперь, как видим, QuerySet состоит только из двух нужных нам записей.

В других случаях, может возникнуть необходимость вывода только одной записи, которую как правило выбирают по id с помощью метода get().

News.objects.get(pk=5)

```
>>> News.objects.get(pk=5)
<News: Новость 5>
>>>
```

Рисунок 30

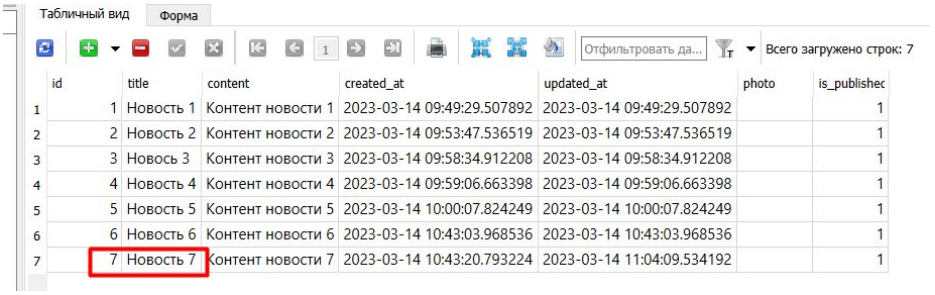
Следующий процесс – это обновление записи.

Чтобы обновить запись необходимо ее получить. Давайте обновим запись с id=7 и переименуем название новости (рисунок 31).

```
>>> news7=_
>>> news7
<News: Новость 7>
>>> news7.title='Новость 7'
>>> news7.save()
>>>
```

Рисунок 31

Обновляем БД (рисунок 32).



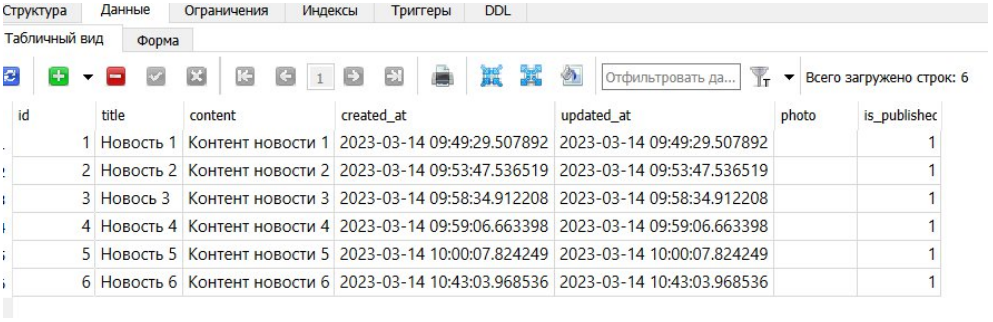
	id	title	content	created_at	updated_at	photo	is_publishec
1	1	Новость 1	Контент новости 1	2023-03-14 09:49:29.507892	2023-03-14 09:49:29.507892		1
2	2	Новость 2	Контент новости 2	2023-03-14 09:53:47.536519	2023-03-14 09:53:47.536519		1
3	3	Новось 3	Контент новости 3	2023-03-14 09:58:34.912208	2023-03-14 09:58:34.912208		1
4	4	Новость 4	Контент новости 4	2023-03-14 09:59:06.663398	2023-03-14 09:59:06.663398		1
5	5	Новость 5	Контент новости 5	2023-03-14 10:00:07.824249	2023-03-14 10:00:07.824249		1
6	6	Новость 6	Контент новости 6	2023-03-14 10:43:03.968536	2023-03-14 10:43:03.968536		1
7	7	Новость 7	Контент новости 7	2023-03-14 10:43:20.793224	2023-03-14 11:04:09.534192		1

Рисунок 32

Рассмотрим теперь процесс удаления delete(). Для примера удалим запись с id 7 (рисунок 33).

```
>>> news7=News.objects.get(pk=7)
>>> news7.delete()
(1, {'news.News': 1})
>>>
```

Рисунок 33



	id	title	content	created_at	updated_at	photo	is_publishec
1	1	Новость 1	Контент новости 1	2023-03-14 09:49:29.507892	2023-03-14 09:49:29.507892		1
2	2	Новость 2	Контент новости 2	2023-03-14 09:53:47.536519	2023-03-14 09:53:47.536519		1
3	3	Новось 3	Контент новости 3	2023-03-14 09:58:34.912208	2023-03-14 09:58:34.912208		1
4	4	Новость 4	Контент новости 4	2023-03-14 09:59:06.663398	2023-03-14 09:59:06.663398		1
5	5	Новость 5	Контент новости 5	2023-03-14 10:00:07.824249	2023-03-14 10:00:07.824249		1
6	6	Новость 6	Контент новости 6	2023-03-14 10:43:03.968536	2023-03-14 10:43:03.968536		1

Рисунок 34

После обновления БД видим, что этой записи уже нет.

Рассмотрим еще метод exclude() – исключение не нужных записей.

News.objects.exclude(title='Новость 4')

```
>>> News.objects.exclude(title='Новость 4')
<QuerySet [<News: Новость 1>, <News: Новость 2>, <News: Новось 3>, <News: Новость 5>, <News: Новость 6>]>
>>>
```

Рисунок 35

Как видим из рисунка 35, запись 'Новость 4' была исключена

Лабораторная работа № 4.

Шаблоны. Админка Django

Выведем данные, полученные в консоле в браузере. Для этого запустим сервер и откроем главную страницу с “Hello world”.

Необходимо получить данные из модели. Для этого перейдем в папку views.py и импортируем модель (рисунок 1).

```
from .models import News
```

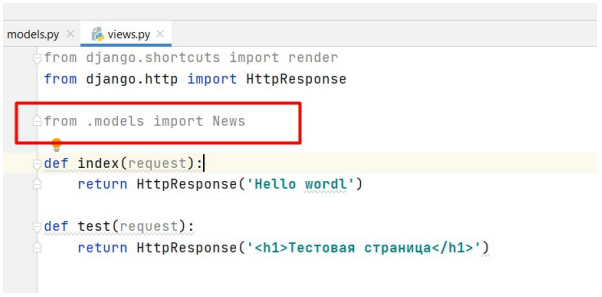


Рисунок 1

Теперь мы готовы получить эти данные через создаваемую переменную news и соответственно обратимся к модели News менеджеру объектов и методом all() получим все новости (рисунок 2).

```
news = News.objects.all()
```

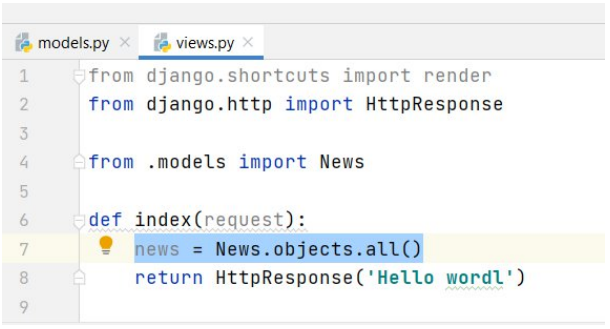


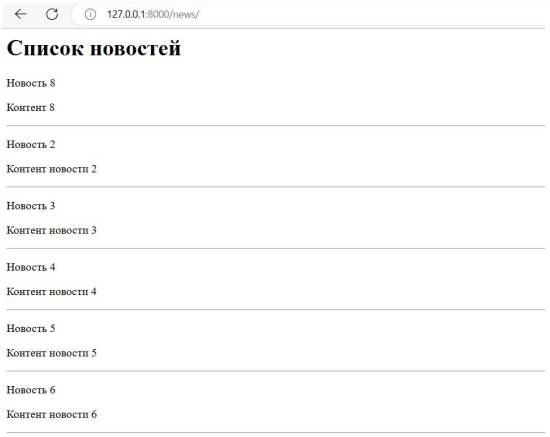
Рисунок 2

Теперь сформируем в следующей строке контроллера index ответ, который будем показывать пользователю (рисунок 3)

```
def index(request):
    news = News.objects.all()
    res = '<h1>Список нововстей</h1>'
    for item in news:
        res+=f'<div>\n<p>{item.title}</p>\n<p>{item.content}</p>\n</div>\n<hr>'
    return HttpResponse(res)
```

Рисунок 3

Обновляем страницу (рисунок 4)



Можно посмотреть исходный код (рисунок 5) (правая кнопка мыши выбрать в меню Просмотр исходного кода (рисунок 4).

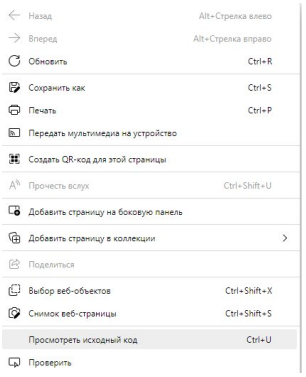


Рисунок 4



Рисунок 5

Видно, что здесь не хватает стандартных тегов <heat> <html>, <body> и др.

Конечно, мы это можем дописать все эти основные теги в строках цикла, но тогда получился бы очень большой код. А еще если подключать стили, скрипты, то это еще больше увеличило код этого контроллера.

Чтобы этого избежать, используются для этого шаблоны. Шаблоны, которые мы можем рендерить.

Рендеринг – процесс, наполнения файла данными из модели, т.е. формирование страницы с данными. Django позволяет рендерить с шаблона и работает с помощью специального шаблонизатора.

Это можно посмотреть в документации Django (https://django.fun/ru/) (рисунок 6)

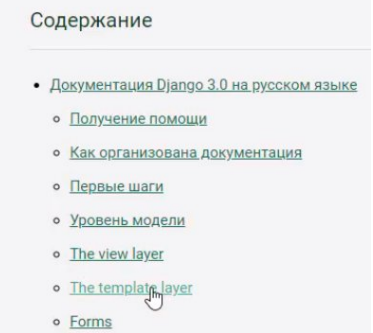


Рисунок 6

Шаблонов много с их описанием можно познакомится например, на русско-язычном сайте (рисунок 7).

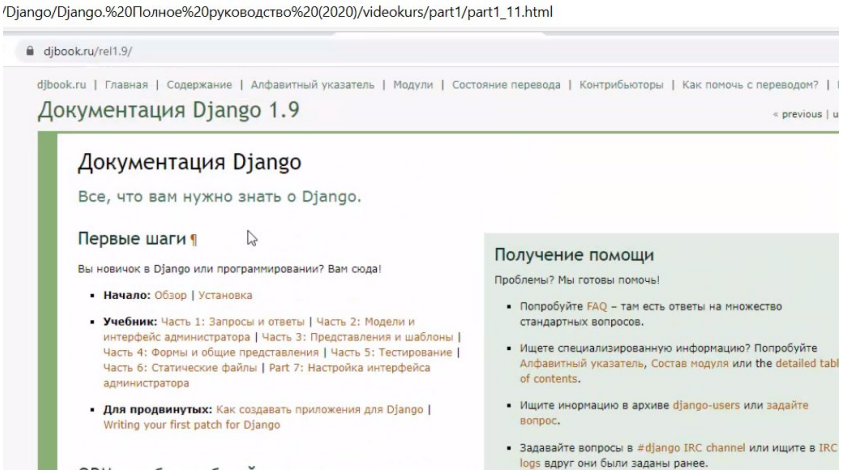


Рисунок 7

Когда мы работаем с шаблонизатором, то мы по сути работаем в нем с тремя объектами: директивы, теги, фильтры.

Рассмотрим пока некоторые базовые вещи.

По умолчанию Django ищет шаблоны в папке templates в соответствующем приложении. По умолчанию эта папка не создаются. Поэтому ее необходимо создать (рисунок 8, 9).

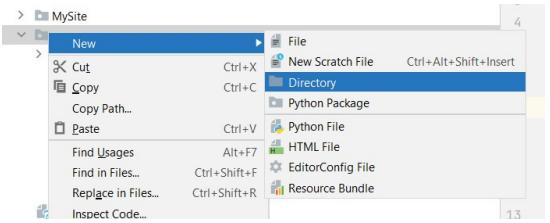


Рисунок 8

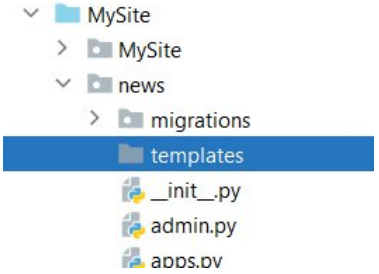


Рисунок 9

Django очень гибко подходит к работе с шаблонами и их можно помещать куда угодно, это все настраивается и Django будет искать там где будет указано. Но Django рекомендует внутри папки templates создать еще одноименную с приложением папку, в нашем случае, news и туда складывать шаблоны (рисунок 10).

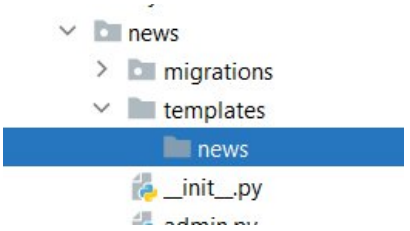


Рисунок 10

Почему рекомендуется именно так? На этапе компиляции, т.е. сбора приложения, Django собирает все шаблоны в одну кучу (локализует) в одном месте и будет указывать первый шаблон, указанный для рендеринга. Но если, например, создадим контроллер index в другом приложении, подобный индексу в приложении news, то возникнет проблема конфликта, и чтобы

этого не было, будем использовать некоторое пространство имен, которым является еще один подкаталог.

Шаблон – это обычный HTML файл. Создадим его и назовем index

Как правило, шаблон создается по названию action (т.е. по названию того контроллера, который возвращает нам ответ (рисунок 11).

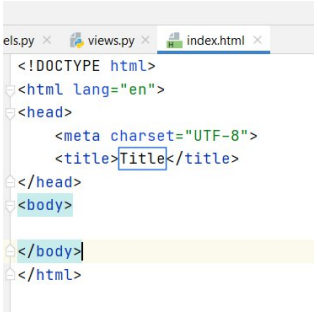


Рисунок 11

Отредактируем этот файл в соответствии с рисунком

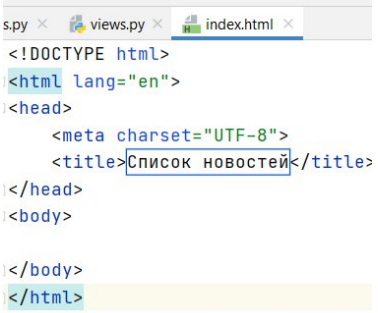


Рисунок 12

Для того, чтобы подключить шаблон, т.е. провести рендеринг, мы должны использовать специальную функцию из модуля shortcuts (рисунок 13)

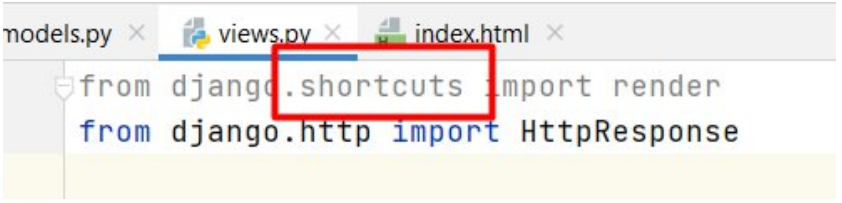


Рисунок 13

Из этого модуля импортируется функция render.

Изменим контроллер index вплоть до значения return, который будет возвращать render.

Render возвращает следующие параметры: параметр request наш объект запроса, название шаблона, который будет использоваться при загрузке данных и также принимает контекст.

Контекст – это словарь, с данными, которые необходимо передать в шаблон, в нашем случае news. Ключ в словаре можно называть любым именем, но для прозрачности кода, лучше задать его одноименным (рисунок 14).

```
def index(request):
    news = News.objects.all()
    return render(request, 'news/index.html',{'news': news, 'title': 'Список новостей'})
```

Рисунок 14

Снова перейдем в наш шаблон.

Как говорилось, шаблонизатор Django предлагает три вещи: директивы, теги, фильтры.

Директивы – это по сути переменные, которые позволяют выводить значения, которые передали в шаблон. В нашем случае это

```
'news': news, 'title': 'Список новостей'
```

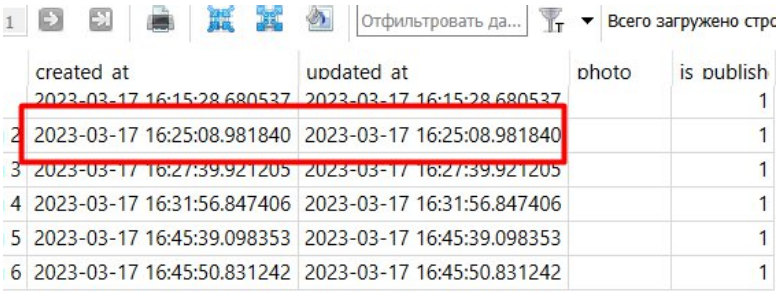
И здесь этими переменными являются 'news' и 'title'

В 'title' будет строка, в 'news' – список, который нужно итерировать. Для того, чтобы итерировать нам понадобятся не только директивы, но и теги.

Теги позволяют производить какие-то действия, они управляют генерированием вывода. Например, есть тег for, if и т.д. По сути, они дублируют синтез Python.

Фильтры – занимаются тем, что выполняют преобразованием перед выводом.

Например, если посмотрим на поле date, то увидите не очень понятный формат (рисунок 15)



	created at	updated at	photo	is publish
	2023-03-17 16:15:28.680537	2023-03-17 16:15:28.680537		1
2	2023-03-17 16:25:08.981840	2023-03-17 16:25:08.981840		1
3	2023-03-17 16:27:39.921205	2023-03-17 16:27:39.921205		1
4	2023-03-17 16:31:56.847406	2023-03-17 16:31:56.847406		1
5	2023-03-17 16:45:39.098353	2023-03-17 16:45:39.098353		1
6	2023-03-17 16:45:50.831242	2023-03-17 16:45:50.831242		1

Рисунок 15

Поэтому необходимо вывести его в более читаемом формате. И в этом случае используются фильтры.



Рисунок 16

И обновим страницу в браузере (рисунок 17).



Рисунок 17

Как мы видим, что вывод не совсем в «читабельном» виде и это конечно, необходимо исправить. И это можно исправить с помощью тегов, которые управляют генерированием вывода. Например, используем тег for, который организует обычный цикл по объектам.

Для того чтобы его создать, используется следующая запись

```
{%                              block                              %}

{% endblock %}
```

Где под block понимается какой-либо тег, реализующий вывод. Например, тег for (рисунок 18).

```
<body>
<h1>{{title}}</h1>

{% for item in news %}
    <div>
        <p>{{item.title}}</p>
        <p>{{item.content}}</p>
        <p>{{item.created_at}}</p>
    </div>
<hr>
{% endfor %}

</body>
```

Рисунок 18

Обновим страницу браузера и увидим уже красивый формат вывода (рисунок 19)

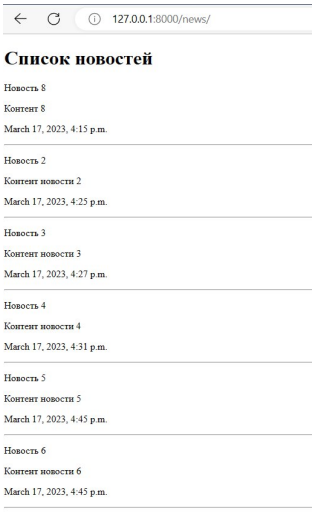


Рисунок 19

Но мы видим, что дата представлена в англо-язычном формате. Для рядового пользователя такой формат времени может быть не понятным. Поэтому внесем поправки. Это можно сделать двумя способами.

Во-первых, можно обратиться к настройкам нашего приложения settings (рисунок 20).

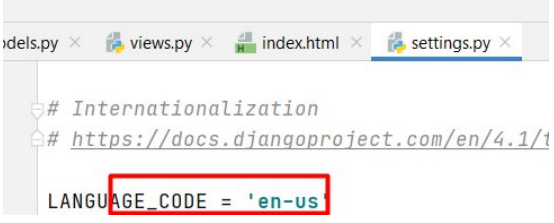


Рисунок 21

Это означает, что приложение использует этот язык и все настройки локализации отталкиваются от этого языка. Чтобы изменить на русский язык можно изменить на ru (рисунок 22).

```
# Internationalization
# https://docs.djangoproject.com/

LANGUAGE_CODE = 'ru'
```

Рисунок 23

После обновления видим список новостей в удобном для нас формате (рисунок 24)



Рисунок 24

Рассмотрим другой вариант, более гибкий с помощью фильтров (рисунок 25).

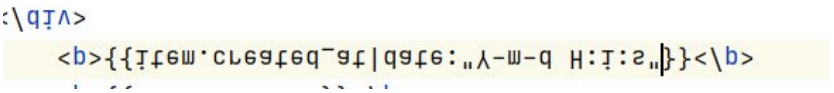


Рисунок 25

После обновления видим улучшенный вариант(рисунок 26)

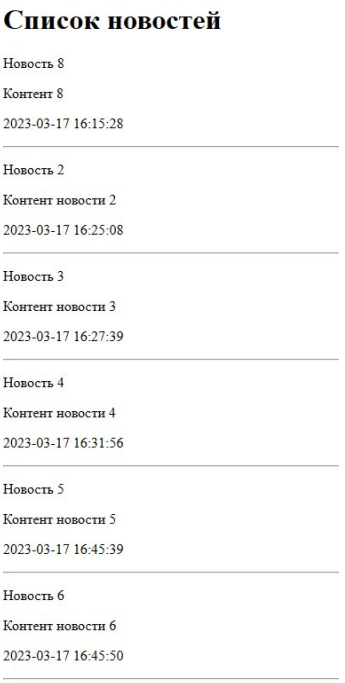


Рисунок 26

Создавая ответ строкой

```
(request, 'news/index.html',{'news': news, 'title': 'Список новостей'})
```

Третьим параметром, мы создавали контекст. Но если, контекста много, то в этом случае создают переменную, которая будет словарем содержать передаваемые значениям (рисунок 27).

```
def index(request):
    news = News.objects.all()
    context={
        'news': news,
        'title': 'Список новостей'
    }
    return render(request, 'news/index.html',context)
```

Рисунок 27

Здесь мы использовали позиционный документ, но можно использовать именованные аргументы (рисунок 28).

```
    'news': news,
    'title': 'Список новостей'
}
return render(request, template_name='news/index.html',context=context)
```

Рисунок 28

Таким образом, мы получили готовую HTML страницу с выводом данных модели.

Добавим еще одно действие, которое расположит новости по дате создания. Для этого в контроллере index заменим all() на order_by() и в нем указать, по какому полю мы хотим отсортировать в обратном порядке. Знак минус указывает на сортировку в обратном порядке `order_by('-created_at')` (рисунок 29).

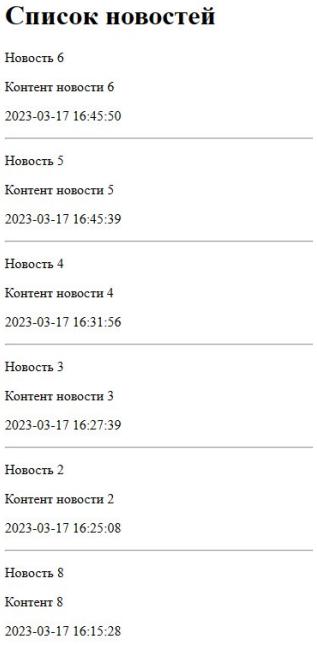


Рисунок 29

Перейдем теперь к работе с админкой.

Как известно, админка уже доступна по умолчанию.

Откроем стартовую страницу админки (рисунок 30) по адресу http://127.0.0.1:8000/admin

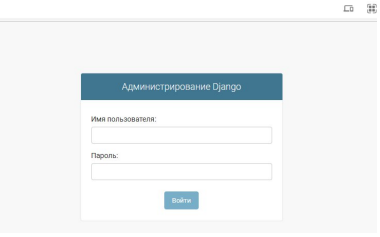


Рисунок 30

Как видно, необходимо авторизоваться, но для этого необходимо создать суперпользователя.

Для этого перейдем в консоль терминала и введем команду

python manage.py createsuperuser

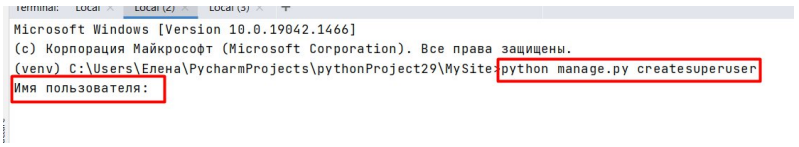


Рисунок 31

После ввода команды будет предложено ввести Имя пользователя, например, admin (совет, в реальных проектах использовать сложное имя). Назначаем имя суперпользователь нажимаем enter (рисунок 32).

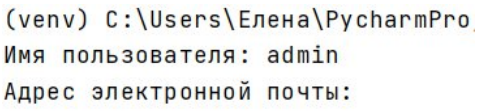


Рисунок 32

Далее будет предложено ввести адрес электронной почты, которое является необязательным. Введем для примера admin@localhost

Далее предлагается ввести пароль. Если вы введете короткий пароль, то Django сообщит вам об этом и даст совет о вводе более длинного пароля и предложит подтвердить уже введенный пароль введением Y или, если вы введете N, то получите возможность назначить другой более сложный пароль (рисунок 33).

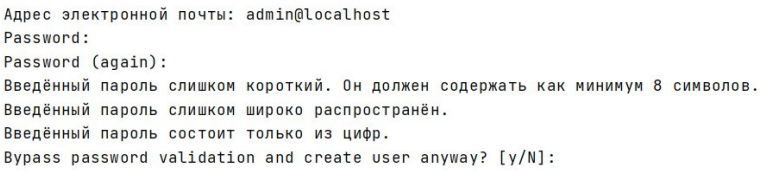


Рисунок 33

Обратите внимание, что при вводе пароля не отображаются символы ввода, для того чтобы посторонний пользователь не мог его подсмотреть.

После подтверждения пароля вы увидите подтверждение об успешном назначении этого пароля суперпользователю (рисунок 34).

```
Введённый пароль состоит только из цифр.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
```

Рисунок 34

Теперь давайте зайдем под этими данными в админку (рисунок 35).

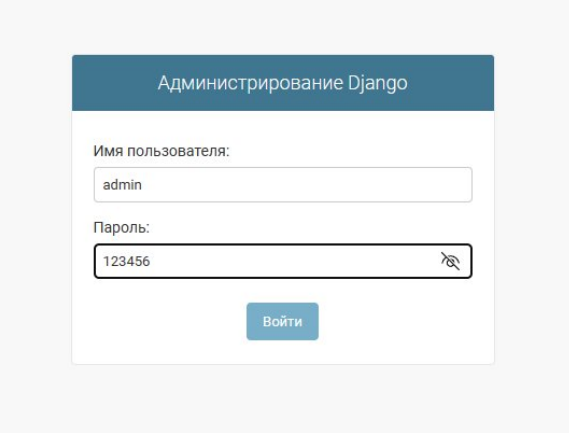


Рисунок 35

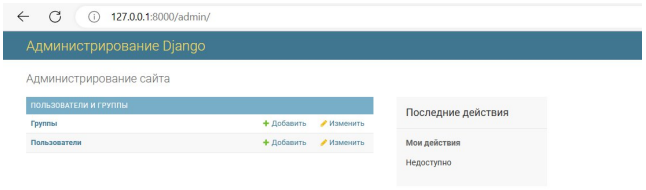


Рисунок 36

Как вы видите, админка здесь уже русифицирована. По умолчанию уже здесь зарегистрировано два приложения: Группа, Пользователи.

Нажав на знак + мы можем здесь добавлять пользователей, редактировать (рисунок 37)

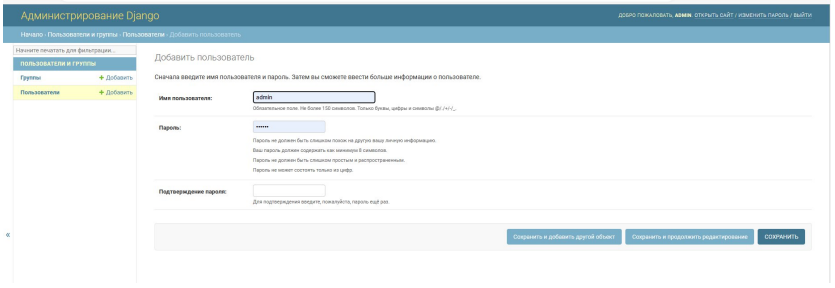


Рисунок 37

Можем добавлять группы (рисунок 38)



Рисунок 38

Назначать роли пользователям и группам (рисунок 39)

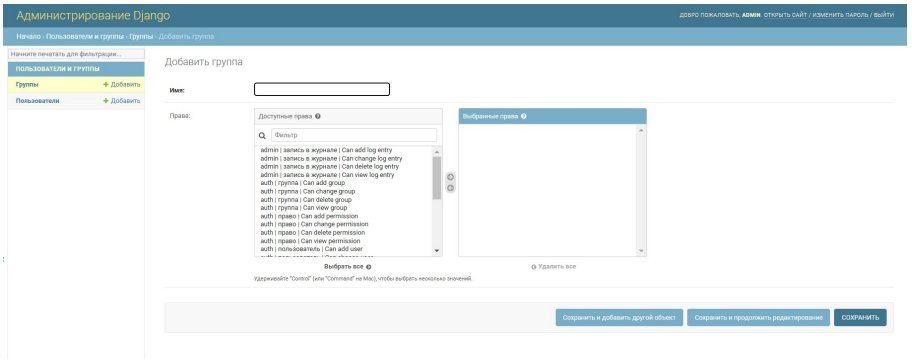


Рисунок 39

Справа в панели админки вы видите список действий (рисунок 40)

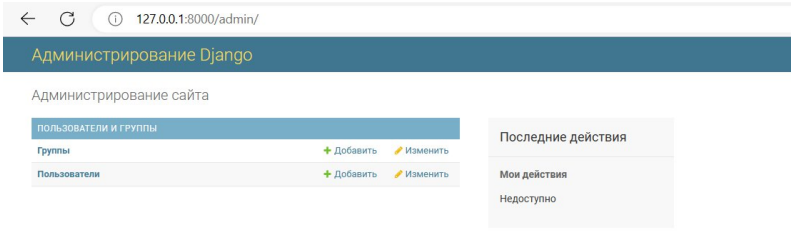


Рисунок 40

Пока конечно он пустой, поскольку мы не выполняли никаких действий, и они не были зарегистрированы в журнале.

Здесь можно будет просматривать действия общего характера, так и для отдельного приложения.

Но здесь мы не видим нашего нового приложения news, потому что мы его не зарегистрировали для админки.

Для этого вернемся в наше приложение news и откроем файл admin.py. И зарегистрируем наше приложение.

В файле импортируем приложение и напишем строку регистрации

```
from .models import News

admin.site.register(News)
```

Регистрация реализуется методом register() и передаем ему название нашего модели. После обновления мы увидим в списках наше приложение (рисунок 41)



Рисунок 41

Django автоматически добавляет к названию регистрируемого приложения -s. Поэтому получилось немного странно, что приложение с двумя s.

Если откроете эту вкладку, то увидите список наших новостей (рисунок 42)

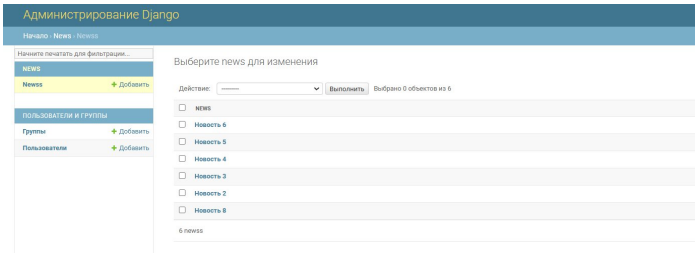


Рисунок 42

Здесь можно выбирать какие-то действия (рисунок 43).

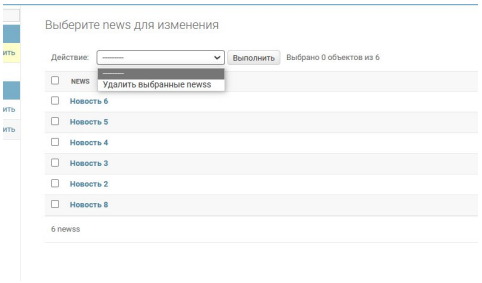


Рисунок 43

Можно добавить новую новость (рисунок 44)

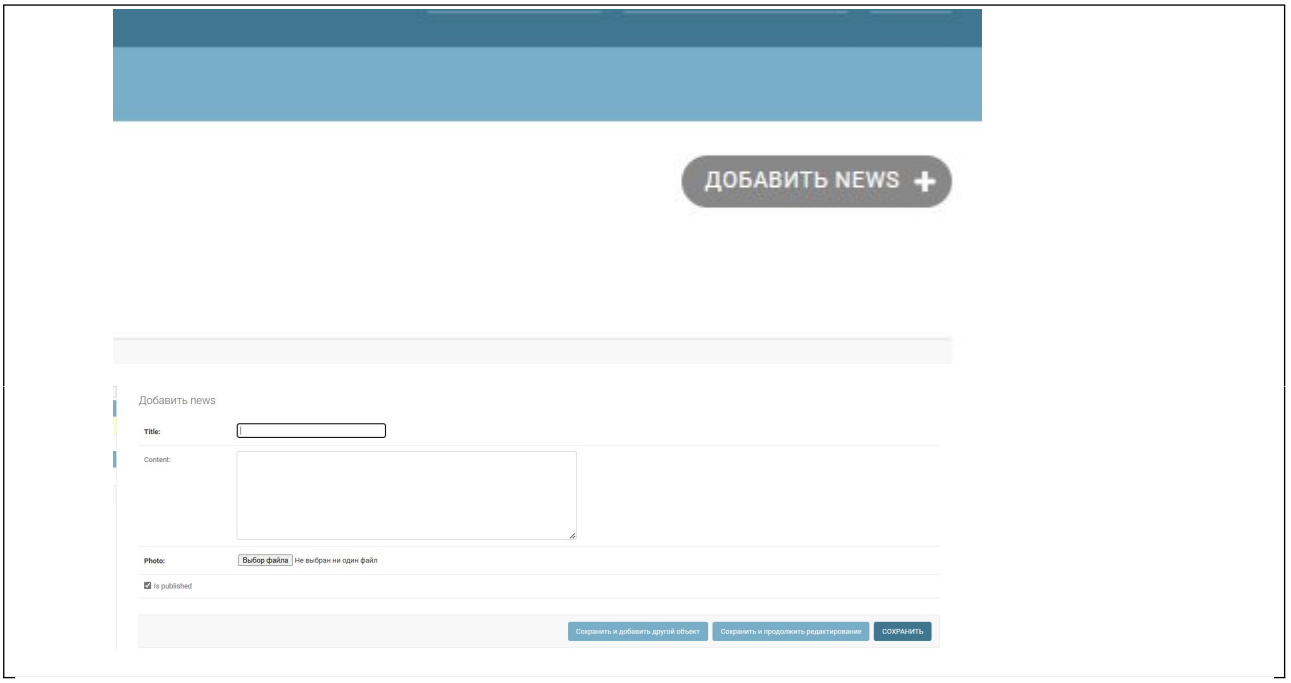


Рисунок 44

Можно отредактировать (рисунок 45)

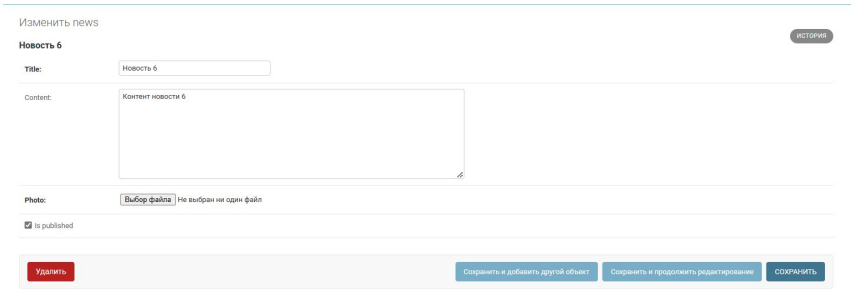


Рисунок 45

И конечно, провести настройки по свои пожелания. Например, назвать приложение более интересно, вывести другие поля, т.е., кроме title, вывести другие поля модели.

Все это делается в нашей модели. Т.е. переходим в файл models.py и создаем специальный подкласс Meta (рисунок 46)

```
    class                                                    Meta:
verbose_name='Новость'
```

Атрибут этого подкласса Meta `verbose_name` - это наименование модели в единственном числе.

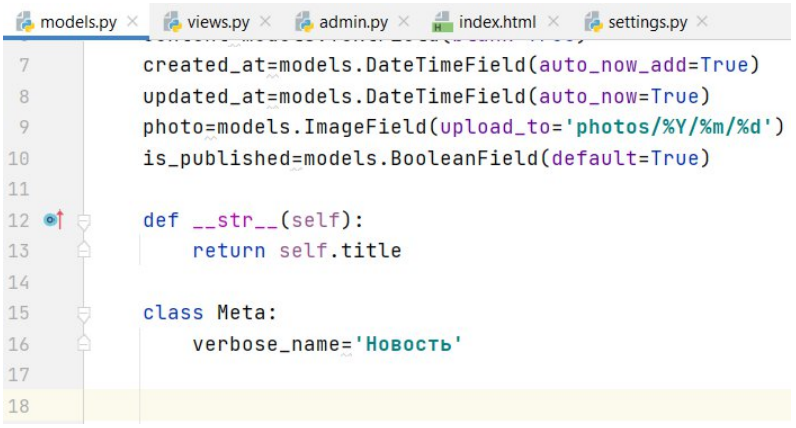


Рисунок 46

Давайте посмотрим результат в админке (рисунок 47).

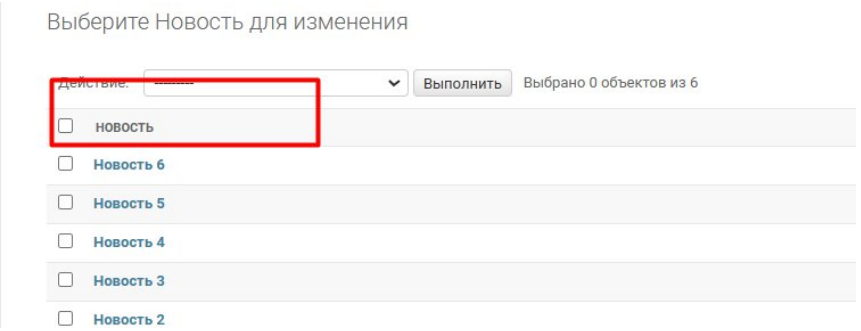


Рисунок 47

Перейдя на страницу изменения любой новости мы увидим обновлённый заголовок (рисунок 48).

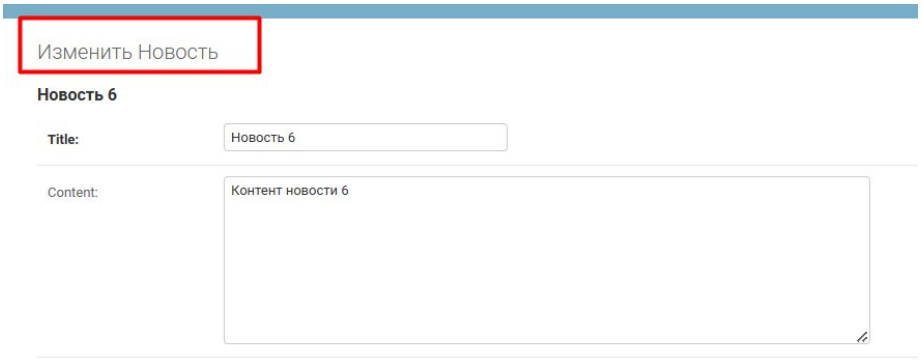


Рисунок 48

Следующий атрибут verbose_name_plural – это наименование модели во множественном числе (рисунок 49).

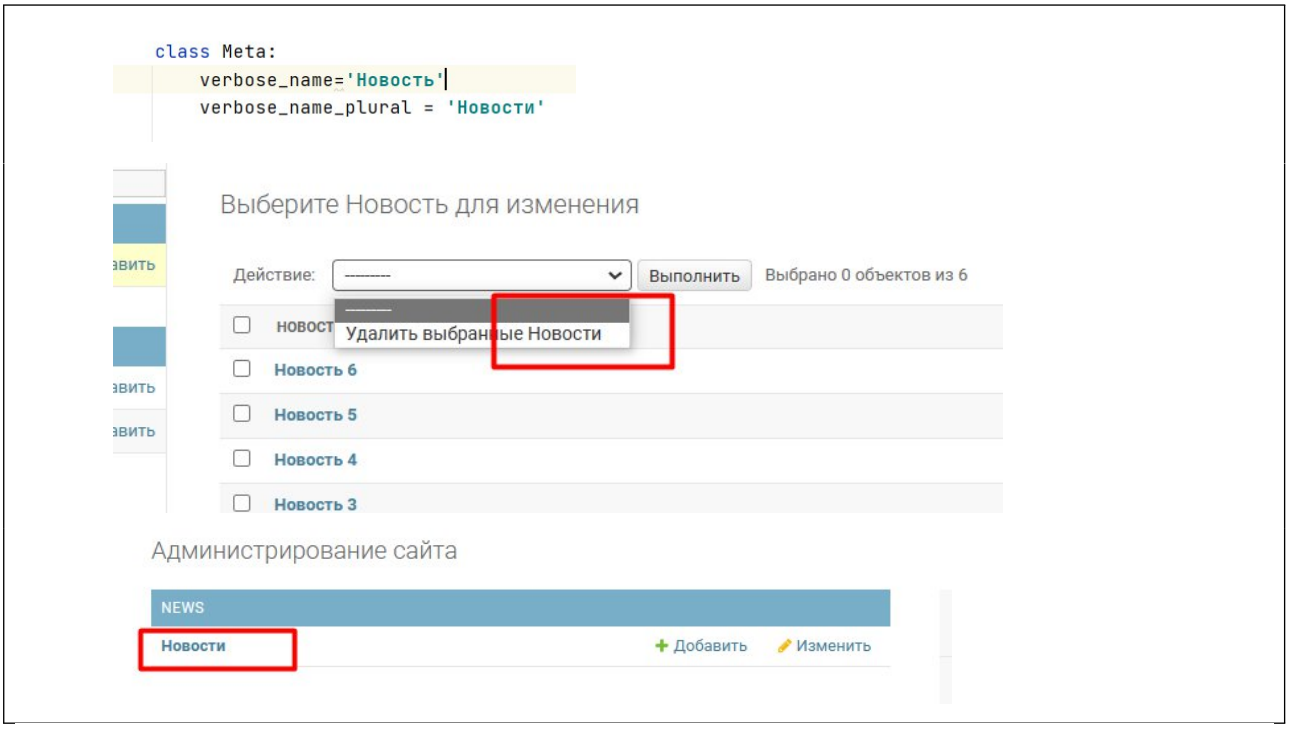


Рисунок 49

Можем поменять порядок следования новостей.

И это делается с помощью атрибута `ordering=['created_at',]`

Параметрами которого являются сортируемые поля (рисунок 50)



Рисунок 50

Этот порядок сортировки влияет не только на админку, но и на пользовательскую часть. И тогда во view.py можно не указывать порядок (рисунок 51)

```
def index(request):
    news = News.objects.order_by('-created_at')
    context={
```

```
def index(request):
    news = News.objects.all()
    context={
        'news': news,
        'title': 'Список новостей'
    }
```

Рисунок 51

И тогда порядок сортировки будет взят из модели.

Обновим страницу клиента (рисунок 52)

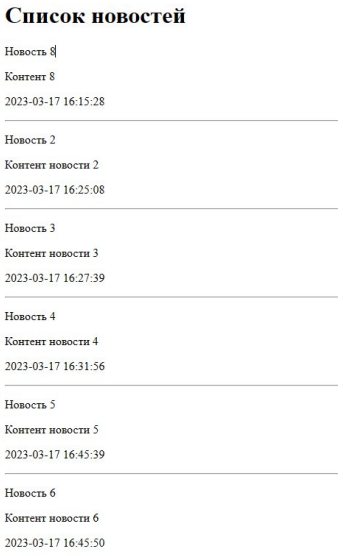


Рисунок 52

```
class Meta:
    verbose_name='Новость'
    verbose_name_plural = 'Новости'
    ordering=['-created_at',]
```

Рисунок 53

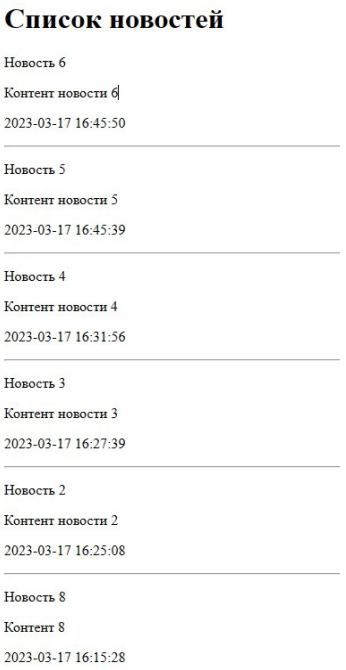


Рисунок 54

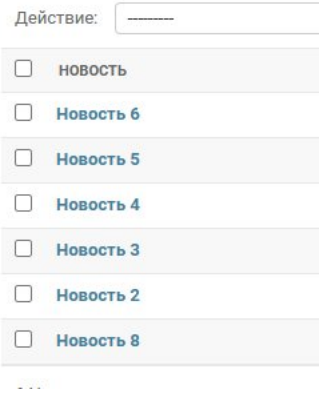


Рисунок 55

Поменяем еще слово NEWS на русский язык (рисунок 56)

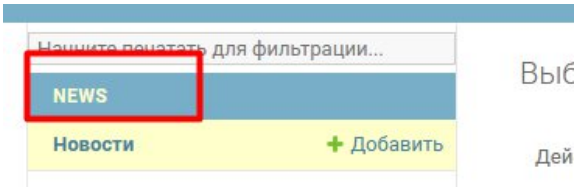


Рисунок 56

Здесь придёт на помощь конфигурация нашего приложения (рисунок 57, 58)

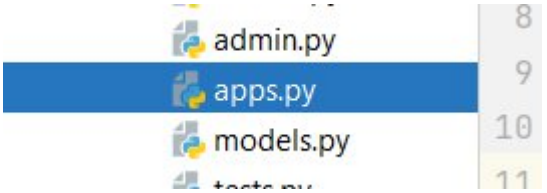


Рисунок 57

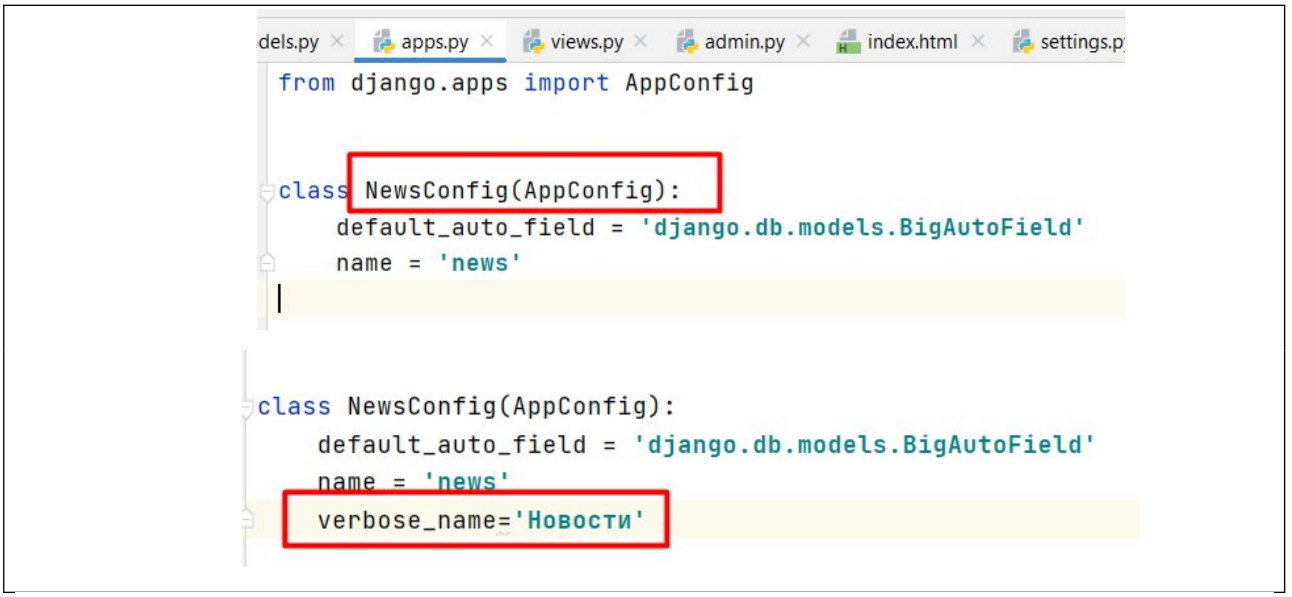


Рисунок 58

Обновим страницы браузера (рисунок 59)

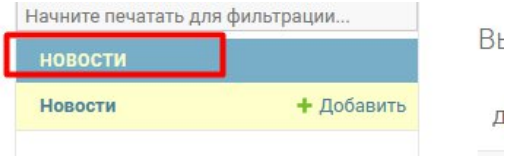


Рисунок 59

Следующим шагом добавим поля в списке новостей в админке. Это делается в admin.py с помощью класса редактора, в котором можно настроить представления модели в админке (рисунок 60).

```
class NewsAdmin(admin.ModelAdmin):
    list_display = ('id', 'title', 'created_at', 'updated_at','is_published')
```

Рисунок 60

Для того чтобы это вступило в действие нам необходимо его зарегистрировать вместе с News (рисунок 61).

```
class NewsAdmin(admin.ModelAdmin):
    list_display = ('id', 'title', 'created_at', 'updated_at','is_published')


admin.site.register(News, NewsAdmin)
```

Рисунок 61

Необходимо обратить внимание, что здесь важен порядок. Т.е. вначале регистрируем саму модель News, а потом ее настраиваем.

NewsAdmin- это настраивающий класс и конечно, логично вначале создать класс, а потом его настраивать.

Обновим страницу браузера админки (рисунок 62).

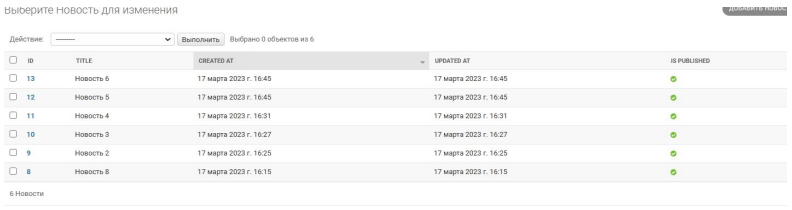


Рисунок 62

Как мы видим поля названы на английском, как в модели.

Эту ситуацию можно исправить, если передать в конструкторы, соответствующие аргументы (рисунок 63, 64).

```
class News(models.Model):
    title=models.CharField(max_length=150, verbose_name='Наименование')
    content=models.TextField(blank=True)
```

Рисунок 63

```
class News(models.Model):
    title=models.CharField(max_length=150, verbose_name='Наименование')
    content=models.TextField(blank=True, verbose_name='Контент')
    created_at=models.DateTimeField(auto_now_add=True, verbose_name='Дата публикации')
    updated_at=models.DateTimeField(auto_now=True, verbose_name='Обновлено')
    photo=models.ImageField(upload_to='photos/%Y/%m/%d', verbose_name='Фото')
    is_published=models.BooleanField(default=True, verbose_name='Опубликовано')
```

Рисунок 64

Обновим страницу браузера (рисунок 64)

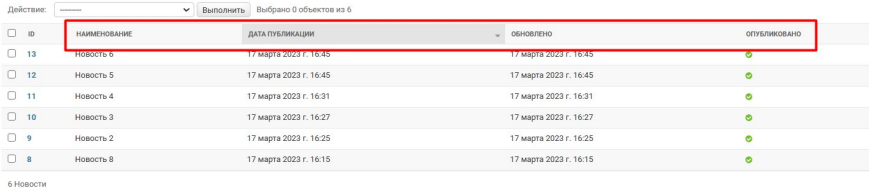


Рисунок 65

Для редактирования нам доступен id (рисунок 66)

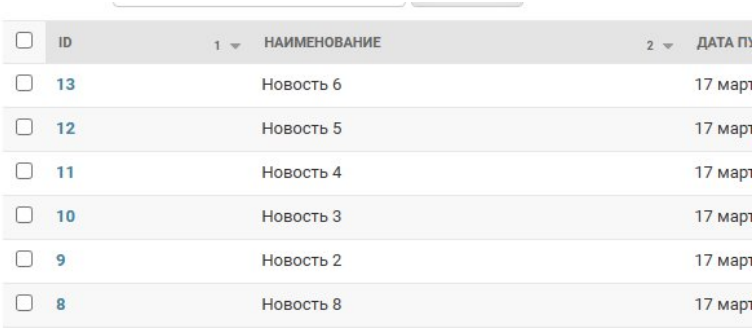


Рисунок 66

Если, необходимо сделать, чтобы ссылками были другие записи поля для их редактирования, то для этого необходимо обратиться к атрибуту (рисунок 67, 68) `list_display_links`

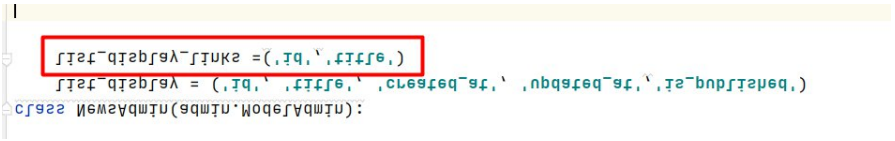


Рисунок 67

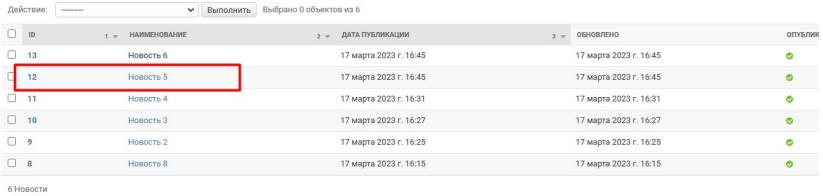


Рисунок 68

Настроим поля по которым можно проводить поиск. Для этого используется атрибут search_fields (рисунок 69)

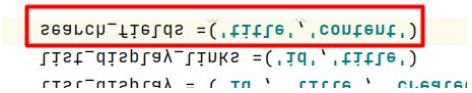


Рисунок 69

После обновления появится поле поиска (рисунок 70)

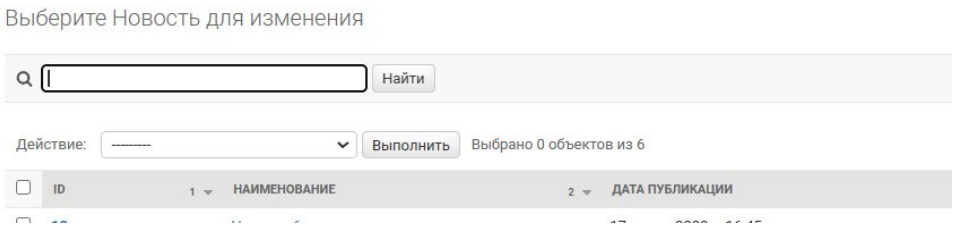


Рисунок 70

Давайте попробуем добавить новость из админки (рисунок 71).

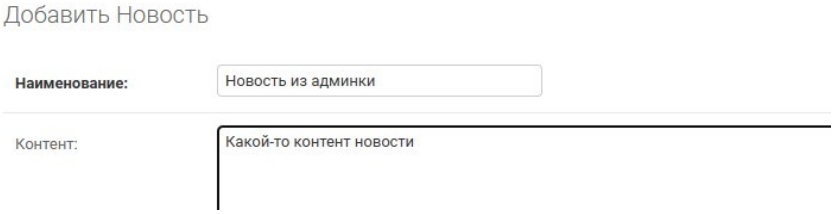


Рисунок 71

Если мы попробуем Сохранить, то Django выведет ошибку, потому что не было заполнено обязательное поле фотографии (рисунок 72).

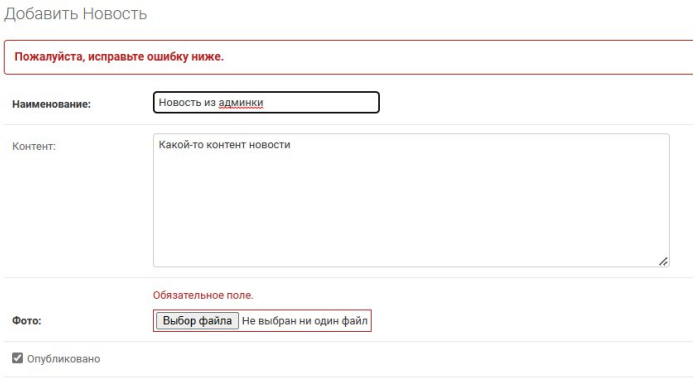
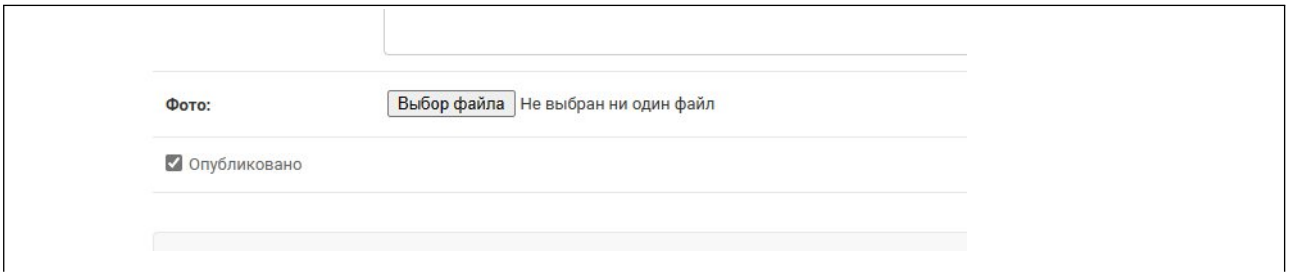


Рисунок 72

Добавим фотографию (рисунок 73)



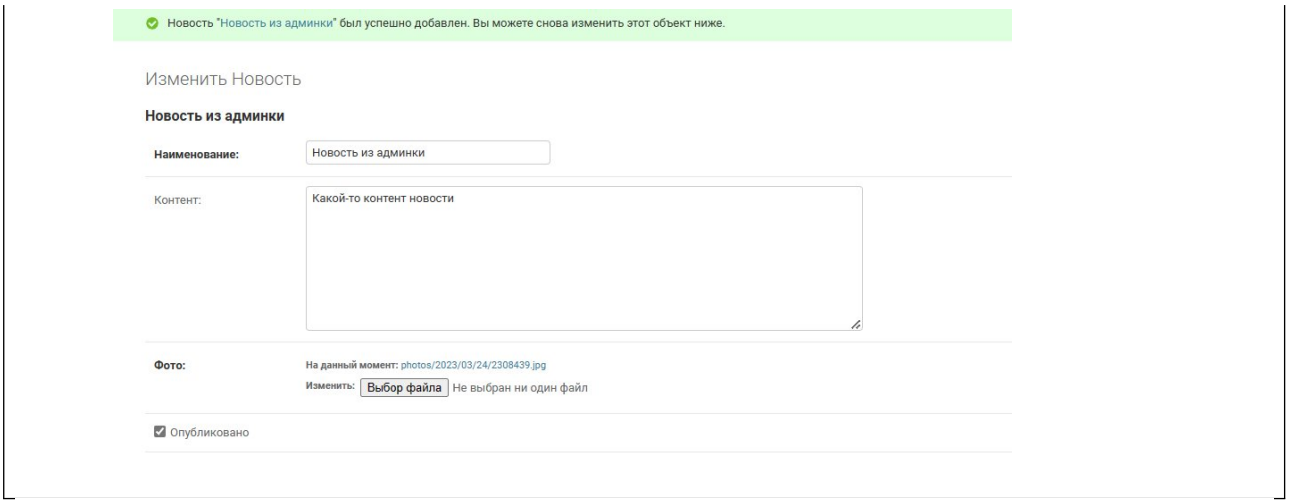


Рисунок 73

При этом в самом проекте появилась новая папка media (рисунок 74)

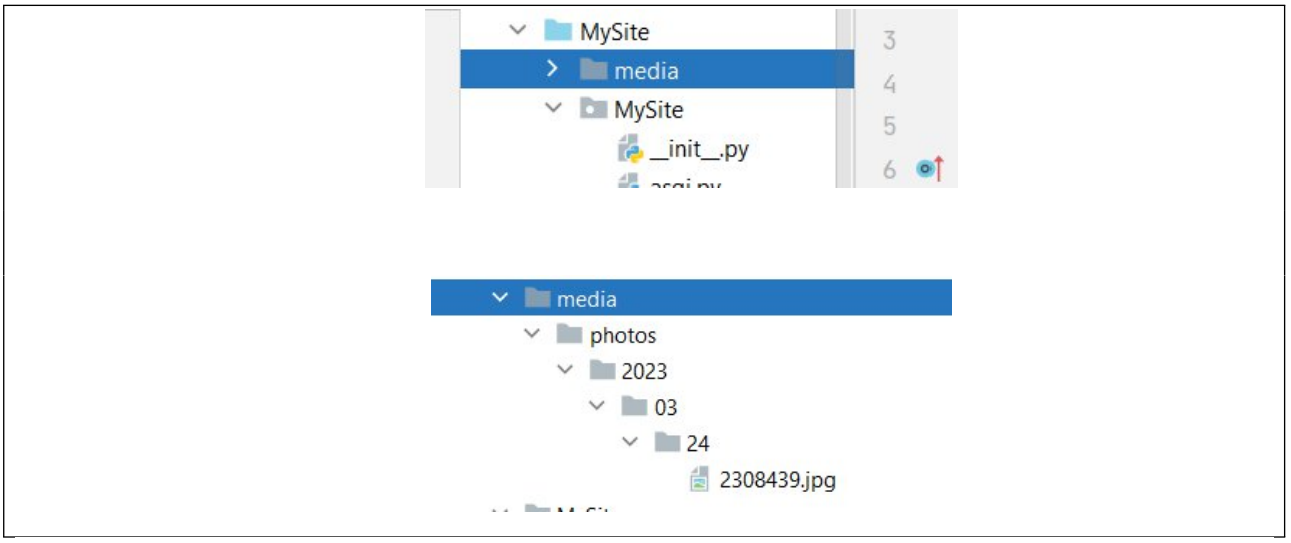


Рисунок 74

Если мы попробуем создавать новые записи, то появятся ошибки, связанные с заполнением обязательных полей.

Здесь необходимо сказать о валидации модели. Исправим это установление атрибута полю Фото как не обязательное для заполнения.

Для этого в файле модели в описании атрибутов поля Фото добавим атрибут blank=True (рисунок 75)

```
photo=models.ImageField(upload_to='photos/%Y/%m/%d', verbose_name='Фото', blank=True)
is_published=models.BooleanField(default=True, verbose_name='Опубликовано')
```

Рисунок 75

Но чтобы внести все эти изменения в модели, необходимо создать файл миграции с помощью уже известной команды (рисунок 76)

python manage.py makemigrations

```
(venv) C:\Users\Елена\PycharmProjects\pythonProject29\MySite>python manage.py makemigrations
Migrations for 'news':
  news\migrations\0002_alter_news_options_alter_news_content_and_more.py
    - Change Meta options on news
    - Alter field content on news
    - Alter field created_at on news
    - Alter field is_published on news
    - Alter field photo on news
    - Alter field title on news
    - Alter field updated_at on news

(venv) C:\Users\Елена\PycharmProjects\pythonProject29\MySite>
```

Рисунок 76

Как видим из рисунка, был выведен список всех изменений модели и в дереве списка создан новый файл миграции (рисунок 77)

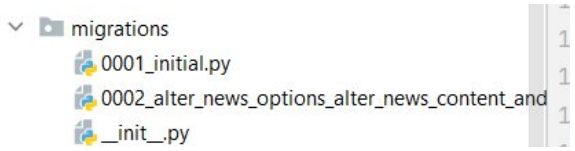


Рисунок 78

Далее выполняем саму миграцию с помощью тоже уже известной команды (рисунок 78)

python manage.py migrate

```
    - Alter field updated_at on news
(venv) C:\Users\Елена\PycharmProjects\pythonProject29\MySite>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, news, sessions
Running migrations:
  Applying news.0002_alter_news_options_alter_news_content_and_more... OK

(venv) C:\Users\Елена\PycharmProjects\pythonProject29\MySite>
```

Рисунок 79

Теперь попробуем изменить запись без прикрепления фото.

Лабораторная работа № 5.

Связи моделей. Внешний вид шаблона

Рассмотрим связи моделей, которые описываются в Django полями отношений (смотрите документацию https://django.fun/ru/docs/django/4.1/ref/models/fields/) (рисунок 1).

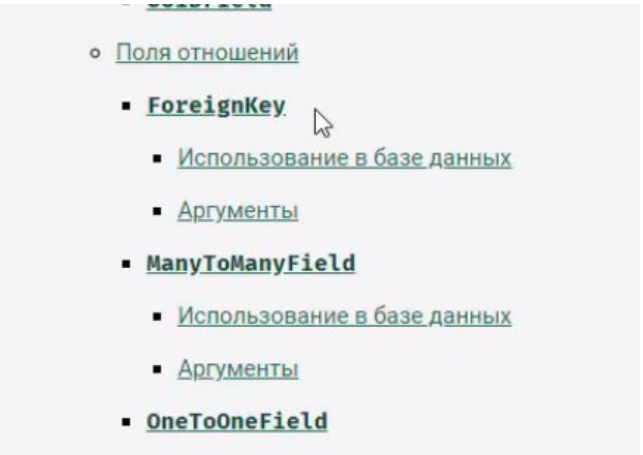
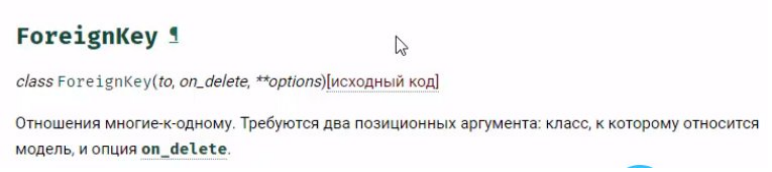


Рисунок 1

Как известно из теории баз данных, данные могут храниться в различных таблицах. Можно все данные хранить в одной таблице, но тогда нарушаются принципы нормализации данных. Например, рассмотрим наш пример с новостным сайтом. Как правило, все новости можно разбить на категории. И можно было бы создать в таблице news это дополнительное поле Категории новости и в нем писать название новости, которая относится к этой категории. Но опят, мы здесь бы нарушали принцип нормализации данных. Поэтому нужно создать еще одну табличку, в которой будет храниться категория. А в таблице news, будет храниться номер категории, к которой относиться данная новость. Таким образом, мы избегаем следующей проблемы, что если нам для чего-то нужно будет в какой-то момент переименовать категорию, то это можно сделать только один раз в самой таблице Категорий, а в самой таблице news, как был когда-то назначено значение внешнего ключа, так оно и останется не тронутым, не смотря на изменения.

Таким образом, мы понимаем, что связанные данные могут храниться в различных таблицах.

Для этого Django предлагает три специальных класса для организации связей.



ForeignKey ¶

class ForeignKey(*to*, *on_delete*, ***options*)[исходный код]

Отношения многие-к-одному. Требуются два позиционных аргумента: класс, к которому относится модель, и опция **on_delete**.

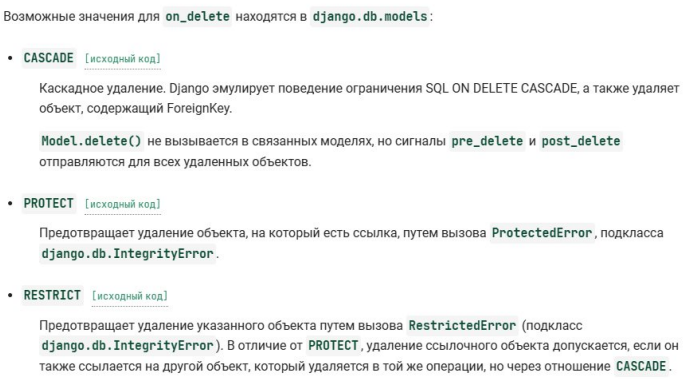
1 категория – много новостей

Нам необходимо объявить поле типа ForeignKey - отношения многие-к-одному.

Опция on_delete позволяет организовать ограничения «Что делать с зависимым объектом при удалении объекта, на который он ссылается.

Например, есть категория Культура и с ней связано много новостей. Если вдруг удалили категорию Культуру, то вторым аргументов on_delete необходимо определить, что делать в этом случае.

В качестве значений on_delete можно указать различные виды поведений (рисунок 2)



Возможные значения для `on_delete` находятся в `django.db.models`:

- `CASCADE` [исходный код]

  Каскадное удаление. Django эмулирует поведение ограничения SQL ON DELETE CASCADE, а также удаляет объект, содержащий ForeignKey.

  `Model.delete()` не вызывается в связанных моделях, но сигналы `pre_delete` и `post_delete` отправляются для всех удаленных объектов.

- `PROTECT` [исходный код]

  Предотвращает удаление объекта, на который есть ссылка, путем вызова `ProtectedError`, подкласса `django.db.IntegrityError`.

- `RESTRICT` [исходный код]

  Предотвращает удаление указанного объекта путем вызова `RestrictedError` (подкласс `django.db.IntegrityError`). В отличие от `PROTECT`, удаление ссылочного объекта допускается, если он также ссылается на другой объект, который удаляется в той же операции, но через отношение `CASCADE`.

Рисунок 2

Рассмотрим тип ForeignKey.

Перейдем к нашей модели и создадим новую модель Категории. Модель Категория первичная, Новости – вторичная

db_index=True – этот параметр необходим для индексирования этого поля

Индексация позволяет организовать более быстрый поиск. Как вы знаете, СУБД поле primary key является индексированным по умолчанию (рисунок 3).

```
class Category(models.Model):
    title=models.CharField(max_length=150, db_index=True, verbose_name='Наименование категории')
```

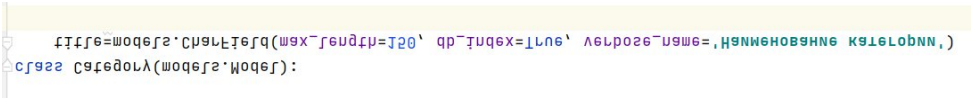


Рисунок 3

И также создадим подкласс Meta (рисунок 4):

```
class Category(models.Model):
    title=models.CharField(max_length=150, db_index=True, verbose_name='Наименование категории')

    class Meta:
        verbose_name='Категория'
        verbose_name_plural = 'Категории'
        ordering=['title']# сортировка по наименованию названия
```

Рисунок 4

Можно сделать миграцию, конечно. Но для начала, необходимо связать эти модели и для этого воспользуемся классом ForeignKey. Он записывается во вторичной модели как поле

Ссылку на модель необходимо указано, до того, как объявлена первичная модель. Потому первичную модель необходимо написать раньше. Т.е. перенесем ее над классом news (рисунок 5)



Рисунок 5

И теперь можем изменить модель news (рисунок 6, 7, 8).



Рисунок 6

Но если класс был объявлен позже, то этот параметр класса ForeignKey, должен быть указан в виде строки (рисунок 7)

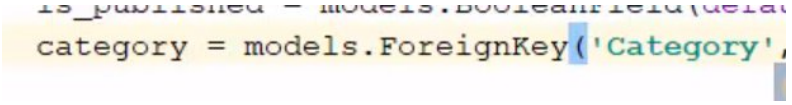


Рисунок 7

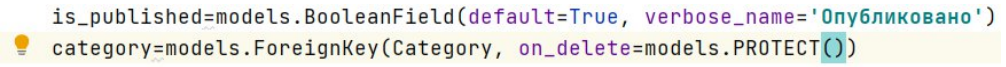


Рисунок 8

Не забудьте прописать второй аргумент класса ForeignKey on_delete.

При вводе функции PROTECT в данном случае необходимо удалить скобки (). В соответствии с документацией не мы вызываем эту функцию с параметрами, а Django сам вызовет ее в нужный момент. Мы ее просто прописываем.

Итак, models.PROTECT обеспечивает защиту удаления связанных данных,

Далее, известно, что все поля по умолчанию прописываются как обязательные для заполнения. При этом в нашей модели уже есть записи, и мы ее связываем с моделью Категории с помощью ForeignKey. В связи с этим СУБД будет строить какие-то ограничения и поэтому добавить новое обязательное поле в модель, которая имеет уже записи уже так просто не получится. СУБД не позволит нам это сделать. Поэтому мы воспользуемся опцией NULL (рисунок 9)

```
is_published=models.BooleanField(default=True, verbose_name='Опубликовано')
category=models.ForeignKey(Category, on_delete=models.PROTECT, null=True)
```

Рисунок 9

Этот параметр позволит нам не заполнять это поле.

Допишем еще параметр default=1, который обозначает, что по умолчанию поле заполнится 1.

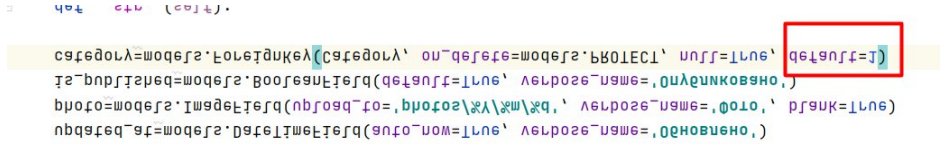


Рисунок 10

Теперь перейдем к созданию и выполнению миграции (рисунок 11).

```
(venv) C:\Users\Елена\PycharmProjects\pythonProject29\MySite>python manage.py makemigrations
Migrations for 'news':
  news\migrations\0003_category_news_category.py
    - Create model Category
    - Add field category to news

(venv) C:\Users\Елена\PycharmProjects\pythonProject29\MySite>
```

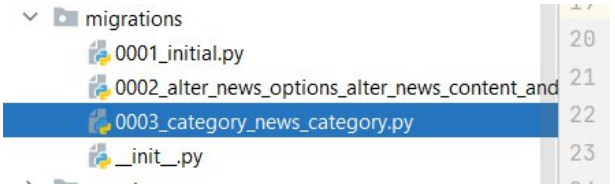


Рисунок 11

Теперь выполним миграцию (рисунок 12)



Рисунок 12

При этом у нас насыпались ошибки (рисунок 13)

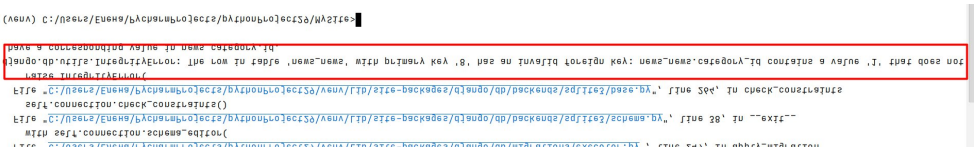


Рисунок 13

Это произошло потому, что у нас нет в таблице Категории такого идентификатора с 1.

Это нарушает целостность данных. И как результат возникновение этой ошибки.

Удаляем default=1 (рисунок 14)

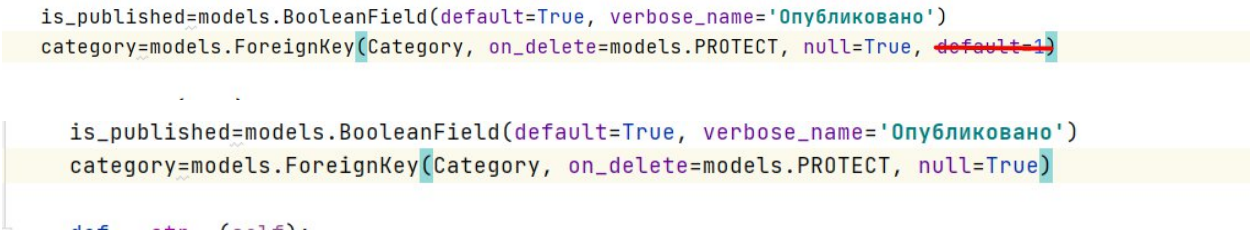


Рисунок 14

Снова создаем миграцию с номером 0004 (рисунок 15).

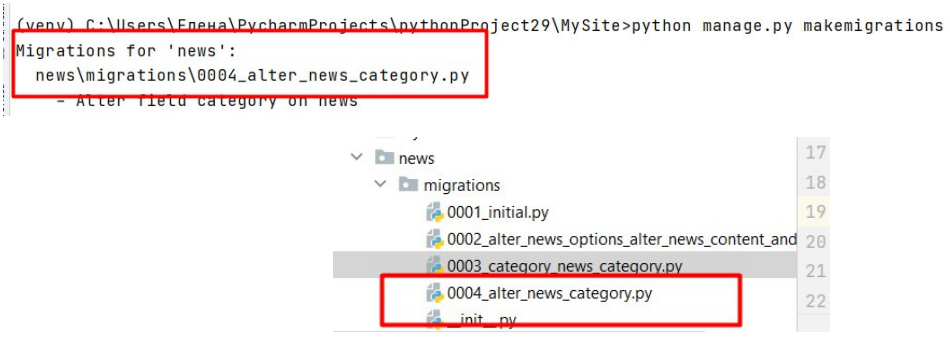


Рисунок 15

Снова делаем миграцию и у нас снова ошибка (рисунок 16)

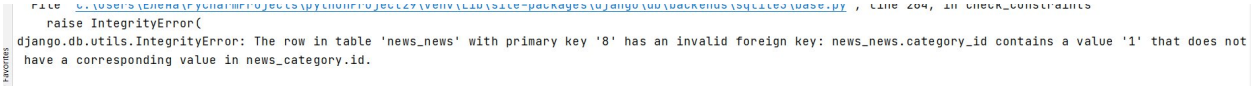


Рисунок 16

Это получилось так, потому что у нас есть миграция 003, которую Django так же послушно ее выполняет, которая содержит default=1

Посмотрим файл этой третьей миграции (рисунок 17)

```
),
migrations.AddField(
    model_name='news',
    name='category',
    field=models.ForeignKey(default=1, null=True, on_delete=django.db.models.deletion.PROTECT, to='news.categor
),
```

Рисунок 17

Поэтому удаляем 003 и 004 миграции и повторяем еще раз этапы миграции (рисунок 18).

```
(venv) C:\Users\Елена\PycharmProjects\pythonProject29\MySite>python manage.py makemigrations
Migrations for 'news':
  news\migrations\0003_category_news_category.py
    - Create model Category
    - Add field category to news
```

Рисунок 18

И как видим, у нас получилось выполнить миграции. Перезапустим БД (рисунок 19).

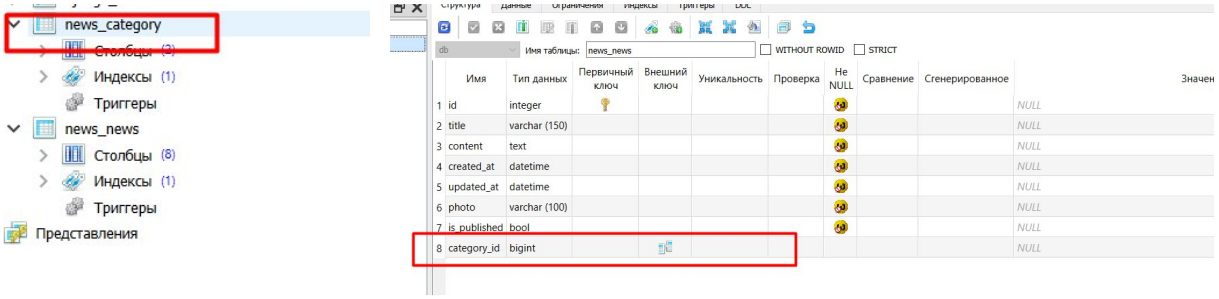


Рисунок 19

Как мы видим, в таблице появилось новое поле Категория (рисунок 20).

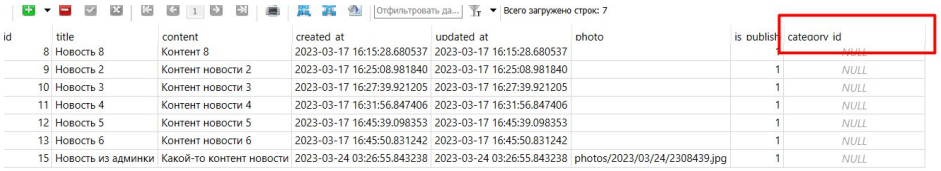


Рисунок 20

В ограничениях мы видим уже прописанные ограничения по полю Категории id первичный ключ (рисунок 21).

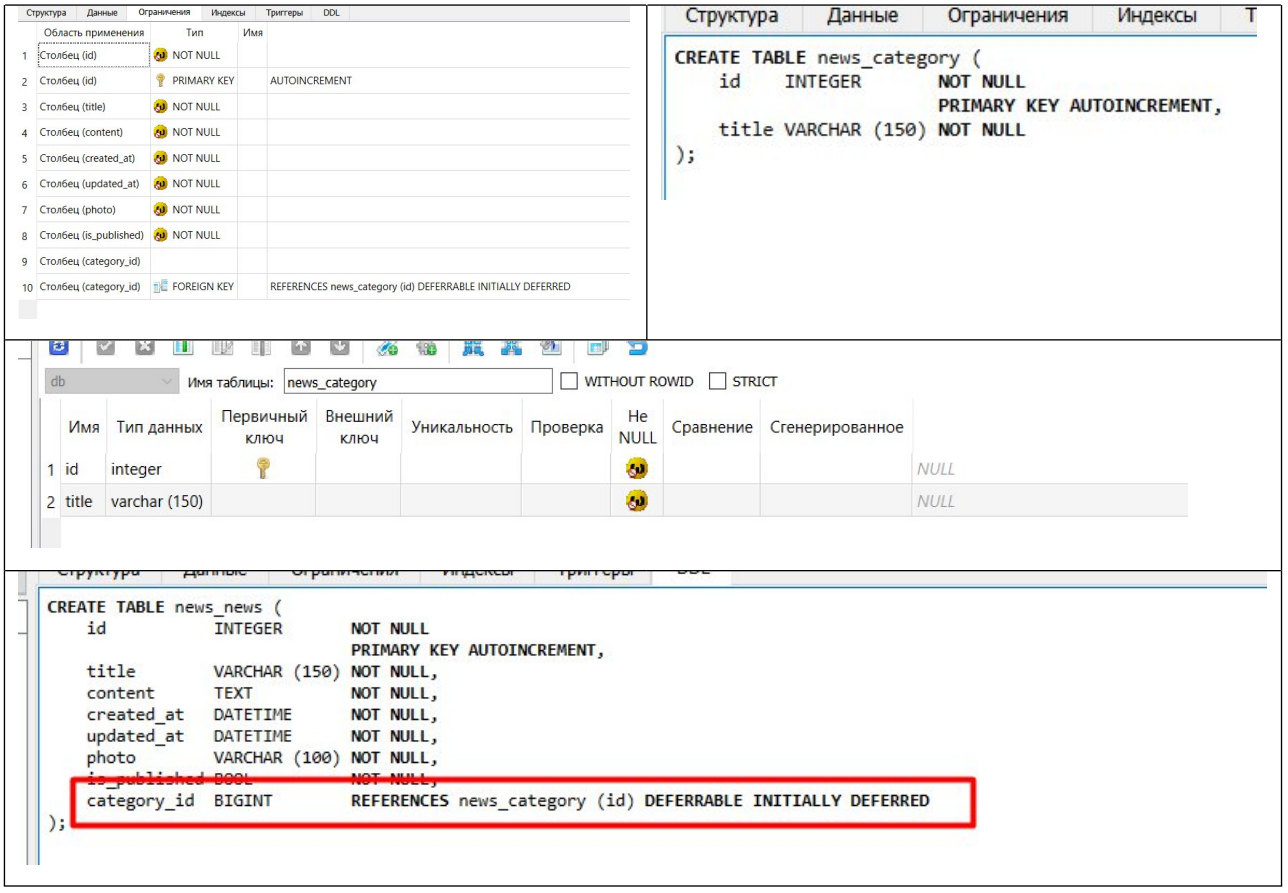
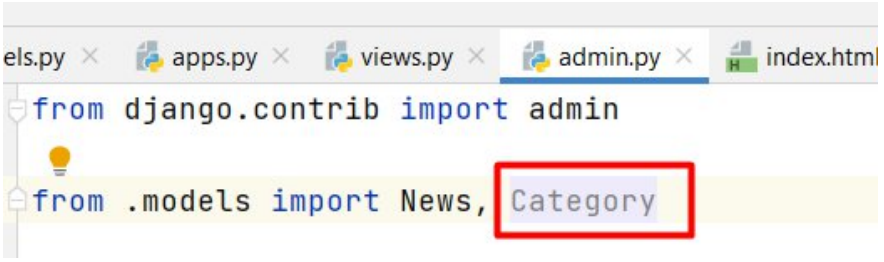


Рисунок 21

Разберемся, что необходимо еще сделать, чтобы эта связь заработала.

По сути дела, уже все готово для работы, только можно настроить внешний вид админки.

Первое, это надо зарегистрировать модель в файле admin.py (рисунок 22)



```
admin.site.register(News, NewsAdmin)
admin.site.register(Category)
```

Рисунок 23

И настроим (рисунок 24)

```
class CategoryAdmin(admin.ModelAdmin):
    list_display = ('id', 'title')
    list_display_links =('id','title')
    search_fields =('title',)# здесь аргументом является кортеж, поэтому в конце ставим запятую,
    # указывя, что остальных значений картежа нет


admin.site.register(News, NewsAdmin)
admin.site.register(Category, CategoryAdmin)
```

Рисунок 24

Обновим админку в браузере (рисунок 25)

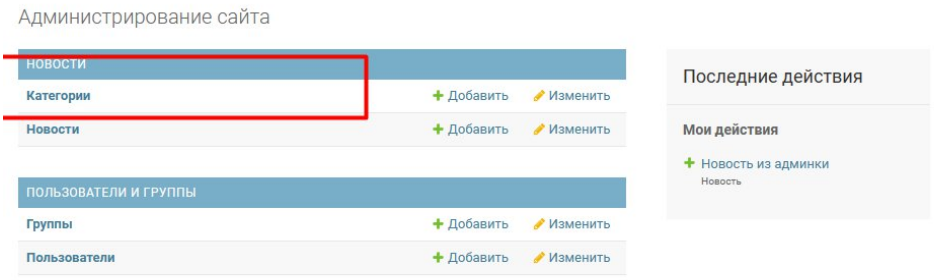
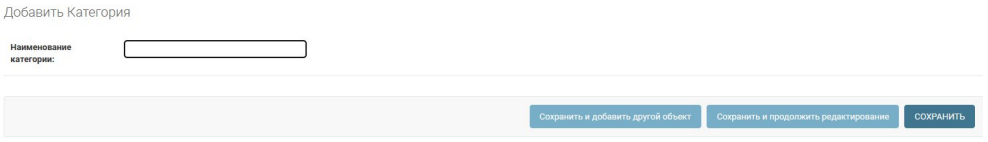


Рисунок 25

Добавим категории (категория 26).



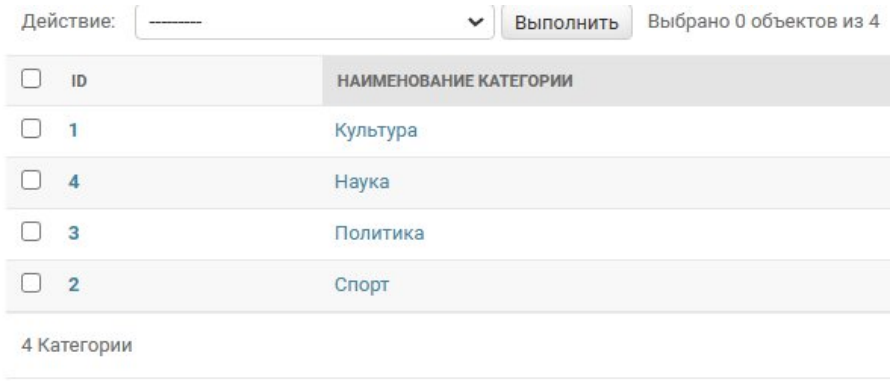


Рисунок 26

Перейдем в Новости (рисунок 27)

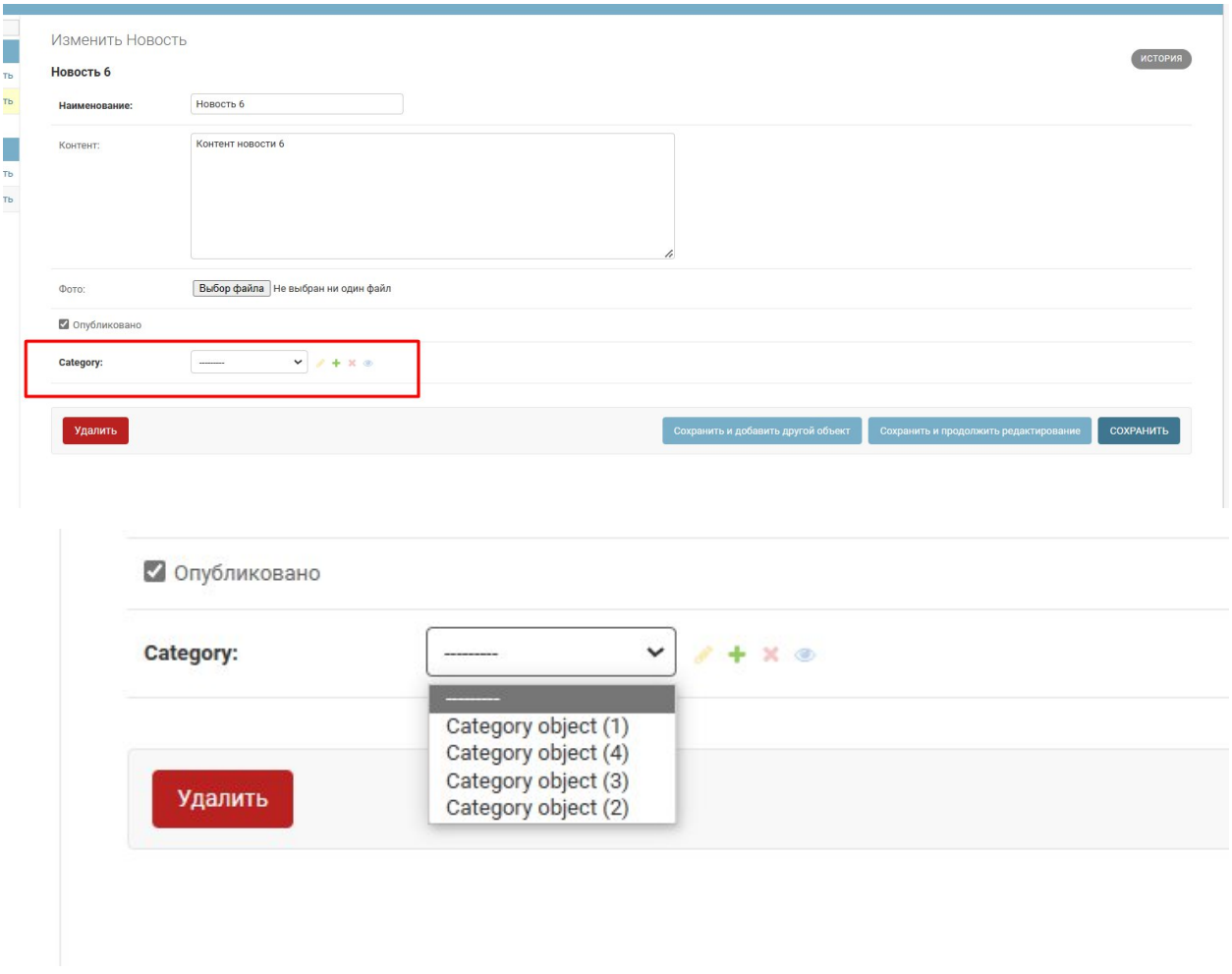


Рисунок 27

Давайте уберем непонятный object().

Для этого добавим строгое представление объекта (рисунок 28).

```python
from django.db import models

class Category(models.Model):
    title=models.CharField(max_length=150, db_index=True, verbose_name='Ha

    def __str__(self):
        return self.title

    class Meta:
        verbose_name='Категория'
        verbose_name_plural = 'Категории'
        ordering=['title']# сортировка по наименованию названия
```

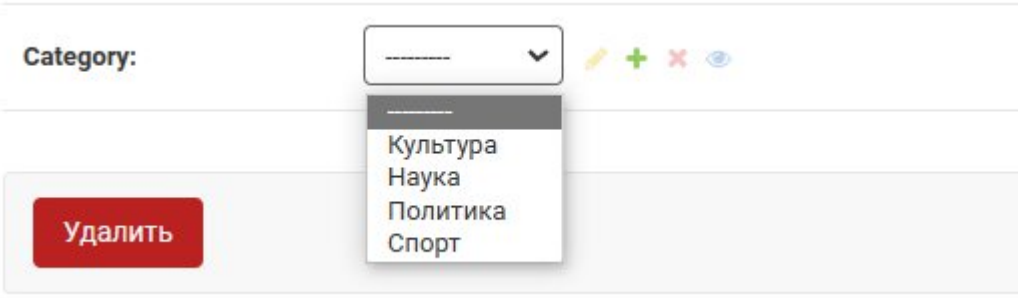


Рисунок 28

Попытаемся Сохранить, но будет ошибка (рисунок 29)

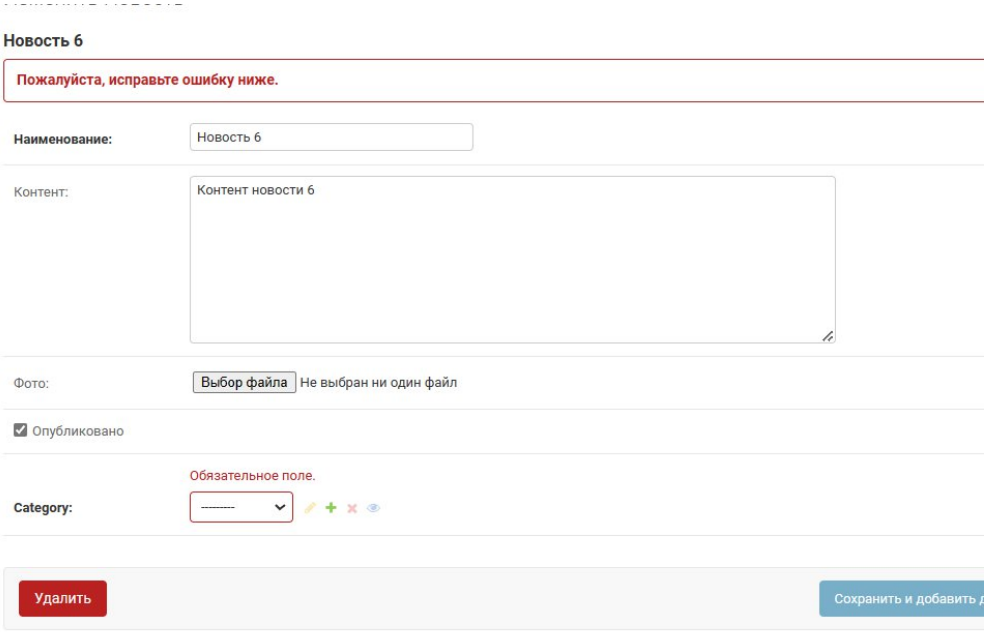


Рисунок 29

Потому что, оно обязательное для валидатора.

И тогда нужно добавить blank=True

Но это мы делать не будем, а просто выберем категории и заполним поле.

Пройдем по всем новостям и присвоим им категории (рисунок 30).

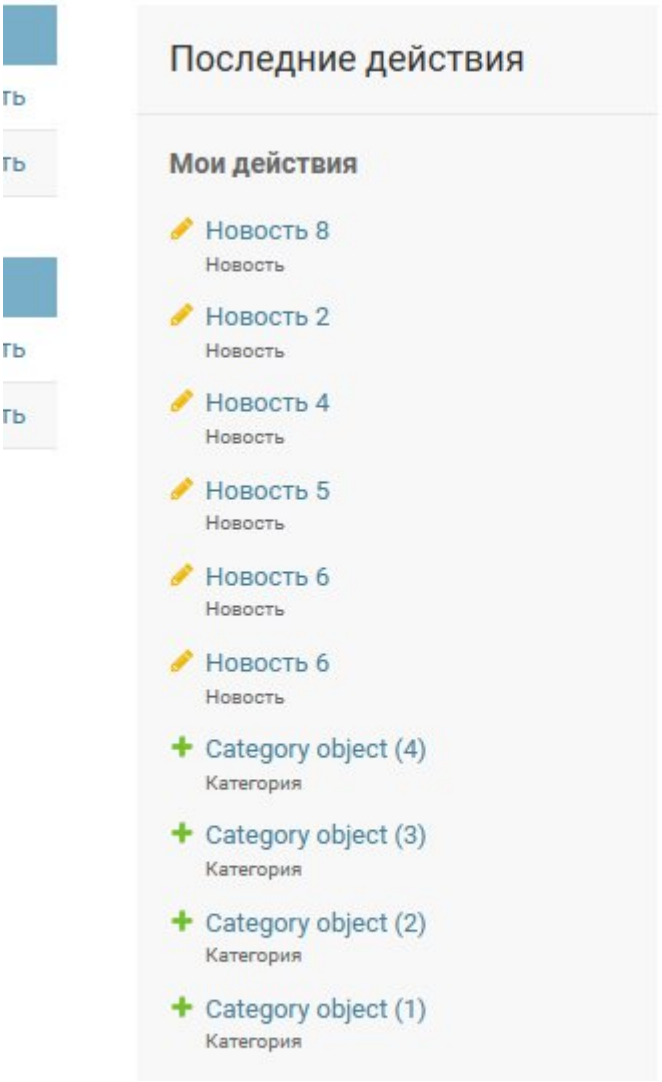


Рисунок 30

Настроим админку в модели Новости выводилась категория (рисунок 31)

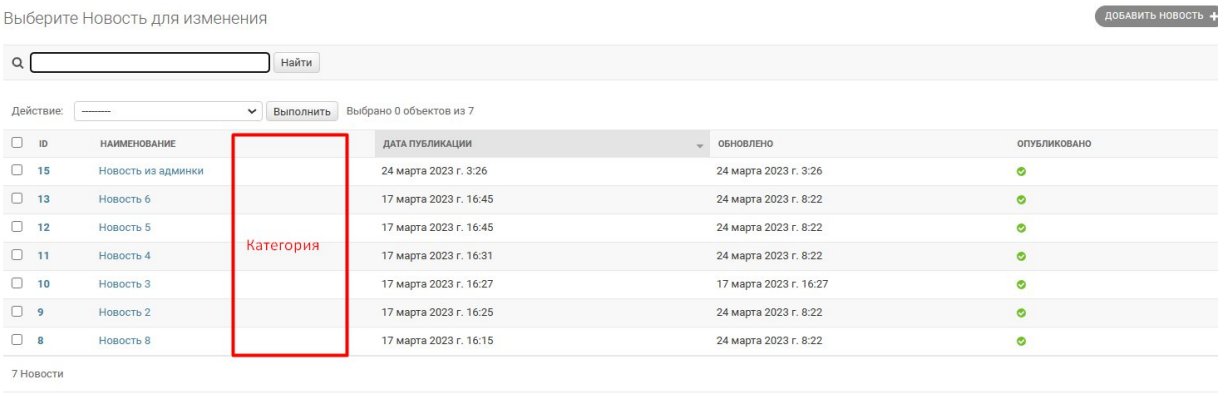


Рисунок 32

```
class NewsAdmin(admin.ModelAdmin):
    list_display = ('id', 'title', 'category', 'created_at', 'updated_at','is_published')
    list_display_links =('id','title')
    search_fields =('title','content')

class CategoryAdmin(admin.ModelAdmin):
```

Рисунок 32

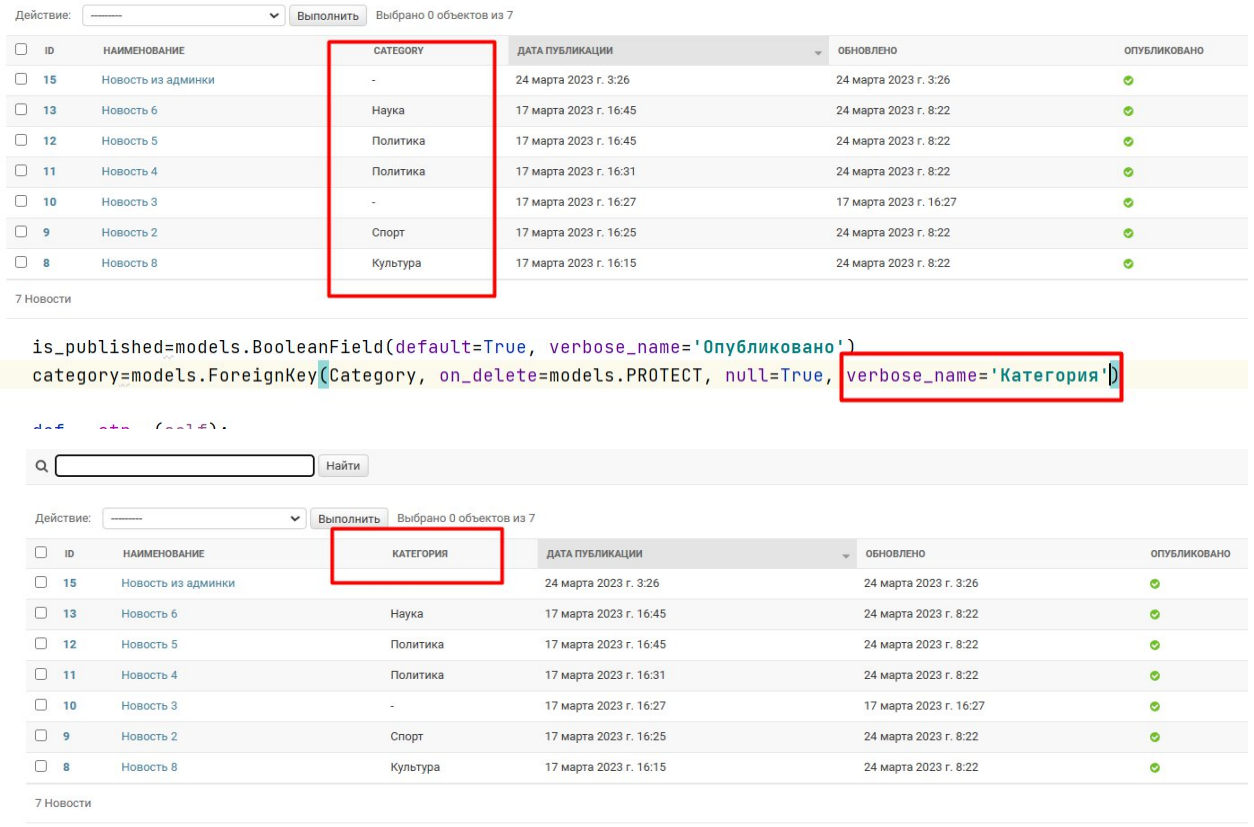


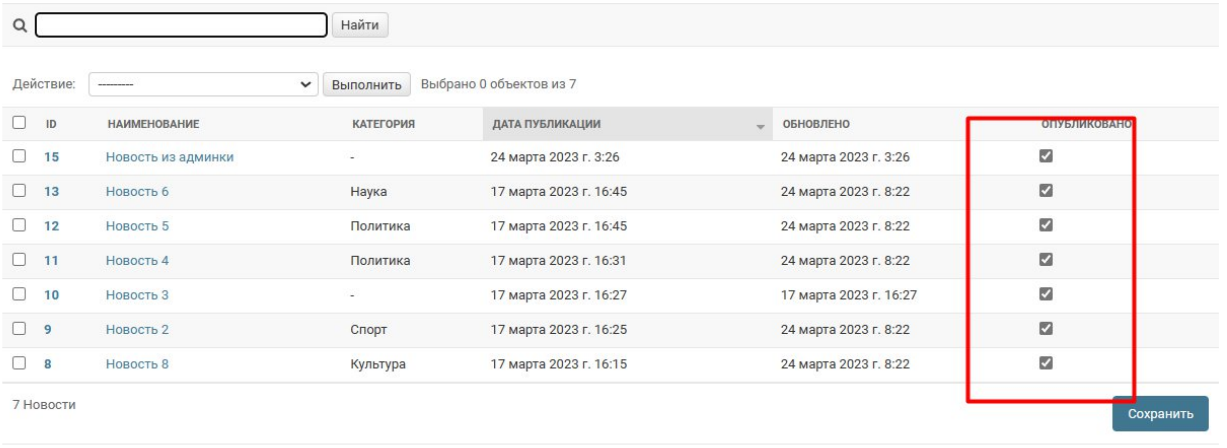
Рисунок 33

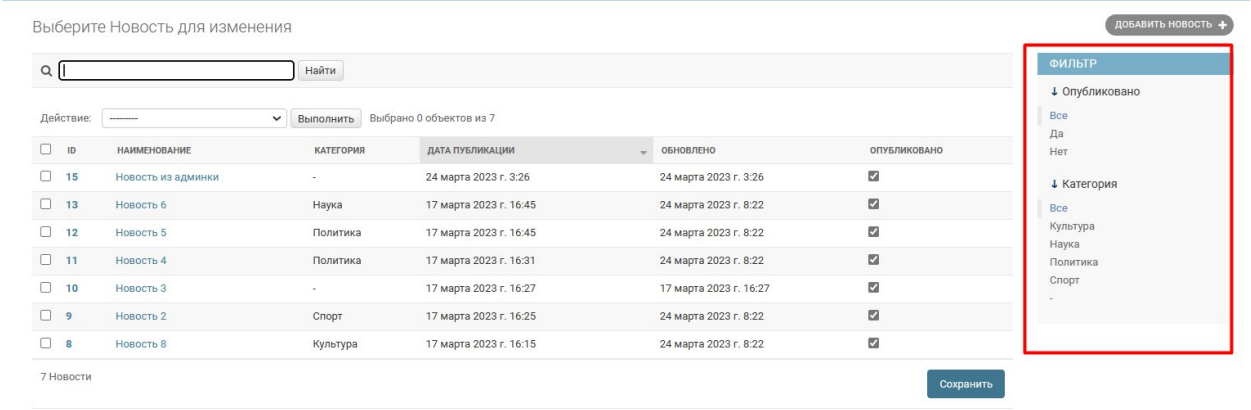
Давайте еще улучшим нашу админку. Сделаем редактируемые поля и добавим фильтры. Например, в поле is_published сделать возможность установки из поля админки (рисунок 34).

```
class NewsAdmin(admin.ModelAdmin):
    list_display = ('id', 'title', 'category', 'created_at', 'updated_at','
    list_display_links =('id','title')
    search_fields =('title','content')
    list_editable = ('is_published',)
    list_filter = ('is_published', 'category')
```

Рисунок 34

Проверим результаты настройки (рисунок 35)





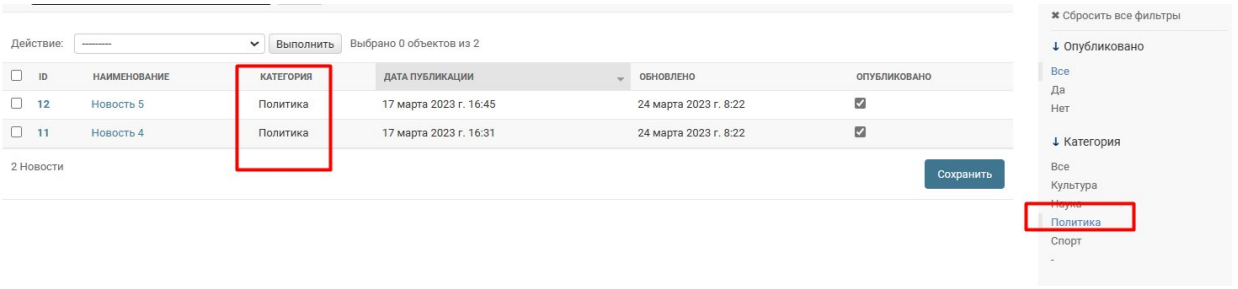


Рисунок 35

Лабораторная работа № 6.

Внешний вид шаблона. Директивы. Теги. Фильтры.

Вернемся к нашему списку новостей, в котором сформирован HTML код новостной ленты (рисунок 1).

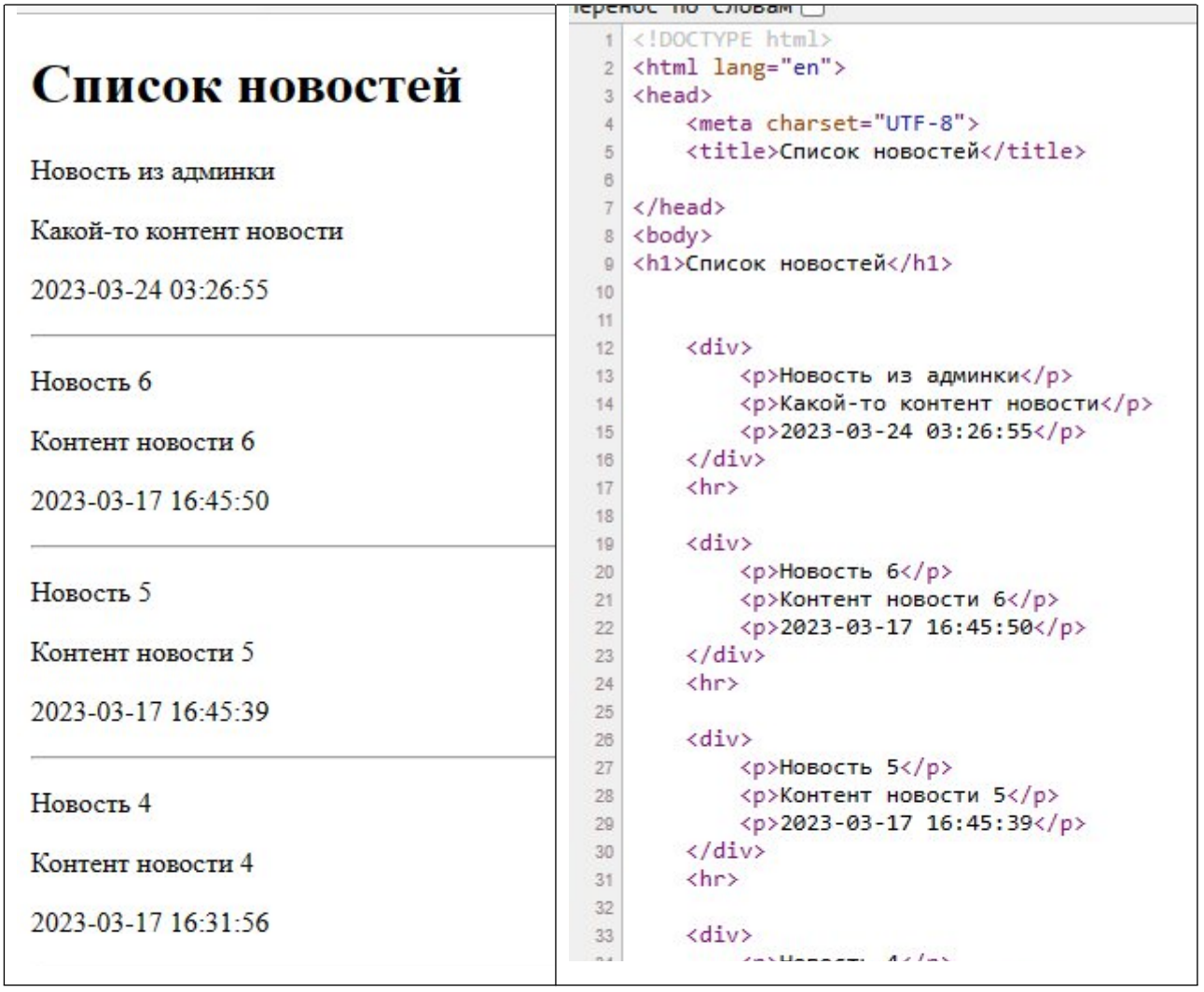


Рисунок 1

Но не смотря, на форматный вывод информации на нашей новостной странице, страница выглядит убогой. Для улучшения дизайна вывода информации воспользуемся

Bootstrap — это открытый и бесплатный HTML, CSS и JS фреймворк, который используется веб-разработчиками для быстрой вёрстки адаптивных дизайнов сайтов и веб-приложений.

Перейдем на сайт getbootstrap.com (рисунок 2)

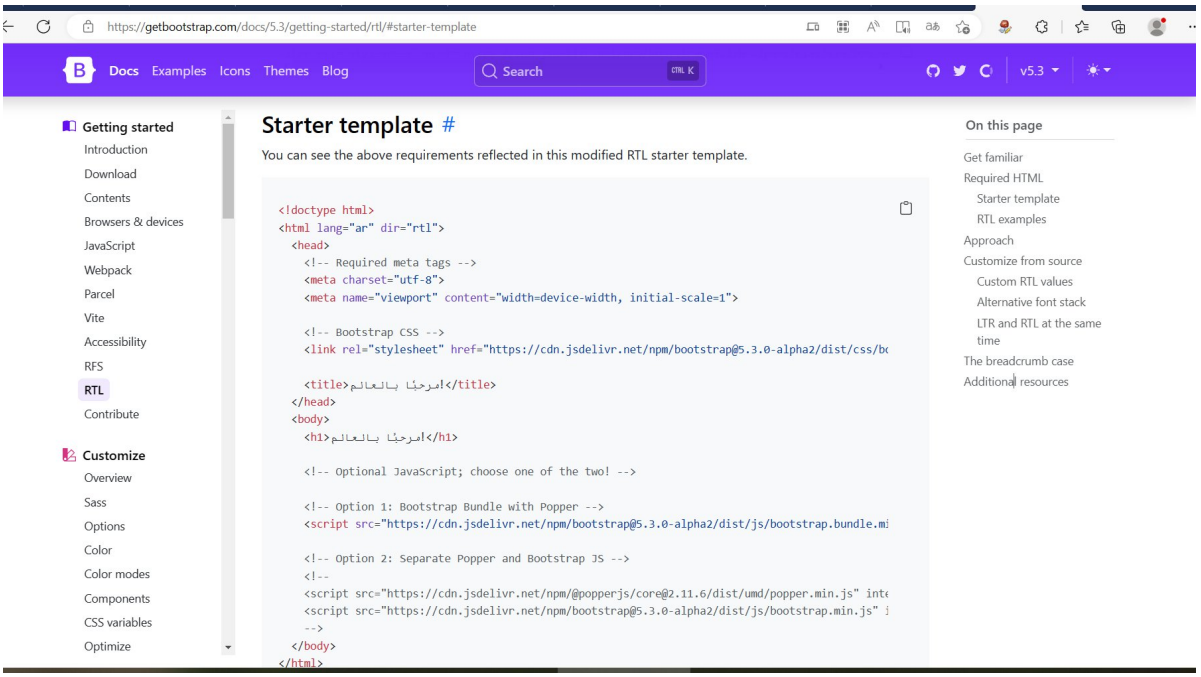


Рисунок 2

Для начала познакомьтесь с фрейверком https://getbootstrap.com/docs/5.3/getting-started/introduction/

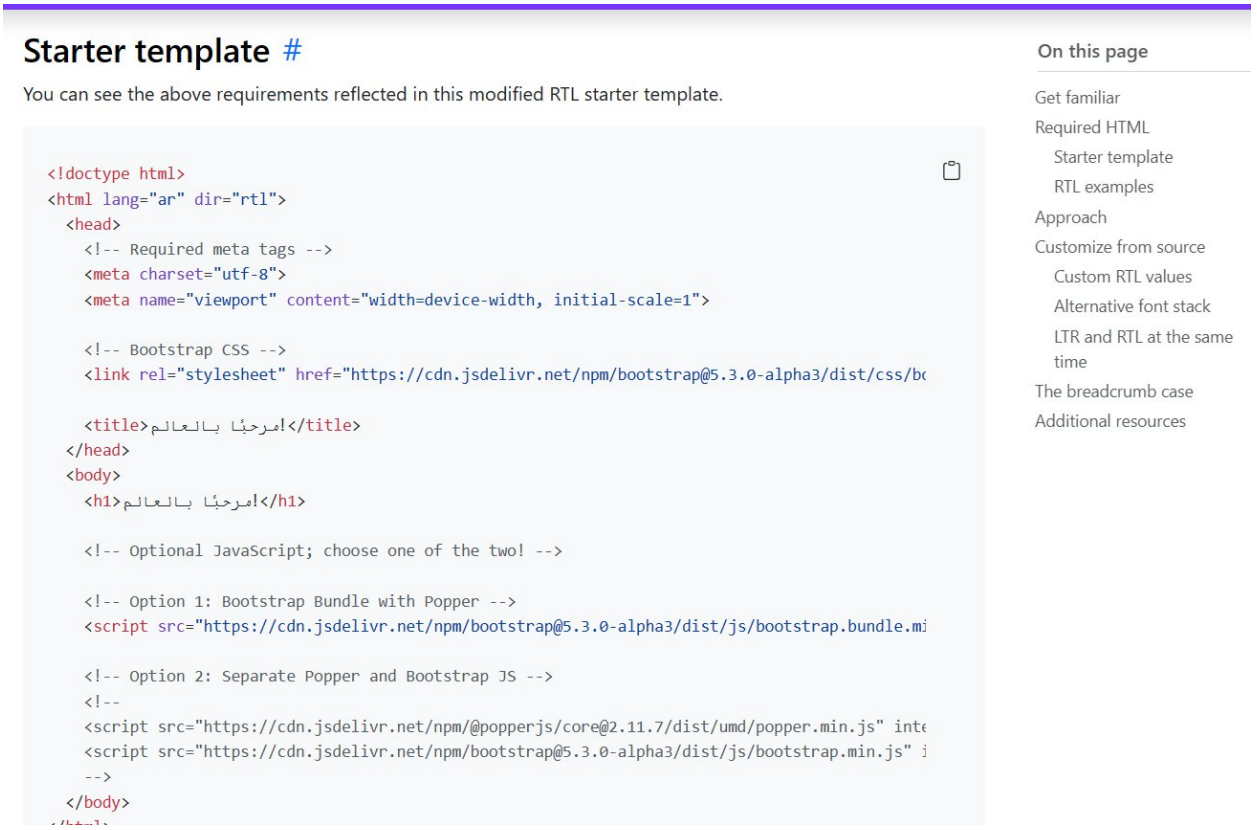
Для своей работы мы не будем ни чего загружать и скачивать. Воспользуемся просто шаблоном (рисунок 3), загружаемый через CDN (рисунок 3) по ссылке https://getbootstrap.com/docs/5.3/getting-started/rtl/.

CDN (Content Delivery Network) — это географически распределённая сетевая инфраструктура, обеспечивающая быструю доставку контента пользователям веб-сервисов и сайтов. Входящие в состав CDN серверы географически располагаются таким образом, чтобы сделать время ответа для пользователей сайта/сервиса минимальным.



Рисунок 3

Выберем Starter template (рисунок 4)



Starter template

You can see the above requirements reflected in this modified RTL starter template.

```
<!doctype html>
<html lang="ar" dir="rtl">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/css/bo

    <title>مرحبًا بالعالم!</title>
  </head>
  <body>
    <h1>مرحبًا بالعالم!</h1>

    <!-- Optional JavaScript; choose one of the two! -->

    <!-- Option 1: Bootstrap Bundle with Popper -->
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/js/bootstrap.bundle.mi

    <!-- Option 2: Separate Popper and Bootstrap JS -->
    <!--
    <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.7/dist/umd/popper.min.js" inte
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/js/bootstrap.min.js" i
    -->
  </body>
</html>
```

On this page

- Get familiar
- Required HTML
  - Starter template
  - RTL examples
- Approach
- Customize from source
  - Custom RTL values
  - Alternative font stack
  - LTR and RTL at the same time
- The breadcrumb case
- Additional resources

Рисунок 4

Скопируем это шаблон и вставим в наше приложение в файл index.html

Обратите внимание, что в этом шаблоне установлен арабский язык.
Поменяйте на en (рисунок 5) и удалите `dir="rtl"`



index.html

```
<!doctype html>
<html lang="en">
  <head>

    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3

    <title>{{title}}</title>
  </head>
```

Рисунок 5

Проведем редактирование шаблона. Удалим все комментарии.

Удалим арабские надписи (рисунок 6)



Рисунок 6

И теперь подстроим страницу под наши замыслы. Начнем с title.

Введем в нему переменную, которая передается из контроллера (рисунок 7).



Рисунок 7

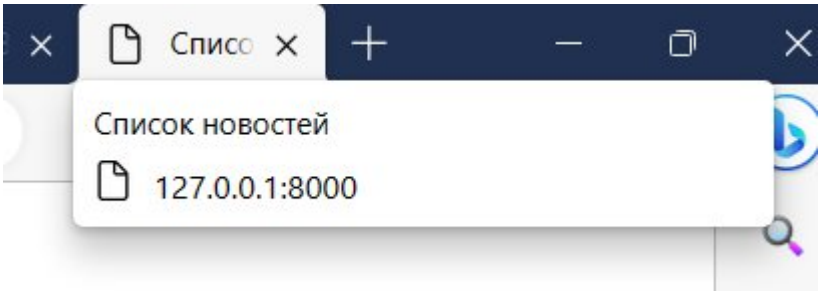


Рисунок 7

Теперь внедрим шаблон Navbar из списка компонентов (рисунок 8).

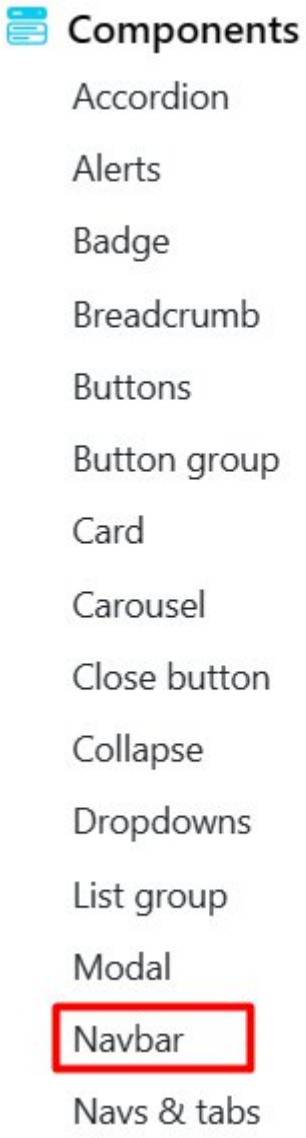


Рисунок 8

Скопируем шаблон вставим вместо строки <h>Новости</h1> (рисунок 9, 10).

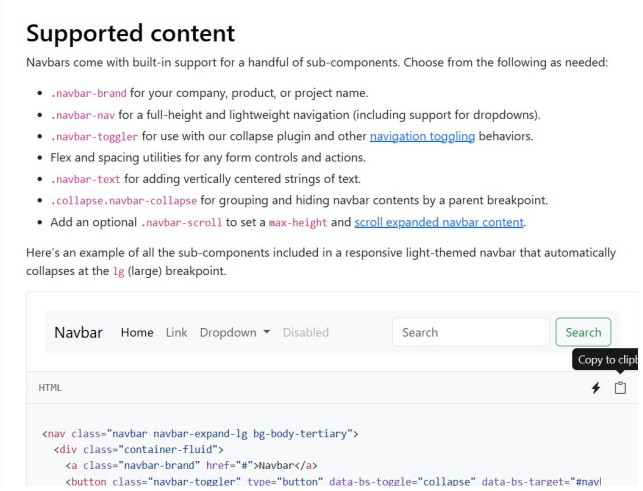


Рисунок 10

Проведем небольшое редактирование (рисунок 11)

```
    <title>{{title}}</title>
</head>
<body>
    <nav class="navbar navbar-expand-lg bg-body-tertiary">
<div class="container-fluid">
    <a class="navbar-brand" href="#">Navbar</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-tar
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarSupportedContent">
      <ul class="navbar-nav me-auto mb-2 mb-lg-0">
        <li class="nav-item">
          <a class="nav-link active" aria-current="page" href="#">Home</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="#">Link</a>
        </li>
        <li class="nav-item dropdown">
          <a class="nav-link dropdown-toggle" href="#" role="button" data-bs-toggle="d
            Dropdown
          </a>
          <ul class="dropdown-menu">
            <li><a class="dropdown-item" href="#">Action</a></li>
            <li><a class="dropdown-item" href="#">Another action</a></li>
            <li><hr class="dropdown-divider"></li>
```

Рисунок 11

Удалим форму

```
          <a class="nav-link disabled">Disabled</a>
        </li>
      </ul>
      <form class="d-flex" role="search">
        <input class="form-control me-2" type="search" placeholder="Search" aria-label="Search">
        <button class="btn btn-outline-success" type="submit">Search</button>
      </form>
    </div>
</div>
```

Рисунок 12

Оставим одну ссылку на главную страницу.

Ниже вы можете скопировать отредактированный шаблон

```
<<nav class="navbar navbar-expand-lg bg-body-tertiary">
  <div class="container-fluid">
    <a class="navbar-brand" href="#">Navbar</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
data-bs-target="#navbarSupportedContent" aria-
controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle
navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarSupportedContent">
      <ul class="navbar-nav me-auto ">
        <li class="nav-item">
          <a class="nav-link" href="#">Link</a>
        </li>

      </ul>

    </div>
  </div>
</nav>


    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-
alpha3/dist/js/bootstrap.bundle.min.js" integrity="sha384-
ENjdO4Dr2bkBIFxQpeoTz1HIcje39Wm4jDKdf19U8gI4ddQ3GYNS7NTKfAdVQSZe"
crossorigin="anonymous"></script>

  </body>
</html>
```

В строке Link заменим на Главная и заменим # на / (рисунок 13).

```
<div class="container-fluid">
  <a class="navbar-brand" href="/">Navbar</a>
  <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-targ
    <span class="navbar-toggler-icon"></span>
  </button>
  <div class="collapse navbar-collapse" id="navbarSupportedContent">
    <ul class="navbar-nav me-auto ">
      <li class="nav-item"><a class="nav-link" href="/">Главная</a></li>
      <li class="nav-item"><a class="nav-link" href="/">Добавить новость</a></li>
```

```
<ul class="navbar-nav me-auto ">
  <li class="nav-item">
    <a class="nav-link" href="/">Главная</a>
  </li>

</ul>
```

Рисунок 13

Создадим еще ссылку (рисунок 14)

```
<div class="collapse navbar-collapse" id="navbarSupportedContent">
  <ul class="navbar-nav me-auto ">
    <li class="nav-item"><a class="nav-link" href="/">Главная</a></li>
    <li class="nav-item"><a class="nav-link" href="/">Добавить новость</a></li>

  </ul>
```

Рисунок 14

Обновим страницу (рисунок 15)

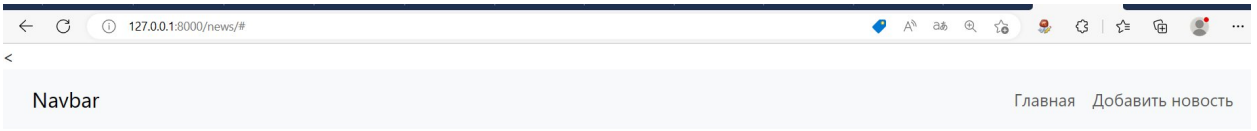


Рисунок 15

Создадим список новостей с помощью контейнера (рисунок 16)

```
    </div>
  </div>
</nav>
<div class='container'></div>
    <h1> {{title}}</h1>
    <div class="row">
      {% for item in news %}

      {%endfor%}

    </div>
```

Рисунок 16

И включим компонент карточки, который также возьмем на сайте getbootstrap.com (рисунок 17)

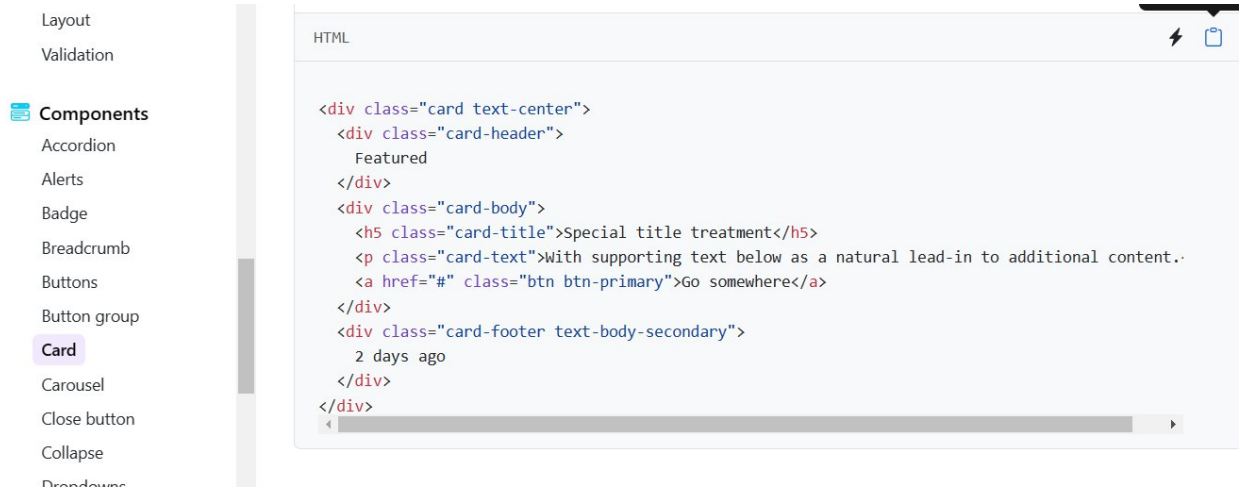


Рисунок 17

И вставим между тегами (рисунок 18).



Рисунок 18

Применим CSS свойство margin (рисунок 19)

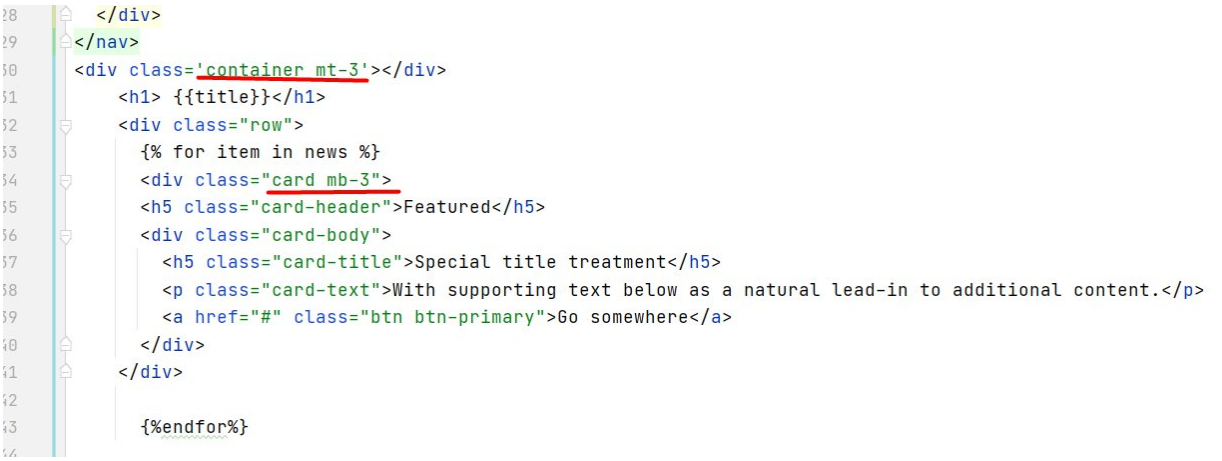


Рисунок 19

Пусть в карточке выводится Категория. Для этого заменим Featured на «Категорию» (рисунок 20).



Рисунок 20

Special title treatment заменим на вывод заголовка статьи

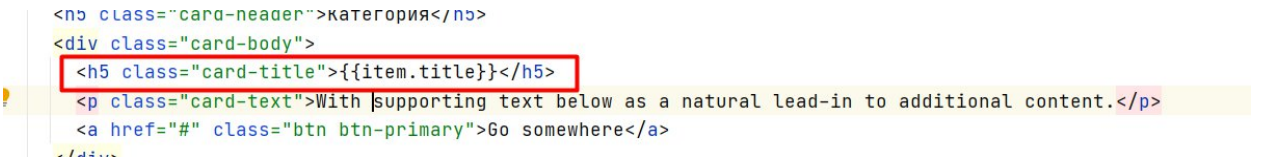


Рисунок 21

With supporting text below as a natural lead-in to additional content заменим на вывод контента новости (рисунок 22)

```
    <div class="card-body">
      <h5 class="card-title">{{item.title}}</h5>
      <p class="card-text">{{item.content}}</p>
      <a href="#" class="btn btn-primary">Go somewhere</a>
    </div>
  </div>

  {%endfor%}
```

Рисунок 22

Go somewhere заменим на Read more (рисунок 23)

```
    <div class="card-body">
      <h5 class="card-title">{{item.title}}</h5>
      <p class="card-text">{{item.content}}</p>
      <a href="#" class="btn btn-primary">Read more</a>
    </div>
  </div>

  {%endfor%}

</div>
```

Рисунок 23

В качестве футера заменим **2 days ago** на переменную создания новости (рисунок 24).

```
      <div class="card-header">Категория</div>
      <div class="card-body mb-3">
        <h5 class="card-title">{{item.title}}</h5>
        <p class="card-text">{{item.content}}</p>
        <a href="#" class="btn btn-primary">Read more...</a>
      </div>
      <div class="card-footer text-body-secondary">
        {{created_at|date: "Y-m-d H:i:s" }}
      </div>
    </div>

    {%endfor%}

  </div>
```

Рисунок 24

В результате имеем следующий листинг файла index.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-
alpha3/dist/css/bootstrap.rtl.min.css" integrity="sha384-
T5m5WERuXcjgzF8DAb7tRkByEZQGcpraRTinjpywg37AO96WoYN9+hrhDVoM6CaT"
```

```
crossorigin="anonymous">
    <title>{{title}}</title>
  </head>
  <body>
    <<nav class="navbar navbar-expand-lg bg-body-tertiary">
  <div class="container-fluid">
    <a class="navbar-brand" href="/">Navbar</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
data-bs-target="#navbarSupportedContent" aria-
controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle
navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarSupportedContent">
      <ul class="navbar-nav">
        <li class="nav-item"><a class="nav-link" href="/">Главная</a></li>
        <li class="nav-item"><a class="nav-link" href="/">Добавить
новость</a></li>

      </ul>

    </div>
  </div>
</nav>
<div class='container mt-3'>
    <h1> {{title}}</h1>
    <div class="row">
        <div class="col-md-12">
      {% for item in news %}
      <div class="card mb-3">
          <div class="card-header">Категория</div>
          <div class="card-body">
            <h5 class="card-title">{{item.title}}</h5>
            <p class="card-text">{{item.content}}</p>
            <a href="#" class="btn btn-primary">Read more...</a>
          </div>
          <div class="card-footer text-muted">
            {{item.created_at|date:"Y-m-d H:i:s"}}
          </div>
      </div>

      {%endfor%}
            </div>

    </div>
</div>

  </body>
</html>
```

После обновления видим следующую страницу в браузере (рисунок 25)

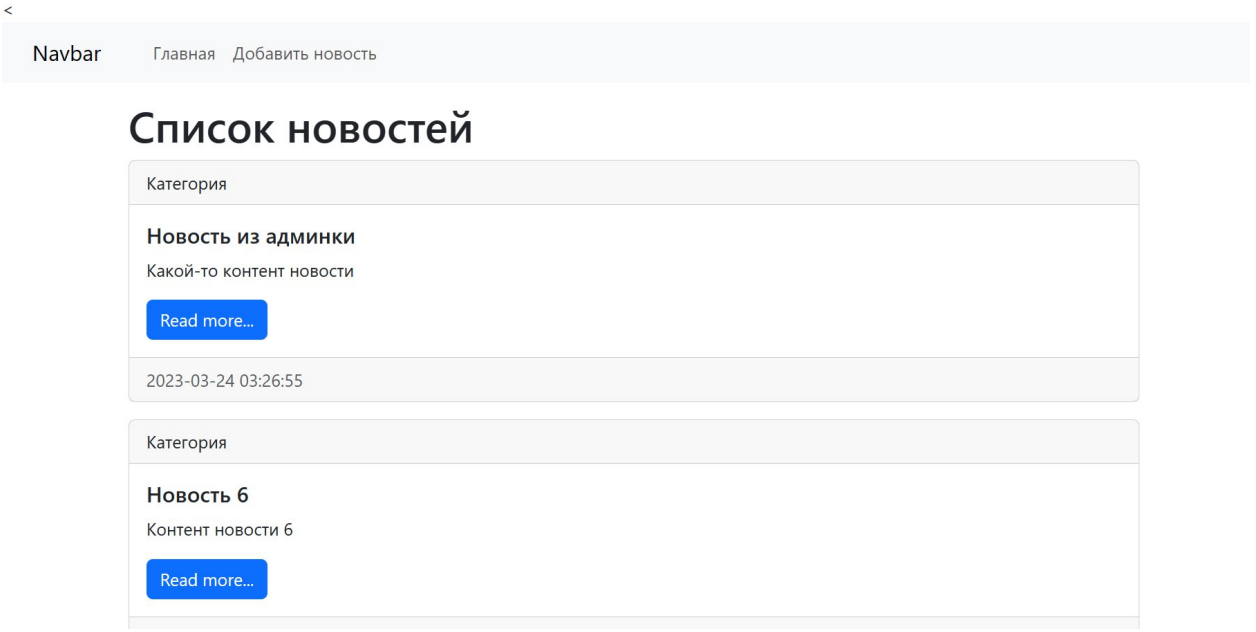


Рисунок 25

Теперь для каждой новости добавим немного текста, например, взяв «рыбу» текста с сайта https://ru.lipsum.com/. Скопируем какой-нибудь текст и через админку добавим в нашу новостную ленту.

Для этого перейдем в админку (рисунок 26) и выберем новость

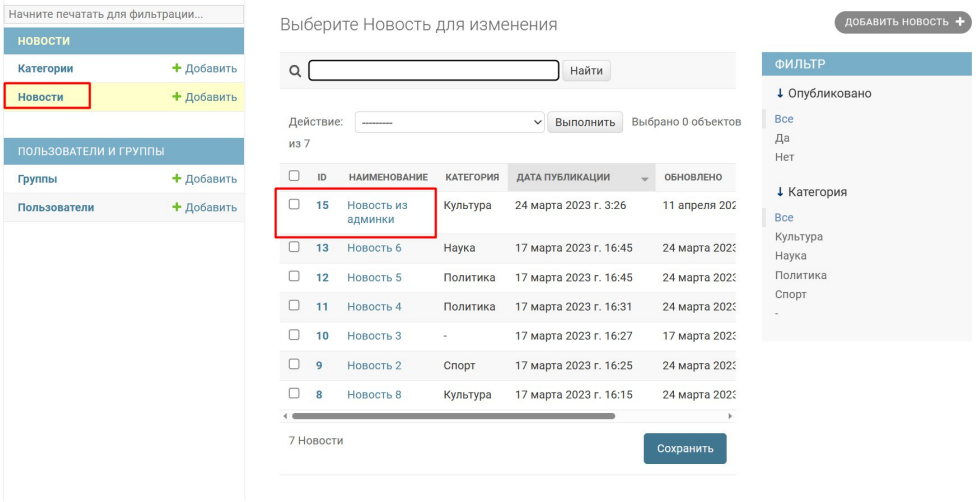


Рисунок 26

Затем добавим текст (рисунок 27)

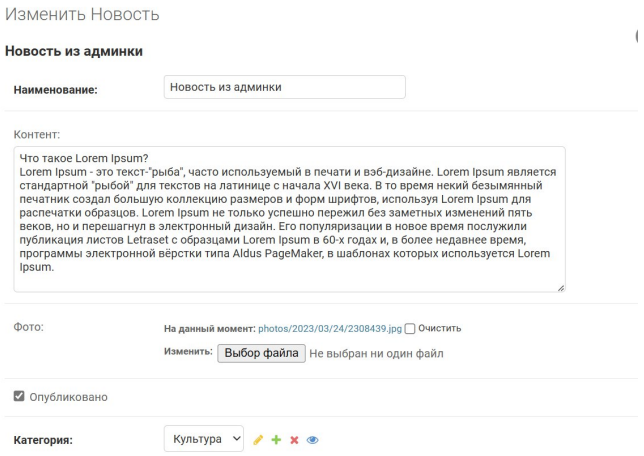


Рисунок 27

Заполните все новости текстом. После обновления рисунок 28

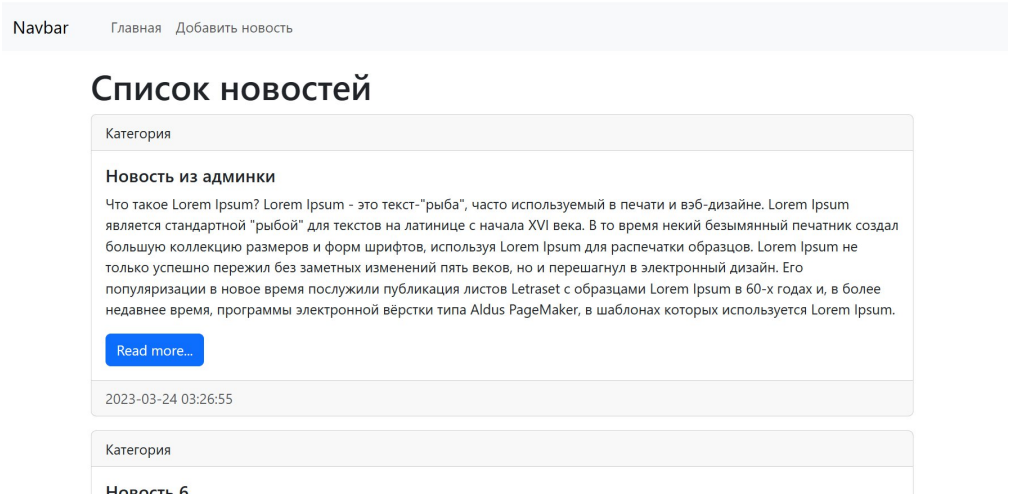


Рисунок 28

Следующим этапом выведем категории.

Чтобы вывести категорию, необходимо обратиться к этом у атрибуту в разделе Категории (рисунок 29) index.html файле



Рисунок 29

После обновления появится на карточке заголовок категории (рисунок 30)

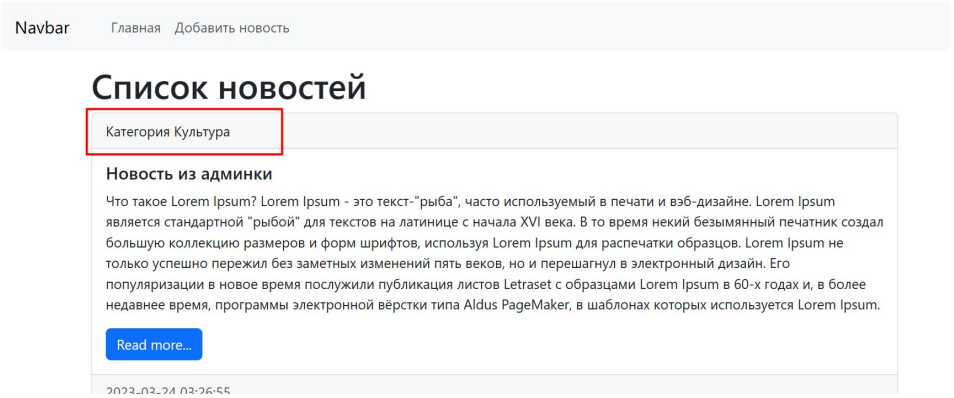


Рисунок 40

В шаблоне мы можем вызывать методы. Это означает, что шаблон имеет доступ не только к атрибутам класса, как мы обращались к атрибутам, как например, через {{item.category}}, но и к методам, которые описаны в модели.

Давайте определим в моделях какой-нибудь метод (рисунок 41).

Параметр self означает ссылку на текущий экземпляр класса.

```
class News(models.Model):
    title=models.CharField(max_length=150, verbose_name='Наименование')
    content=models.TextField(blank=True, verbose_name='Контент')
    created_at=models.DateTimeField(auto_now_add=True, verbose_name='Дата публикации')
    updated_at=models.DateTimeField(auto_now=True, verbose_name='Обновлено')
    photo=models.ImageField(upload_to='photos/%Y/%m/%d', verbose_name='Фото', blank=True)
    is_published=models.BooleanField(default=True, verbose_name='Опубликовано')
    category=models.ForeignKey(Category, on_delete=models.PROTECT, null=True, verbose_name='Категория')

    def my_func(self):
        return 'Последние новости'

    def __str__(self):
        return self.title

    class Meta:
        verbose_name='Новость'
        verbose_name_plural = 'Новости'
        ordering=['-created_at',]
```

Рисунок 41

И теперь в шаблоне index.html обратимся к данному методу (рисунок 42)

```
</nav>
<div class='container mt-3'>
    <h1> {{title}}</h1>
    <div class="row">
        <div class="col-md-12">
      {% for item in news %}
      <div class="card mb-3">
          <div class="card-header">
            {{item.my_func}}
            Категория {{item.category}}
          </div>
          <div class="card-body">
            <h5 class="card-title">{{item.title}}</h5>
            <p class="card-text">{{item.content}}</p>
            <a href="#" class="btn btn-primary">Read more...</a>
          </div>
          <div class="card-footer text-muted">
            {{item.created_at|date:"Y-m-d H:i:s"}}
          </div>
      </div>
```

Рисунок 42

Обновим страницу (рисунок 43)

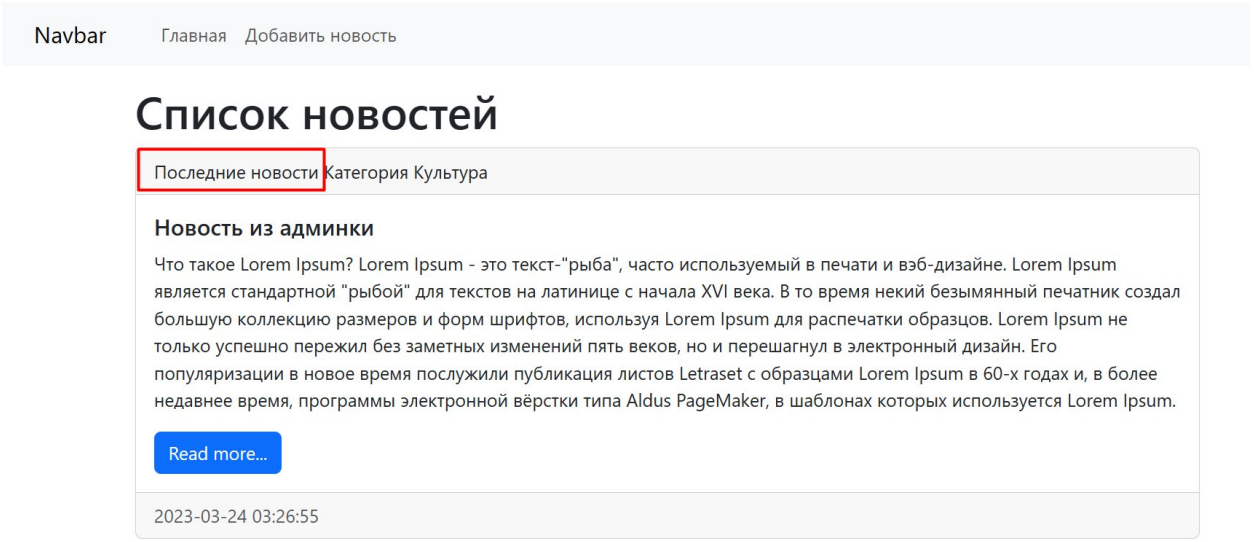


Рисунок 43

Применим тег cycle.

Производит один из своих аргументов каждый раз, когда встречается этот тег. Первый аргумент возникает при первой встрече, второй аргумент - при второй встрече и так далее. Как только все аргументы исчерпаны, тег циклически переходит к первому аргументу и производит его снова.

Этот тег особенно полезен в цикле (рисунок 44).

```
{% for o in some_list %}
    <tr class="{% cycle 'row1' 'row2' %}">
        ...
    </tr>
{% endfor %}
```

Рисунок 44

```
        Категория {{item.category}}
      </div>
      <div class="card-body">
        <h5 class="card-title {%cycle 'text-danger' 'text-success' %}">{{item.title}}</h5>
        <p class="card-text">{{item.content}}</p>
        <a href="#" class="btn btn-primary">Read more...</a>
      </div>
      <div class="card-footer text-muted">
        {{item.created_at|date:"Y-m-d H:i:s"}}
      </div>
    </div>
```

Рисунок 45

После обновления (рисунок 46), как вы видите, заголовки статей чередуются по цвету – красный/зеленый.

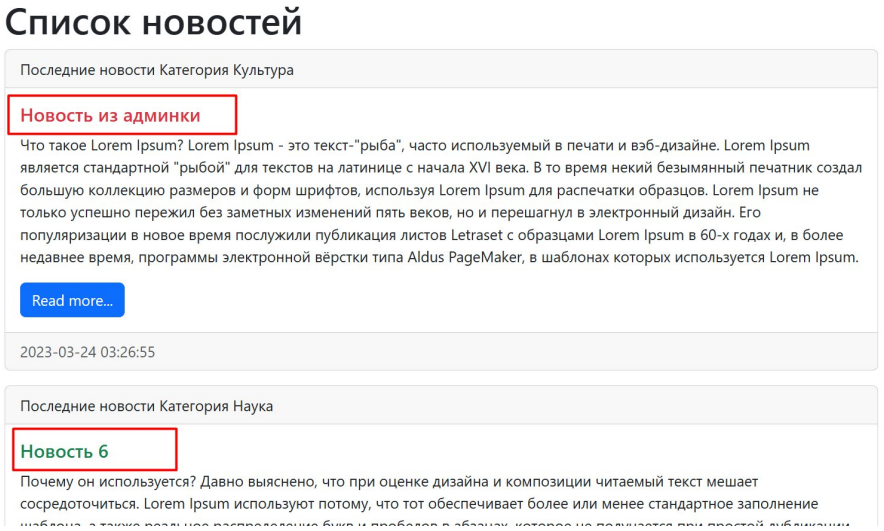


Рисунок 46

Воспользуемся еще одним тегом for, который мы уже использовали для вывод списка новостей. Цикл for устанавливает ряд переменных, доступных в цикле. Например, forloop.counter - текущая итерация цикла (начинается с индекса 1) (рисунок 46).

Встроенные теги и фильтры шаблонов - Django Документация на русском https://djangodoc.ru/3.2/ref/templates/builtins/

```
{% for item in news %}
<div class="card mb-3">
    <div class="card-header">
        {{forloop.counter}}
        {{item.my_func}}
        Категория {{item.category}}
    </div>
```

Рисунок 46

После обновления страницы (рисунок 47) увидим нумерацию наших новостей.



Рисунок 47

Рассмотрим следующий тег now - отображает текущую дату и/или время в формате, соответствующем заданной строке. Такая строка может содержать символы спецификаторов формата (рисунок 48).

```
v>
 class='container mt-3'>
<h1> {{title}}</h1>
{% now 'Y-m-d H:i:s' %}
<div class="row">
    <div class="col-md-12">
    {% for item in news %}
    <div class="card mb-3">
```

Рисунок 48

После обновления появится текущее время (рисунок 49)



Рисунок 49

Теперь попробуем вывести изображение. Добавим еще изображение для какой-нибудь новости через админку (рисунок 50, 51).

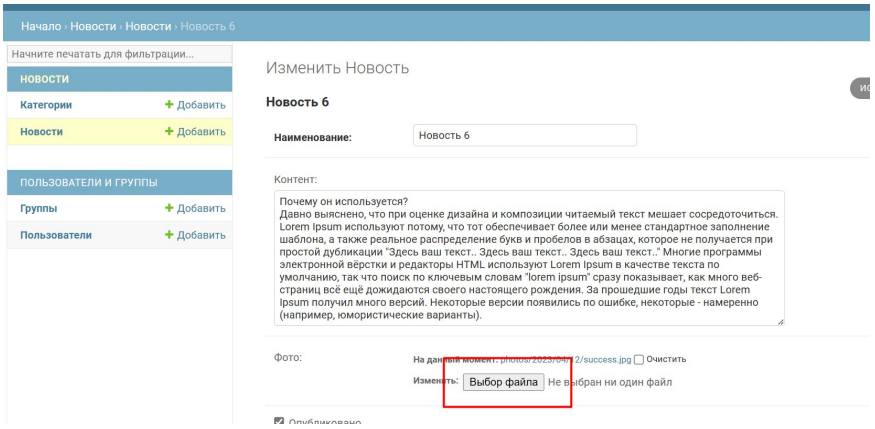


Рисунок 50

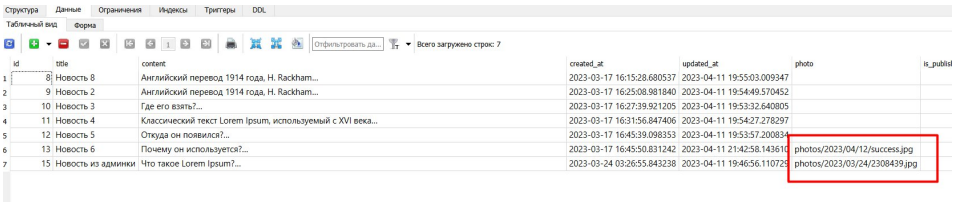


Рисунок 51

Чтобы вывести изображение необходимо использовать формат вывода изображения (рисунок 52)

```
<div class="card-body">
    <div class="media">
        {% if item.photo%}
              <img src="{{item.photo.url}}" alt=" ">
        {% else %}
               <img src="https://picsum.photos/200/300/?blur=2" alt=" ">
        {% endif %}
        <div class ='media-body'>
          <h5 class="card-title {%cycle 'text-danger' 'text-success' %}">{{item.title}}</h5>
          <p class="card-text">{{item.content}}</p>
          <a href="#" class="btn btn-primary">Read more...</a>
    </div>
    </div>
</div>
```

Рисунок 52

После обновления рисунок 53

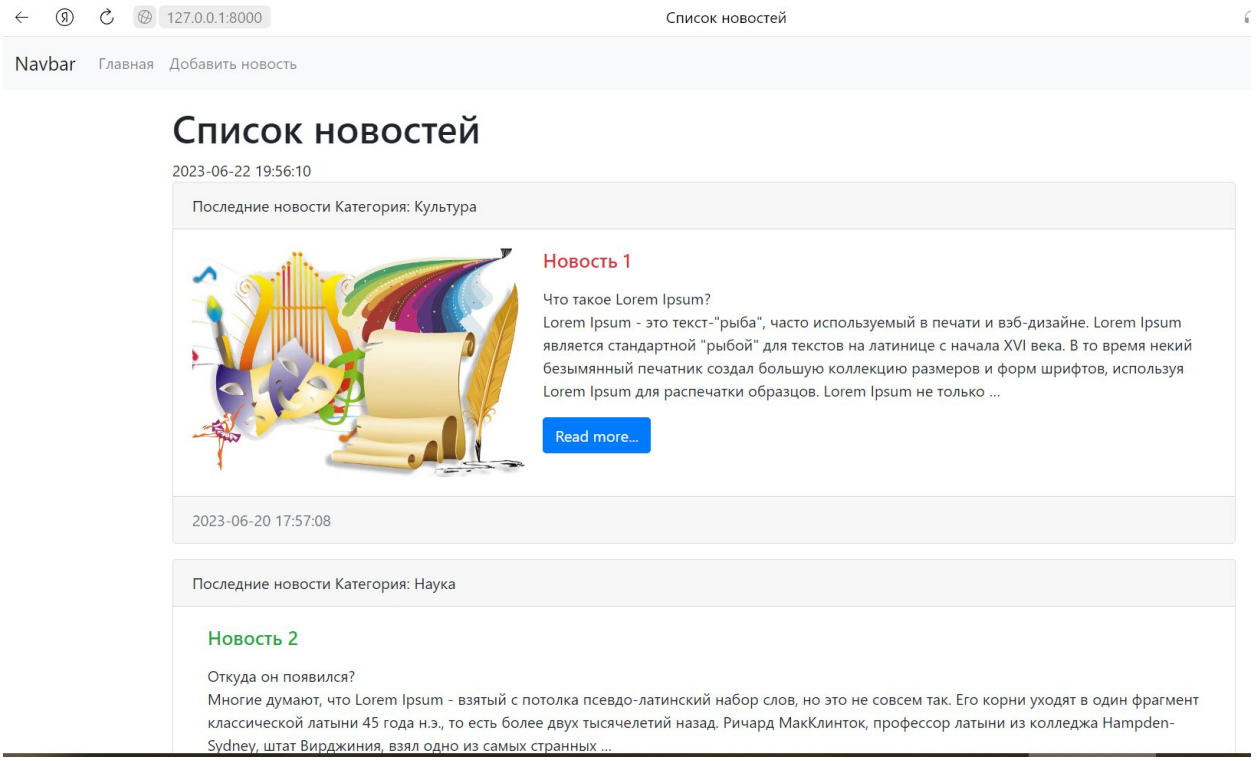


Рисунок 53

Полный листинг файла index.html приведен ниже:

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1,
shrink-to-fit=no">
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css
"
          integrity="sha384-
Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh"
crossorigin="anonymous">
    <title>{{ title }}</title>
```

```
</head>
<body>


<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <a class="navbar-brand" href="/">Navbar</a>
    <button class="navbar-toggler" type="button" data-toggle="collapse" data-
target="#navbarSupportedContent"
            aria-controls="navbarSupportedContent" aria-expanded="false" aria-
label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
    </button>

    <div class="collapse navbar-collapse" id="navbarSupportedContent">
        <ul class="navbar-nav mr-auto">
            <li class="nav-item"><a class="nav-link" href="/">Главная</a></li>
            <li class="nav-item"><a class="nav-link" href="/">Добавить
новость</a></li>
        </ul>
    </div>
</nav>

<div class="container mt-3">
    <h1>{{ title }}</h1>
    {% now 'Y-m-d H:i:s' %}
    <div class="row">
        <div class="col-md-12">
            {% for item in news %}
            <div class="card mb-3">
                <div class="card-header">
                    {{item.my_func}}
                    Категория: {{ item.category }}
                </div>
                <div class="card-body">
                    <div class="media">
                        {% if item.photo %}
                            <img src="{{ item.photo.url }}" alt="" width="350"
class="mr-3">
                        {% else %}
                            <img
src="https://picsum.photos/id/1060/350/235/?blur=2" alt="" class="mr-3">
                        {% endif %}
                        <div class="media-body">
                            <h5 class="card-title {% cycle 'text-danger'
'text-success' %}">{{ item.title }}</h5>
                            <p class="card-
text">{{ item.content|safe|linebreaks|truncatewords:50 }}</p>
                            <a href="#" class="btn btn-primary">Read
more...</a>
                        </div>
                    </div>

                </div>
                <div class="card-footer text-muted">
                    {{ item.created_at|date:"Y-m-d H:i:s" }}
                </div>
            </div>
            {% endfor %}
        </div>
    </div>
</div>
```

```
<script src="https://code.jquery.com/jquery-3.4.1.slim.min.js"
        integrity="sha384-
J6qa4849blE2+poT4WnyKhv5vZF5SrPo0iEjwBvKU7imGFAV0wwj1yYfoRSJoZ+n"
        crossorigin="anonymous"></script>
<script
src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"
        integrity="sha384-
Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo"
        crossorigin="anonymous"></script>
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js"
        integrity="sha384-
wfSDF2E50Y2D1uUdj0O3uMBJnjuUD4Ih7YwaYd1iqfktj0Uod8GCExl3Og8ifwB6"
        crossorigin="anonymous"></script>
</body>
</html>
```

Лабораторная работа № 7

Параметры в URL-запросах. Имена маршрутов

Усовершенствуем нашу страницу со списком категорий. Создадим sidebar и выведем список новостей.

Перейдем во views и в контроллере получим список новостей

Вначале импортируем модель Category (рисунок 1)

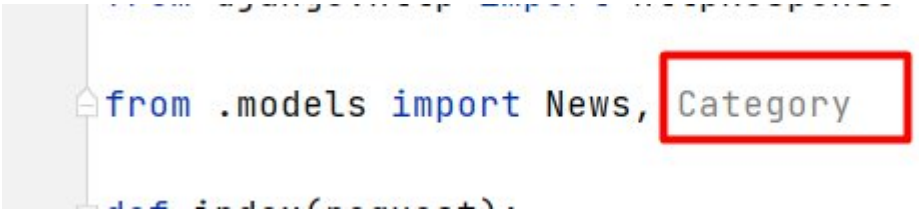


Рисунок 1

```
from .models import News, Category

def index(request):
    news = News.objects.all()
    categories=Category.objects.all()
    context={
        'news': news,
        'title': 'Список новостей',
        'categories': categories,
    }
    return render(request, template_name='news/index.html',context=context)
```

Рисунок 2

И все категории нам необходимо передать в представление (шаблон). Для этого в словаре context допишем переменную categories, которую предварительно создадим (рисунок 2).

Далее переходим в шаблон (index.html)

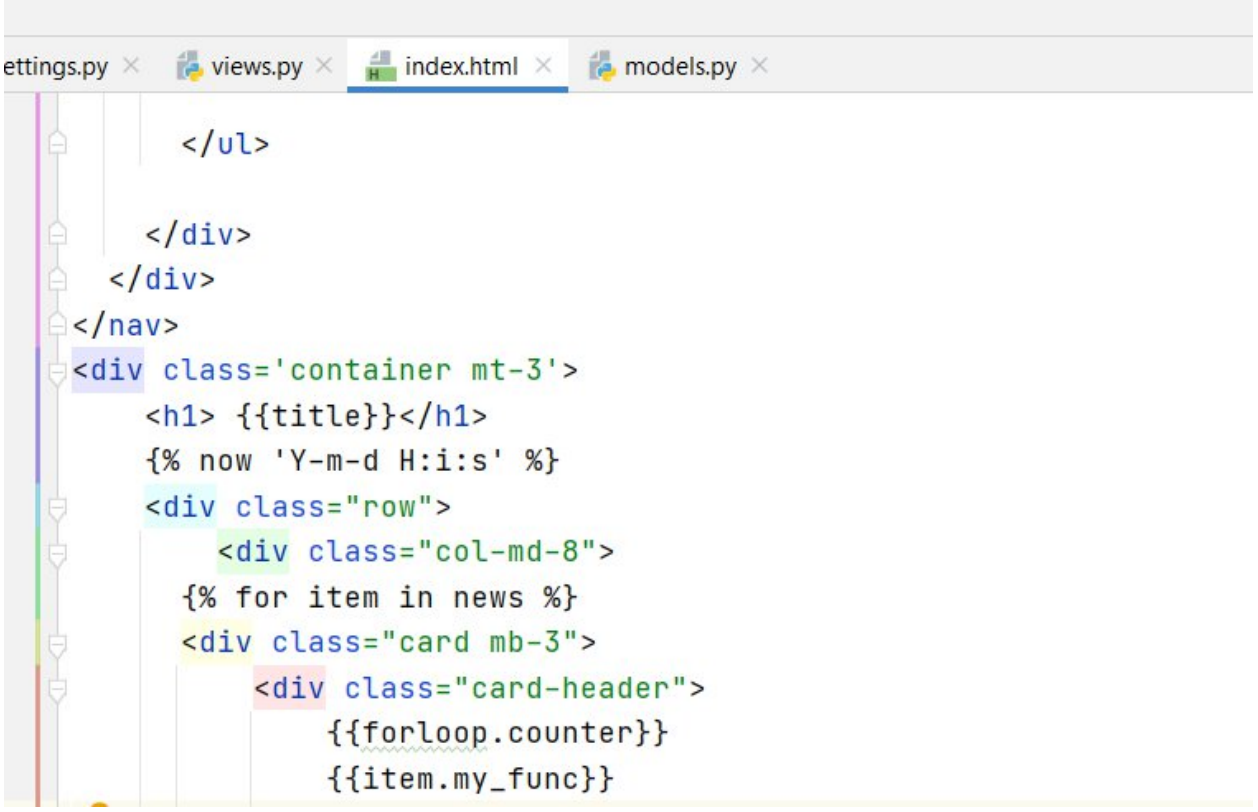


Рисунок - 3

Заменим число колонок 12 на 8

Создадим 4 колонки для сайдбар.



Рисунок 4

Четыре колонки мы отведем под сайдбар. Для этого воспользуемся снова https://getbootstrap.com/ и найдем какой-нибудь блок списков.

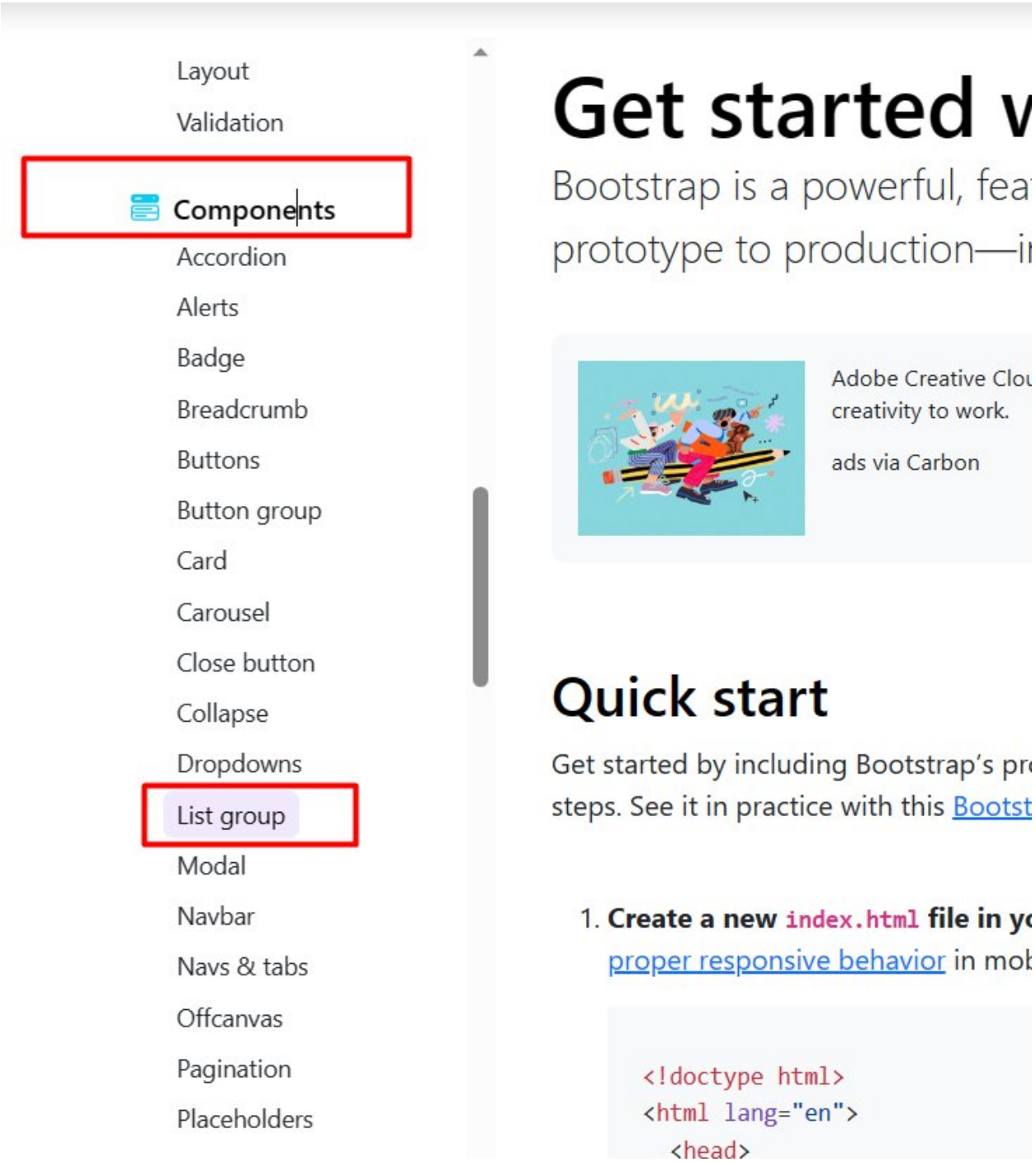


Рисунок 5

Выберим вариант шаблона списка со ссылками

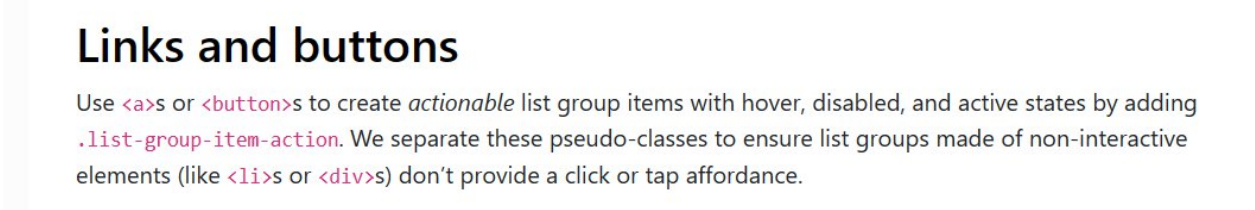


Рисунок 6

Скопируем данный шаблон (рисунок 7).

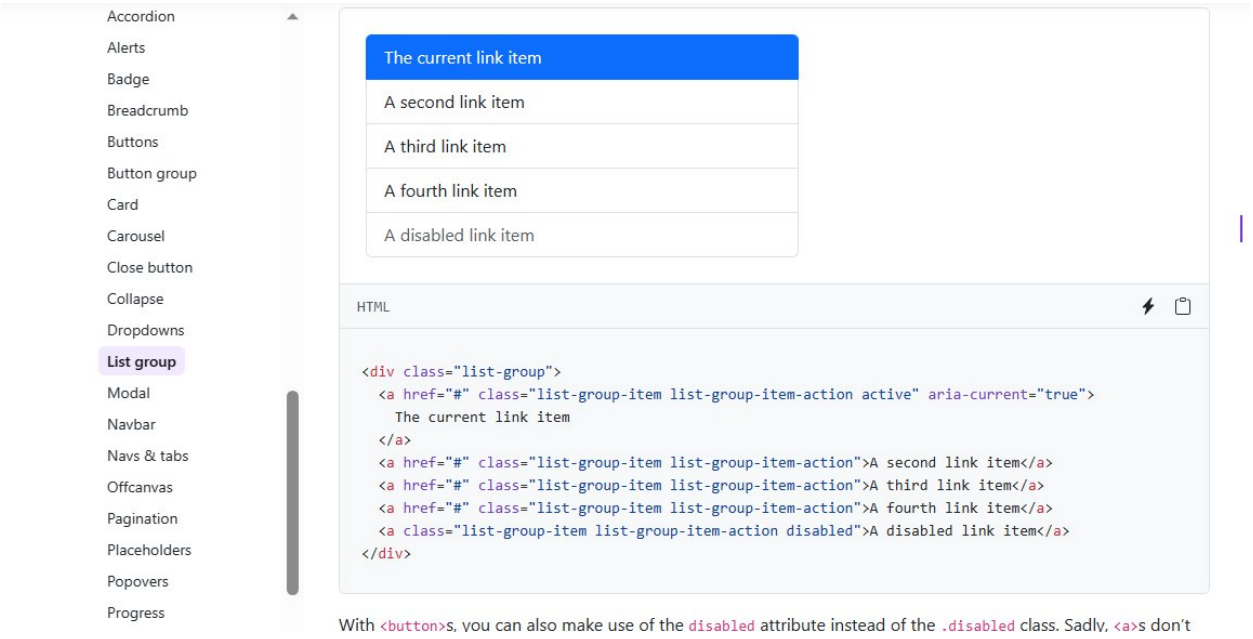


Рисунок 7

Текст шаблона приведен ниже.

```
<div class="list-group">
  <a href="#" class="list-group-item list-group-item-action active" aria-current="true">
    The current link item
  </a>
  <a href="#" class="list-group-item list-group-item-action">A second link item</a>
  <a href="#" class="list-group-item list-group-item-action">A third link item</a>
  <a href="#" class="list-group-item list-group-item-action">A fourth link item</a>
  <a class="list-group-item list-group-item-action disabled">A disabled link item</a>
</div>
```

и вставим в наш шаблон



Рисунок 8

Но для вывода списка наших категорий нам достаточна только одна строка со ссылкой, которую мы зациклим для вывода названия категории. Поэтом у удалим все ссылки, оставив одну и поместим в цикл как указано на рисунке 9.

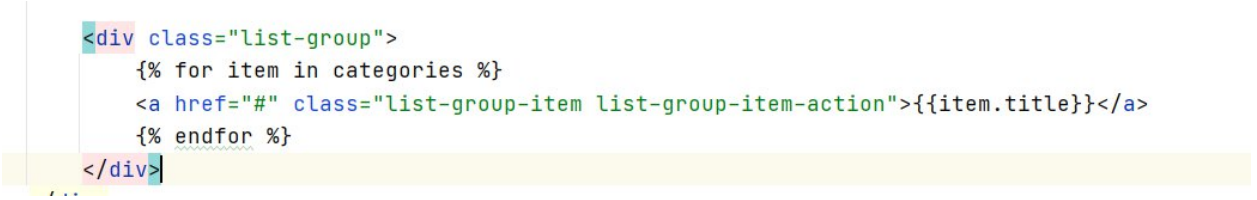


Рисунок 9

Обновим (рисунок 10) и на странице появится меню сайдбара.

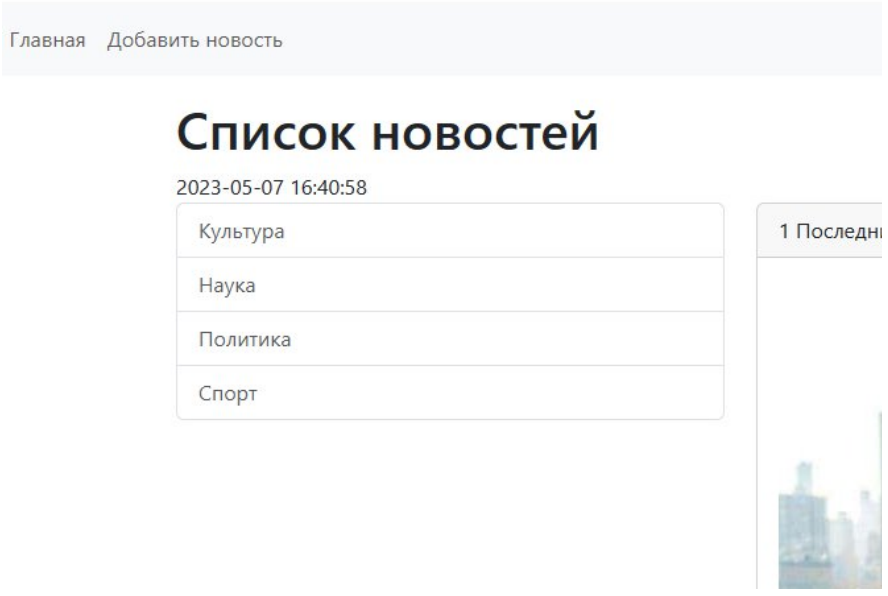


Рисунок 10

Теперь для каждой категории необходимо прописать ссылку (рисунок 11)

```
<div class="list-group">
    {% for item in categories %}
    <a href="/category/{{ item.pk }}" class="list-group-item list-group-item-action">{{ item.title }}</a>
    {% endfor %}
</div>
```

Рисунок 11

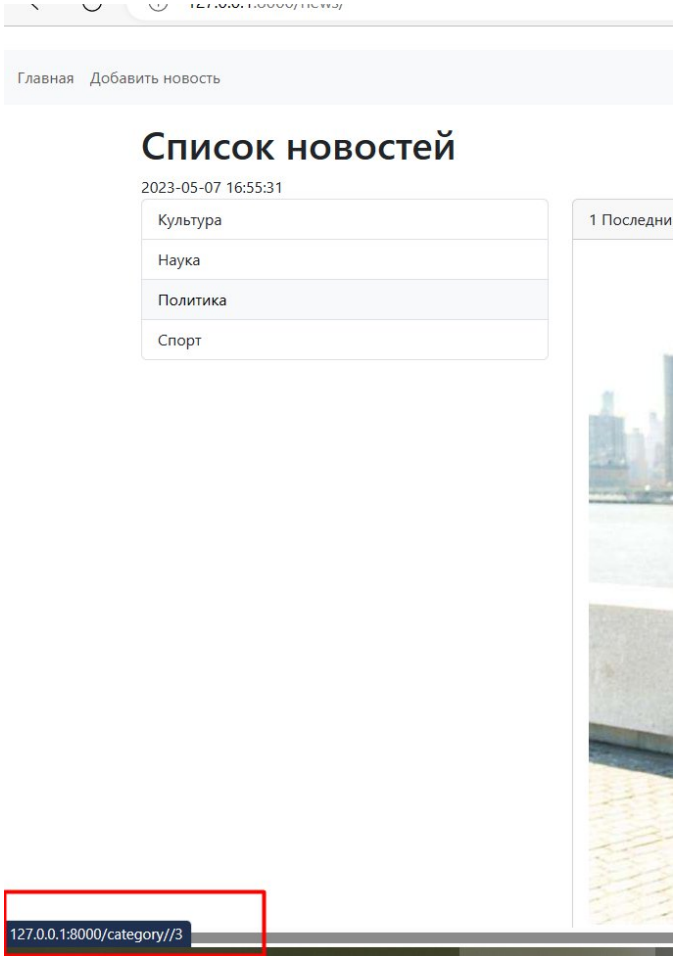


Рисунок 12

Следующим шагом будет организация переход по ссылкам. В настоящий момент, нажав на ссылку на странице мы получаем ошибку (рисунок 13).

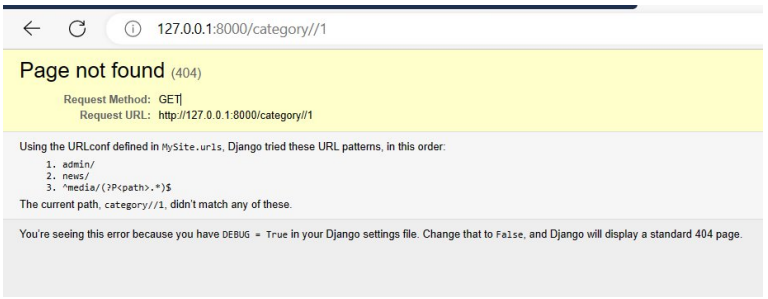


Рисунок 13

Ошибка сообщает о том, что нет у нас такого URL адреса. Поэтому необходимо создать такой URL адрес. А также необходимо создать контроллер в файле views, который будет принимать этот запрос. И по номеру категории доставить все записи с этой категорией.

Создадим контроллер-функцию, которая будет принимать параметры request, category_id. (рисунок 14)

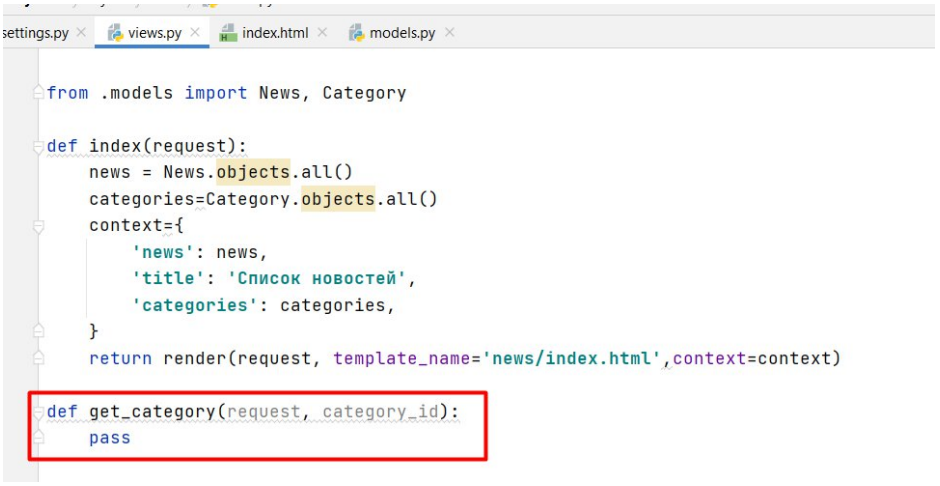


Рисунок 14

И теперь нам необходимо к ней обратиться через файл urls.py, который находится в приложении news и прописать адрес, который будет в адресной строке сайта (рисунок 15).



Рисунок 16

Допишем наш контроллер get_category (рисунок 17)

```
def get_category(request, category_id):
    news=News.objects.filter(category_id=category_id)
    categories = Category.objects.all()
    category=Category.objects.get(pk=category_id)
    return render(request, 'news/category.html', {'news':news, 'categories':categories,
                                                  'category': category})
```

Рисунок 17

В контроллере мы написали, что он возвращает нам страницу category.html.

Перейдем к созданию этой страницы. Для этого в папке шаблонов приложения news создадим новый html файл (рисунок 18) – файл category.html

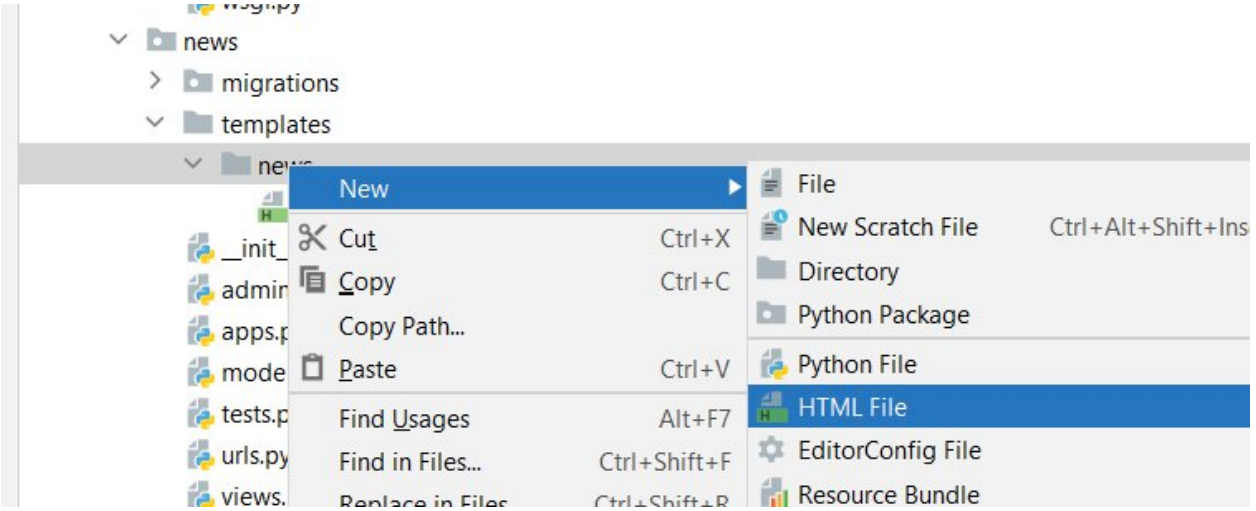


Рисунок 18

После создания этого файла, удалим все содержимое новой страницы. Скопируем код из файла index.html, вставим в файл category.html и отредактируем его.

В заголовочной части поменяем title на category.title (рисунок 19)

```
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/b
  <title>{{category.title}}</title>
</head>
<body>
```

Рисунок 19

После обновления страницы и перехода по любой ссылке категории получим страницу со статьями выбранной категории (рисунок 20).

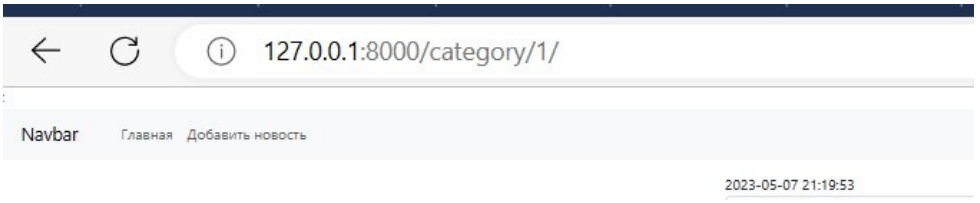


Рисунок 20

Но наш код имеет недостатки, поскольку, например, мы в нескольких местах прописываем путь к категориям

`"/category/{{ item.pk }}"`

и если, по каким-то причинам, нам придётся изменить адрес страницы, то придется его менять в нескольких местах. И вторая проблема, связана с повторяющимися шаблонами. И если, необходимо будет вводит правки в один шаблон, то эти же правки надо будет вносить и в другие шаблоны.

Например, изменим размер колонок в представлении index.html (рисунок 21)

Для категорий:

`class="col-md-3"` изменим `class="col-md-4"` и для основных публикаций `class="col-md-8"` изменим `class="col-md-9"`



Рисунок 21

Тоже самое сделаем и в шаблоне category.html

Теперь перейдем к решению проблем со ссылками. Функция path позволяет создавать псевдоимена, т.е. названия маршрутов на основании которых мы можем Django будет строить нужную нам ссылку на основании пути, который будет указан (рисунок 22).



Рисунок 22

Перейдем в шаблон index.html и воспользуемся именем нашего маршрута, которое мы ему дали для главной страницы home. Также воспользуемся тегом url (см. документацию

https://django.fun/ru/docs/django/4.1/ref/templates/builtins/#url)

Пропишем имя маршрута в шаблоне index.html в теге ссылки на главную страницу (рисунок 23) новую ссылку через имя маршрута (рисунок 24)

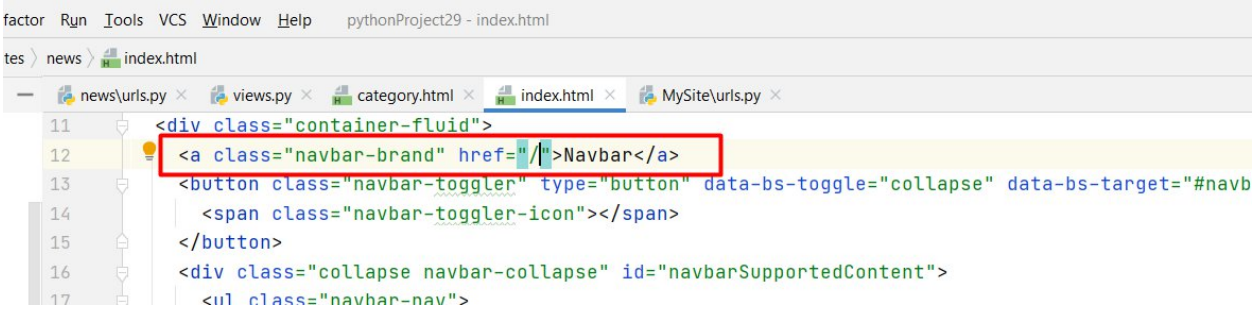


Рисунок 23

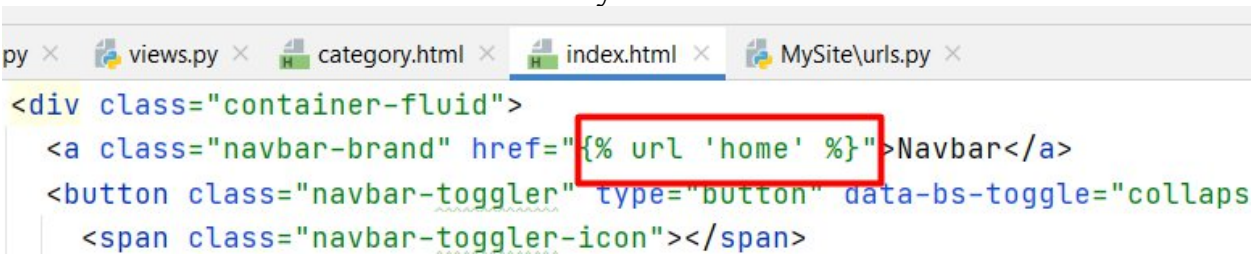


Рисунок 24

Также поменяем эту ссылку в разделе Главная и Добавить новость (рисунок 25, 26)

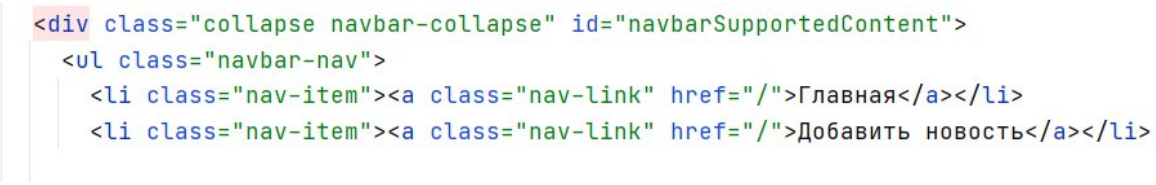


Рисунок 25

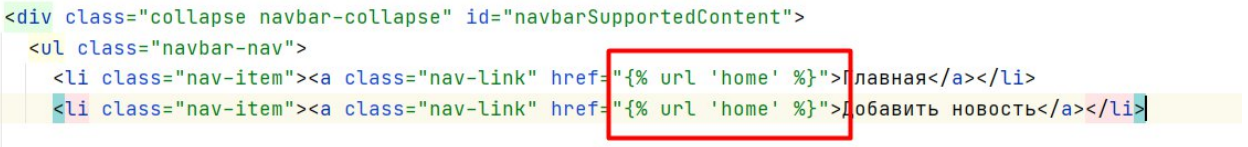


Рисунок 26

Также поменяем обращение к категории по через имя маршрута (рисунок 27)



Рисунок 27

И переименуем его соответственно в category (рисунок 28)



Рисунок 28

Теперь переходим в шаблон category.html и проведем, аналогичные переименования (рисунок 29 и 30)

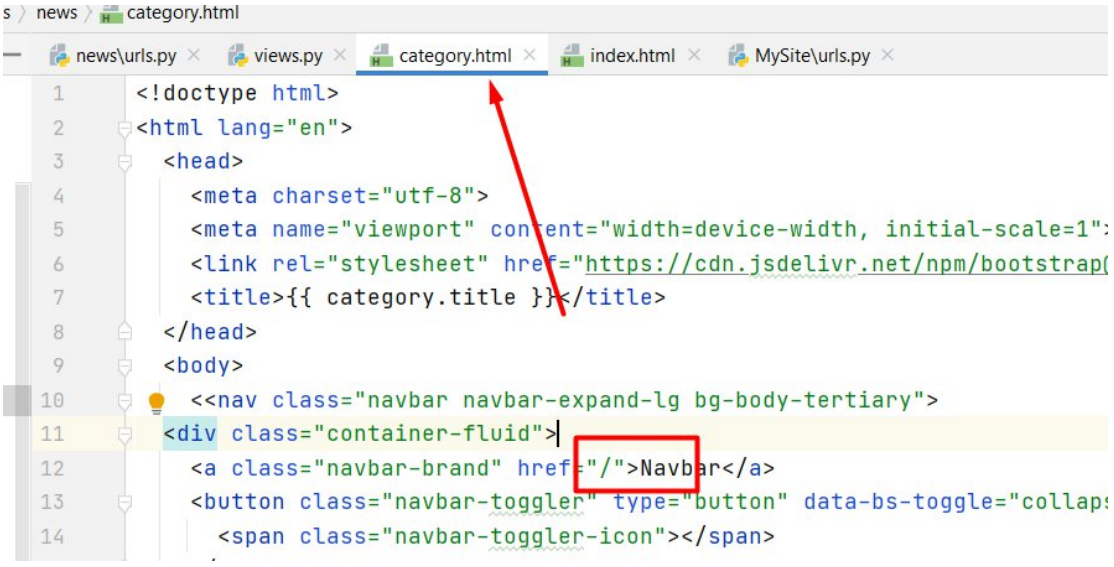


Рисунок 29



Рисунок 30

И для категории (рисунок 31)



Рисунок 31

В шаблонах index.html и category.html сделаем ссылку в коде карточки (рисунок 32, 33)

```
<div class="card mb-3">
    <div class="card-header">
        {{forloop.counter}}
        {{item.my_func}}
        Категория {{item.category}}
    </div>
    <div class="card-body">
        <div class="media">
```

Рисунок 32

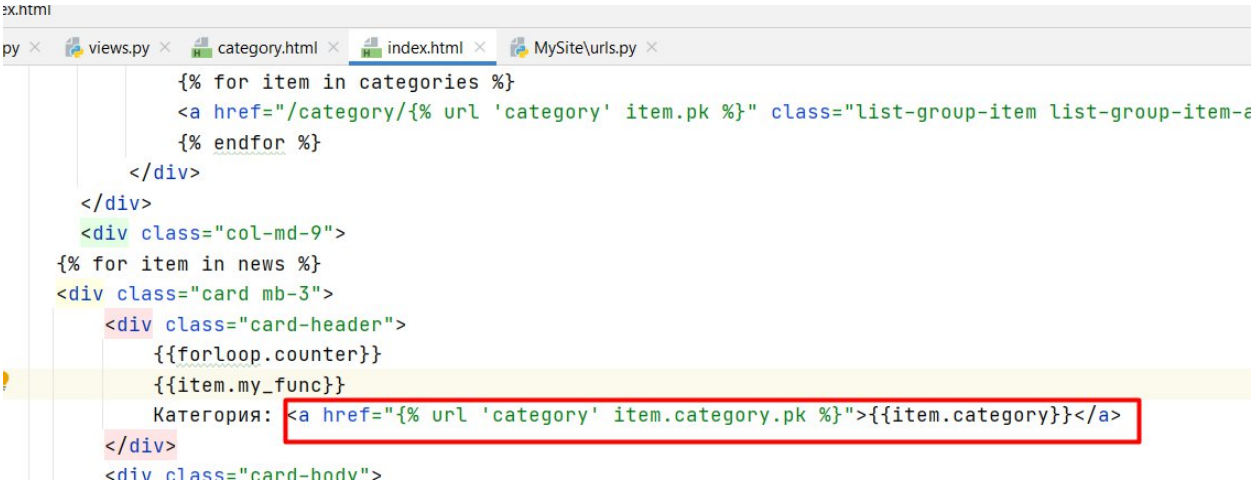


Рисунок 33

Лабораторная работа № 8

Наследование шаблонов.

Создадим некоторый базовый шаблон, который затем будем использовать как основу, а остальные прочие шаблоны будут его наследовать. Наследование происходить благодаря такому тегу как exstends с параметром шаблона, который мы будем наследовать.

В Django принято называть базовые шаблоны base.html. В этом базовом шаблоне не используются теги <html>, <body>, используются block (см. документацию https://djangodoc.ru/3.1/ref/templates/language/?ysclid=lhedu5gcp8110619073).

Как известно, шаблоны хранятся в папке templates, расположенной в папке приложения, в нашем случае, в папке news.

Но для хранения базового шаблона Django необходимо в корне проекта MySite создать папку (рисунок 1)

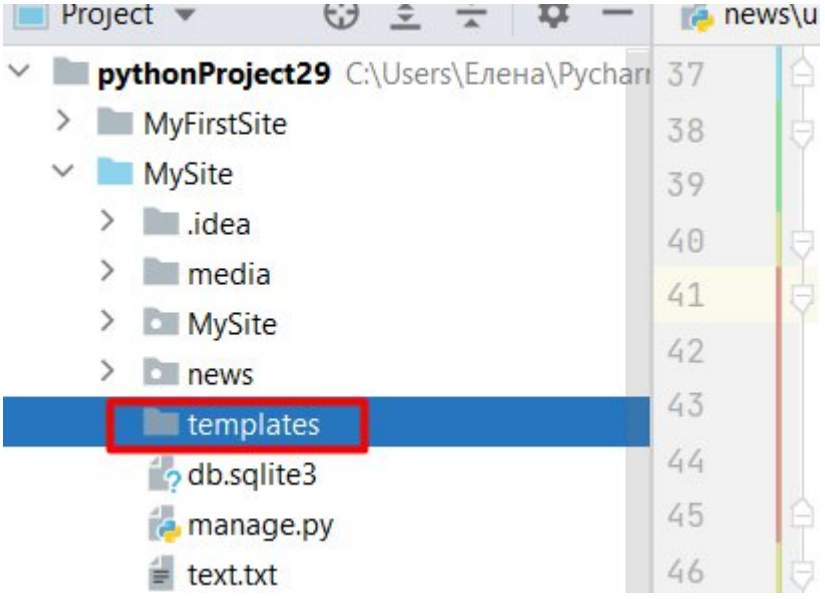


Рисунок 1

В этой папке создаем html файл, который в соответствии с конвенцией называем base.html (рисунок 2, 3).

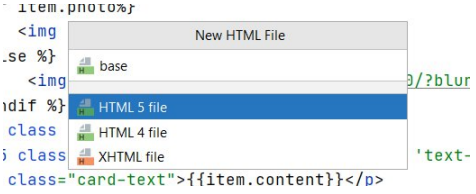


Рисунок 2

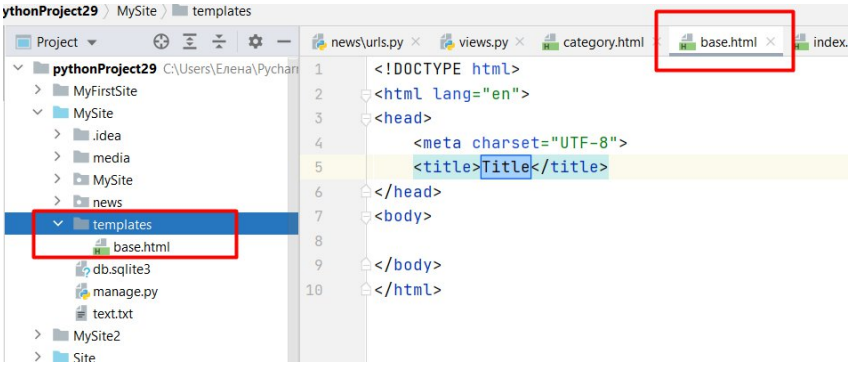


Рисунок 3

Скопируем содержимое файла index.html в базовый шаблон. Сделаем из базового шаблона блоки, которые будут наполняться контентом.

В нашем случае таким блоком будет блок {{title}}(рисунок 4), в котором должна быть некоторая переменная. Поэтому {{title}} убираем (рисунок 4) и пишем (рисунок 5)

```
{% block title %} {% endblock %}
```

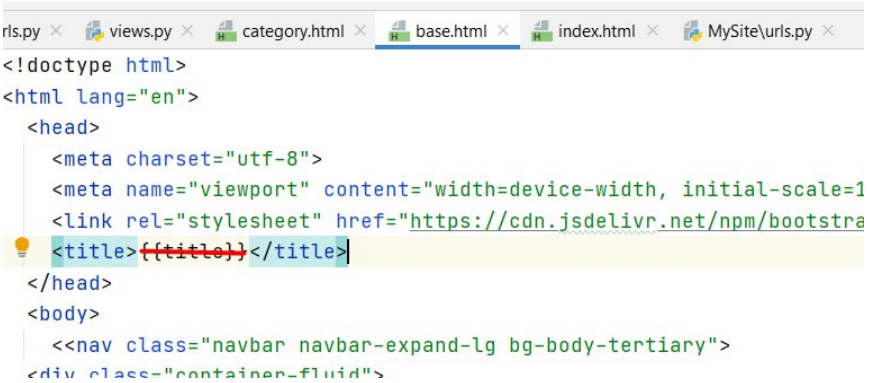


Рисунок 4



Рисунок 5

Между блоками определим какой-нибудь текст названия сайта, например, «Новости со всего мира» (рисунок 6).



Рисунок 6

Продолжим редактировать код шаблона. Удалим строки, указанные на рисунке 7. Мы дальше выведем в контентной части.

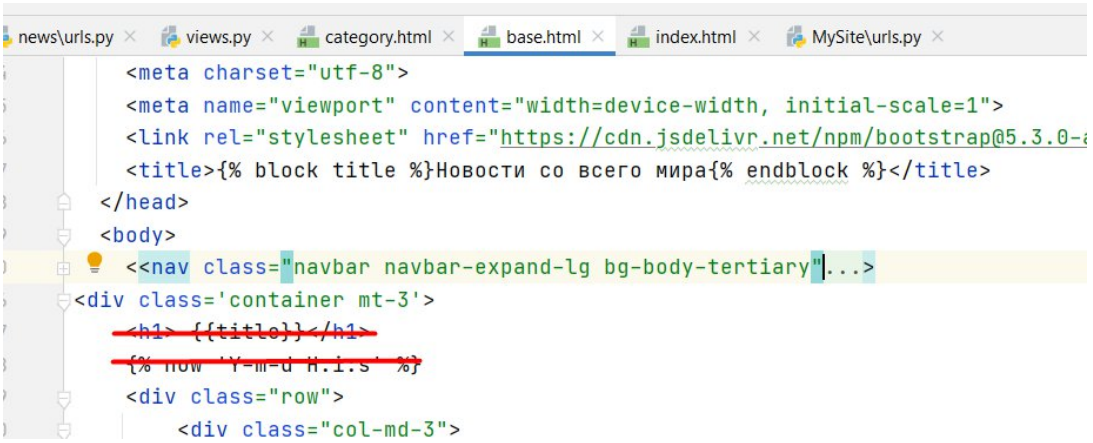


Рисунок 7

В данном представлении мы имеем Navbar, sidebar, content, которые нам необходимо вынести в ключевые блоки. Поэтому внутри row мы напишем блок для sidebar и content (рисунок 8).



Рисунок 8



Рисунок 9

Итого, имеем следующий базовый шаблон (рисунок 10). И этого шаблона, все динамические части, которые мы получали из контроллера, удалили и вместо них поставили блоки.

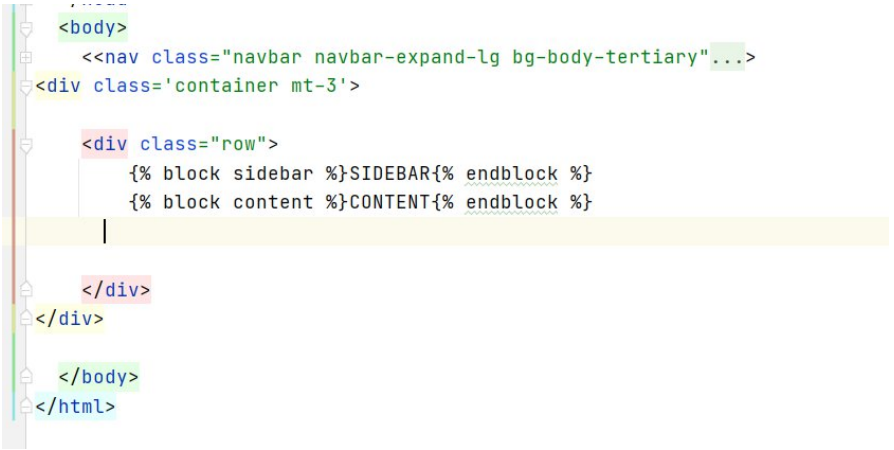


Рисунок 10

Таким образом, если дублируются в нескольких шаблонах содержимое, то их лучше представлять в виде блоков в базовом шаблоне. Это сделает ваш шаблон более гибким и в случае каких-либо правок достаточно будет поправить в базовом шаблоне.

Перейдем к оформлению Navbar. Для этого воспользуемся следующим тегом include, который позволяет подгрузить часть шаблона. Для этого разобьём шаблон на части. И для этого в папке templates создадим еще одну папку inc (сокращенное от include) (рисунок 11)

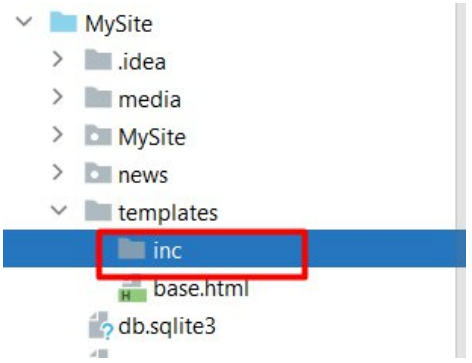


Рисунок 11

Внутри этой папки создадим html файл _nav.html. Включающим частям принято давать имена, начинающиеся с нижнего подчеркивания (рисунок 12).



Рисунок 12

Из базового шаблона скопируем фрагмент Navbar (рисунок 13)

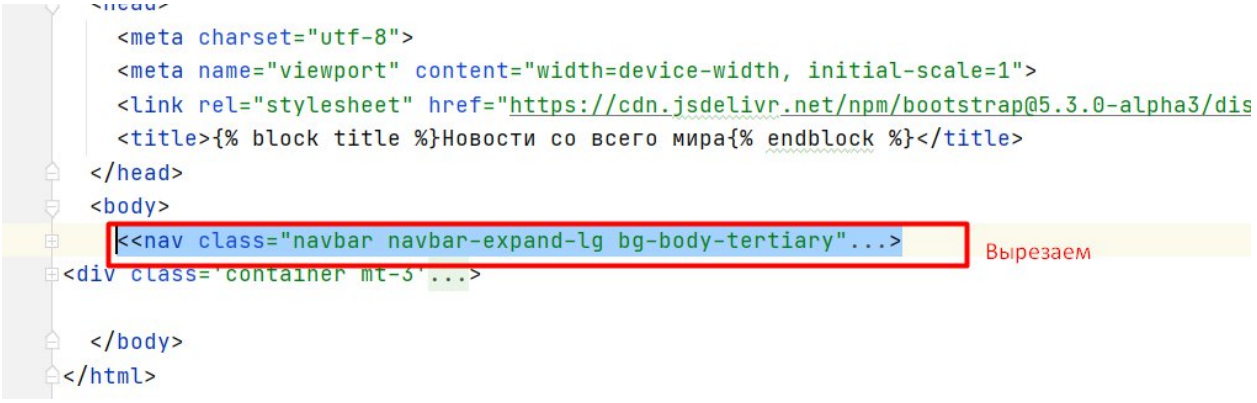


Рисунок 13

Копируемый текст

```
<<nav class="navbar navbar-expand-lg bg-body-tertiary">
  <div class="container-fluid">
    <a class="navbar-brand" href="{% url 'home' %}">Navbar</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
data-bs-target="#navbarSupportedContent" aria-
controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle
navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarSupportedContent">
      <ul class="navbar-nav">
        <li class="nav-item"><a class="nav-link" href="{% url
'home' %}">Главная</a></li>
        <li class="nav-item"><a class="nav-link" href="{% url
```

```
'home' %}">Добавить новость</a></li>

      </ul>

    </div>
  </div>
</nav>
```

И вставляем в новый файл _nav.html (рисунок 14).

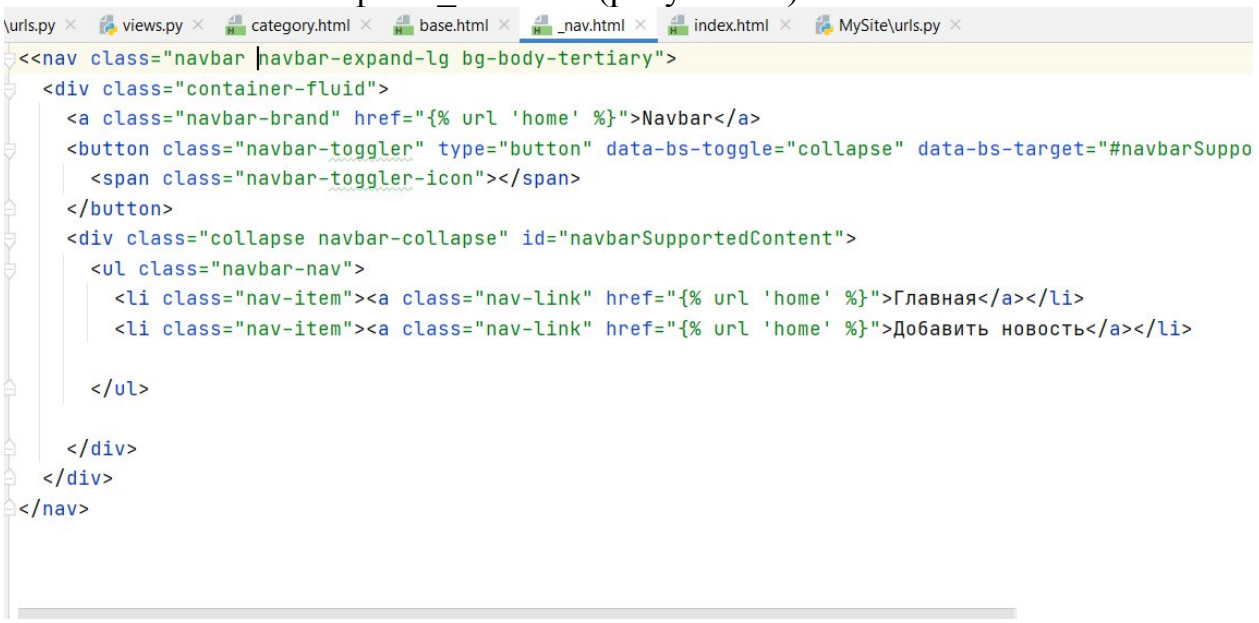


Рисунок 14

В основном шаблоне Navbar подключим с помощью include (рисунок 15).



Рисунок 15

Таким образом, мы получили базовый шаблон, который можно будет наследовать.

Возвращаемся в файл index.html, в котором будем вносить правки.

В наследуемом шаблоне нам необходимо указать, что будем использовать базовый шаблон. Для этого используем тег extends и начнем писать верхней части index.html, а также предопределим блоки, которые создали в базовом шаблоне (рисунок 16)



Рисунок 16

Из кода в нижней части (старый код) вырежем строки из sitebar (рисунок 17) и вставим в блок sitebar (рисунок 18)

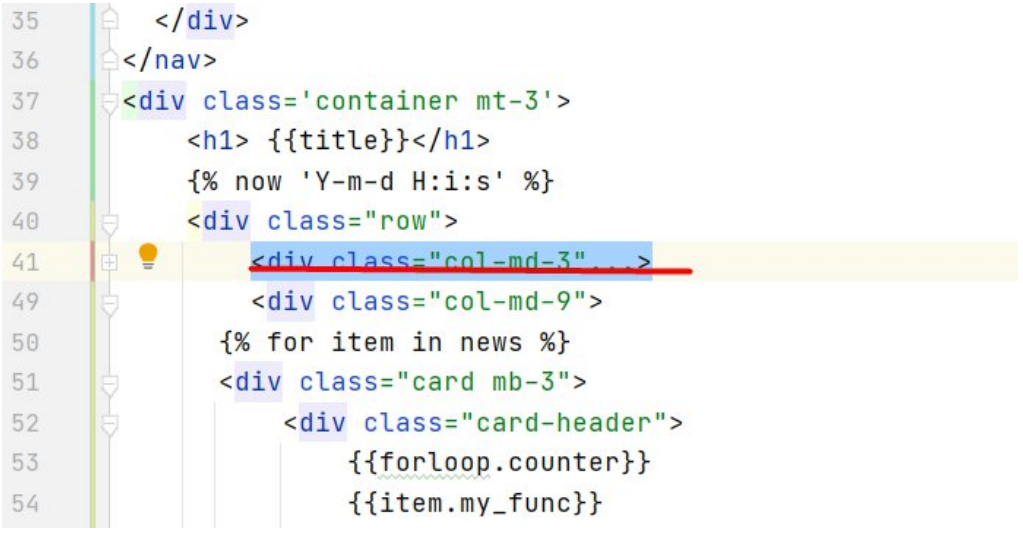


Рисунок 17

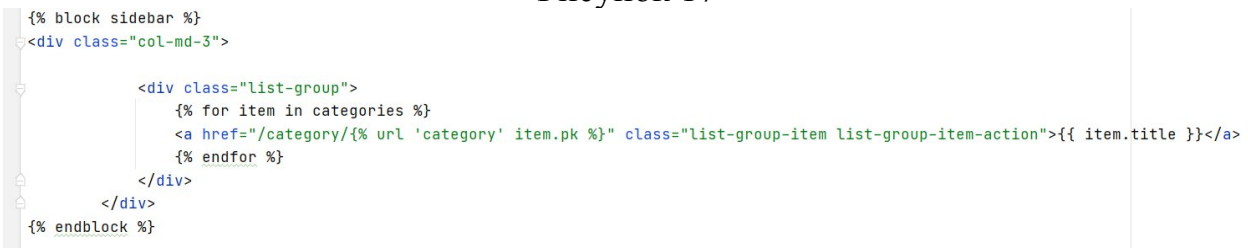


Рисунок 18

Тоже самое проделываем с контентом (рисунок 19, 20)

```
    <h1> {{title}}</h1>
    {% now 'Y-m-d H:i:s' %}
    <div class="row">

        <div class="col-md-9"...>

    </div>
</div>

  </body>
</html>
```

Рисунок 19

```
{% block content %}
 <div class="col-md-9"...>

{% endblock %}
```

Рисунок 20

Остальной код удалим из шаблона, поскольку все остальное будет взято из родительского шаблона.

Запустим сервер или обновим страницу в браузере. В результате получим ошибку (рисунок 21), который сообщает, что не найден базовый шаблон.

TemplateDoesNotExist at /

base.html

Request Method:	GET
Request URL:	http://127.0.0.1:8000/
Django Version:	4.1.7
Exception Type:	TemplateDoesNotExist
Exception Value:	base.html
Exception Location:	C:\Users\Елена\PycharmProjects\pythonProject29\venv\Lib\site-packages\django\template\backends\django.py, line 84, in reraise
Raised during:	news.views.index
Python Executable:	C:\Users\Елена\PycharmProjects\pythonProject29\venv\Scripts\python.exe
Python Version:	3.11.2
Python Path:	['C:\\Users\\Елена\\PycharmProjects\\pythonProject29\\MySite', 'C:\\Users\\Елена\\AppData\\Local\\Programs\\Python\\Python311\\python311.zip', 'C:\\Users\\Елена\\AppData\\Local\\Programs\\Python\\Python311\\DLLs', 'C:\\Users\\Елена\\AppData\\Local\\Programs\\Python\\Python311\\Lib', 'C:\\Users\\Елена\\AppData\\Local\\Programs\\Python\\Python311', 'C:\\Users\\Елена\\PycharmProjects\\pythonProject29\\venv', 'C:\\Users\\Елена\\PycharmProjects\\pythonProject29\\venv\\Lib\\site-packages']
Server time:	Mon, 08 May 2023 17:08:22 +0000

Рисунок 21

Дело в том, что Django по умолчанию ищет шаблоны в папках templates приложения. Базовый шаблон храниться в папке templates в корне проекта. Поэтому чтобы Django знал, где хранятся дополнительно шаблоны, необходимо обратиться к настройкам нашего приложения, т.е. к файлу settings.py и обратиться к настройкам TEMPLATES. TEMPLATES – это список словарей, в котором Django надо указать, где искать шаблоны (рисунок 22). В DIRS прописываем конструкцию (рисунок 23)

```
os.path.join(BASE_DIR), 'templates'
```

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Рисунок 22

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR), 'templates'],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Рисунок 23

Обновим страницу и как видим, ошибка исчезла и снова получили страницу с контентом. Обратим внимание, что название нашего сайта соответствует названию, который мы прописала в шаблоне «Новости всего мира» (рисунок 24)

Рисунок 24

Для переопределения блока title, например (рисунок 25), (рисунок 26)



Рисунок 25

Рисунок 26

По аналогии, мы должны сделать дочерний шаблон для страницы категории.

Теперь все изменения мы можем проводить единожды. В нашем пример, в файле _nav.html заменим Navbar на более понятное пользователю наименование «Новостной сайт» (рисунок 27).

```
base.html | _nav.html | index.html | category.html | settings.py
<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <a class="navbar-brand" href="{% url 'home' %}">Navbar</a>
    <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupportedContent"
            aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
    </button>

    <div class="collapse navbar-collapse" id="navbarSupportedContent">
        <ul class="navbar-nav mr-auto">
            <li class="nav-item"><a class="nav-link" href="{% url 'home' %}">Главная</a></li>
            <li class="nav-item"><a class="nav-link" href="{% url 'home' %}">Добавить новость</a></li>
        </ul>
    </div>
</nav>
```

Рисунок 27

После обновления - рисунок 28

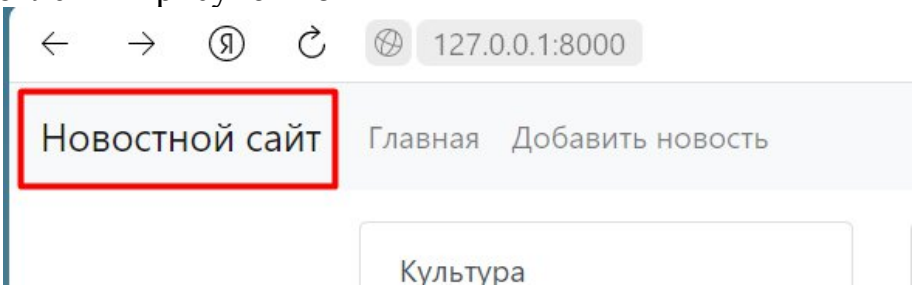


Рисунок 28

Рассмотрим пользовательские теги шаблонов. В views.py мы несколько раз получаем объекты, т.е. происходит дублирование кодов. И если функций в контроллере станет больше, то опять столкнемся с повторениями кода (рисунок 29).

```
def index(request):
    news=News.objects.all()
    categories=Category.objects.all()
    context={
        'news': news,
        'title': 'Список новостей',
        'categories': categories,
    }
    return render(request, template_name='news/index.html', context=context)

def get_category(request, category_id):
    news = News.objects.filter(category_id=category_id)
    categories = Category.objects.all()
    category = Category.objects.get(pk=category_id)
    return render(request, 'news/category.html', {'news': news, 'categories': categories, 'category': category})
```

Рисунок 29

Решим эту проблему за счет работы с кастомными тегами.

Лабораторная работа № 9.

Пользовательские теги шаблона

Цель работы: научиться создавать пользовательские теги шаблонов.

Возникает проблема, связанная с тем что во views мы получаем список категорий. Меню категорий это повторяющиеся части, которые нам нужны на каждой странице, которые в результате дублируются. Со временем, если контролеров станет больше, то и в них нам надо будет делать аналогичные повторения. Поэтому необходимо избежать этого дублирования, которое можно сделать различными способами, например, создание классов, которые могли бы наследоваться (по технологии Django, которые называются миксинами) и которые мы рассмотрим в другой раз. В рамках этой работы мы рассмотрим тему работы с тегами, которые называются кастомными тегами.

До этого мы с вами рассматривали встроенные теги такие как for, if и др.

Django позволяет создавать свои собственные теги, более подробную информацию о которых можно найти в документации по Django (рисунок 1).

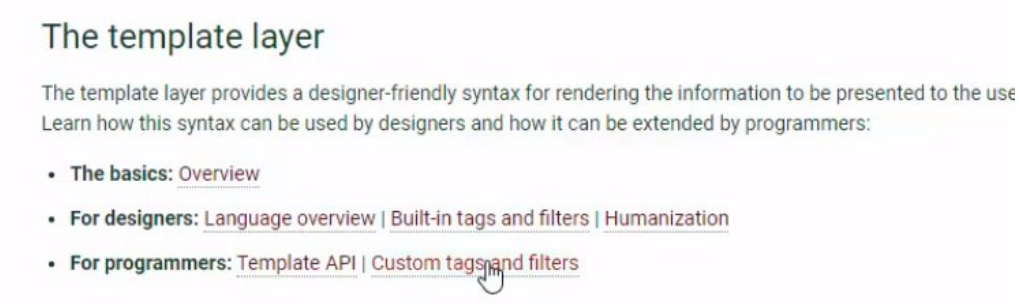


Рисунок 1

Существует два вида собственных тега: simple (простой) и inclusion (включение).

Simple – это тег, который вернет просто данные и с этими данными мы можем что-то делать.

Например в нашем случае, возвращается список категорий и в шаблоне мы проходим в цикле по этим категориям.

Inclusion - делает то же самое что и simple, но кроме этого он умеет еще рендерить шаблоны, т.е. он возвращает не только эти данные, но и их представление.

Приступим к оптимизации нашего и кода и начнем с правок нашего шаблона base.html. В этом файле будем подключать не только блоки sidebar (рисунок 3), но какую-нибудь общую разметку (рисунок 4).

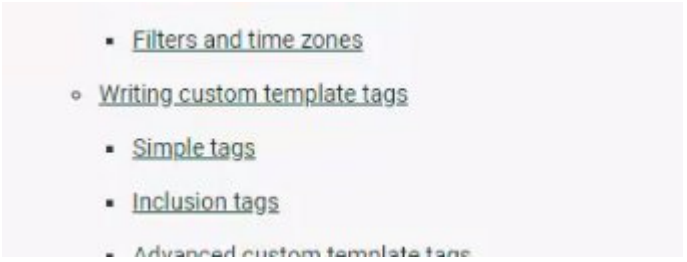


Рисунок 2

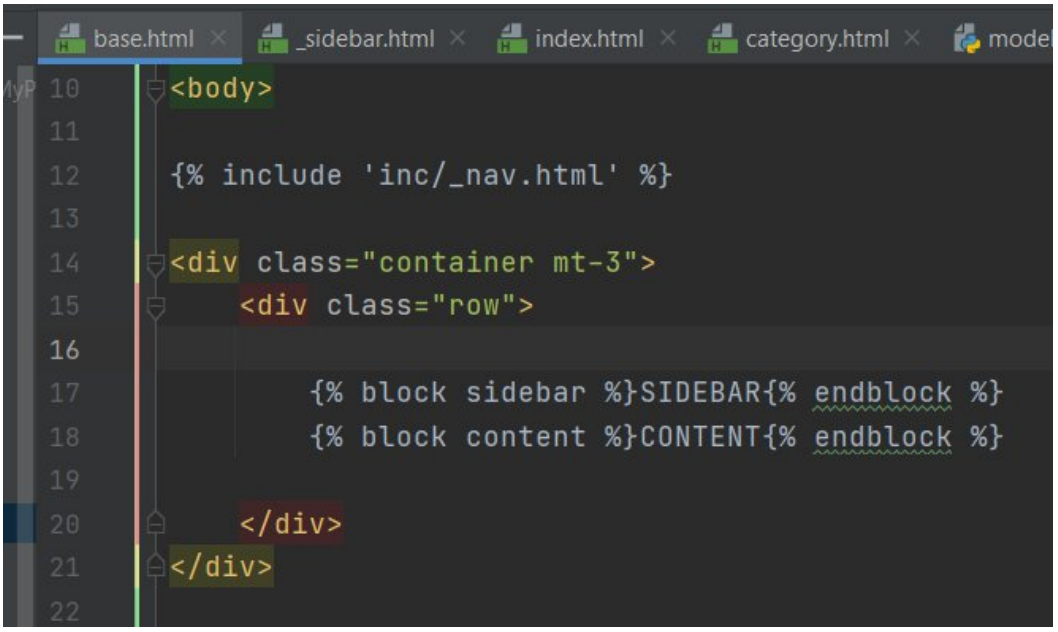


Рисунок 3

И добавим блоки (рисунок 3)

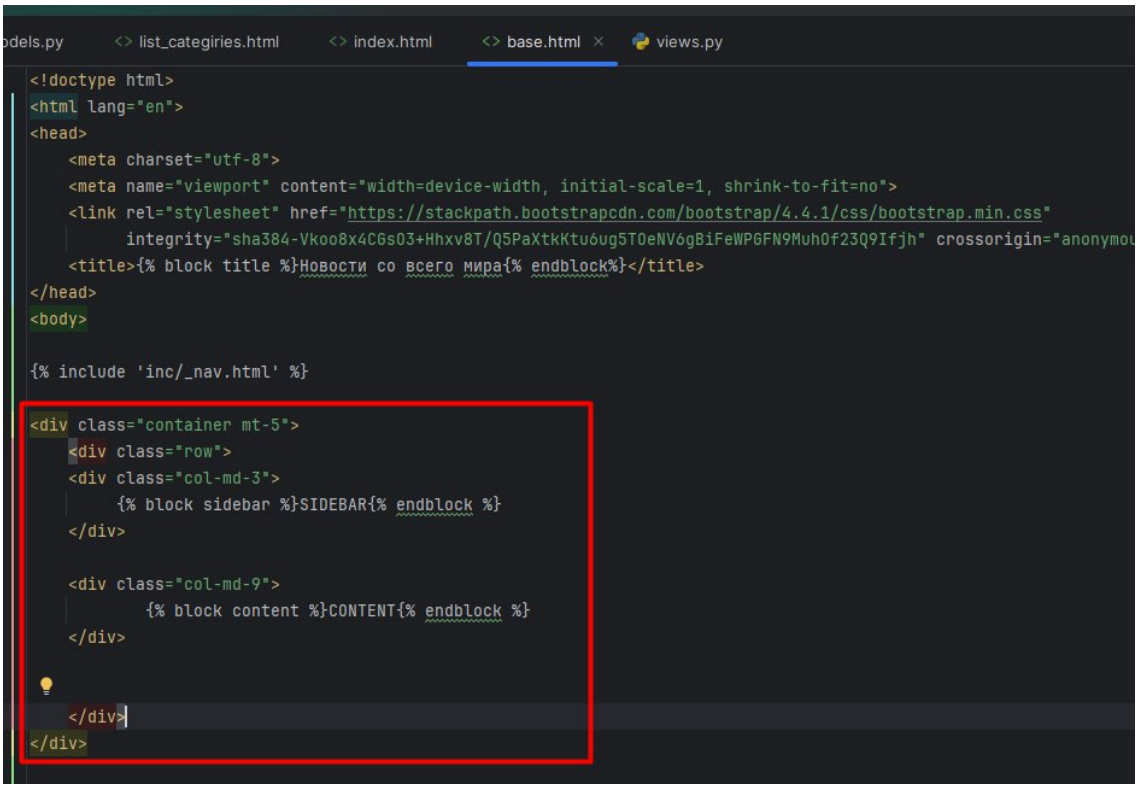


Рисунок 4

Уберем лишнюю разметку в файле index.html.

Полный исправленный текст см.ниже:

```
{% extends 'base.html' %}

{% block title %}
    {{ title }} :: {{ block.super }}
{% endblock %}

{% block sidebar %}

    <div class="list-group">
        {% for item in categories %}
        <a href={{item.get_absolute_url}} class="list-
```

```
group-item list-group-item-action">{{ item.title }}</a>
        {%                    endfor                    %}
    </div>

{%                      endblock                      %}

{%              block              content              %}

    {%       for       item       in       news       %}
    <div                class="card                mb-3">
        <div                          class="card-header">
            Категория:  <a  href="{%  url  'category'
item.category.pk     %}">{{     item.category     }}</a>
        </div>
        <div                           class="card-body">
            <div                             class="media">
                {%        if        item.photo        %}
                <img  src="{{  item.photo.url  }}"  alt=""
width="350"                                  class="mr-3">
                {%                else                %}
                <img
src="https://picsum.photos/id/1060/350/235/?blur=2"
alt=""                                       class="mr-3">
                {%                endif                %}
                <div                   class="media-body">
                    <h5                        class="card-
title">{{                item.title                }}</h5>
                    <p                         class="card-
text">{{   item.content|safe|linebreaks|truncatewords:50
```

```
}}</p>
                    <a    href="#"    class="btn    btn-
primary">Read                                  more...</a>
                </div>
            </div>

        </div>
        <div       class="card-footer       text-muted">
            {{   item.created_at|date:"Y-m-d   H:i:s"   }}
        </div>
    </div>
    {%                      endfor                      %}

{% endblock %}
```

Такие же правки выполним и в файле categories.html

Создадим простой тег (simple teg). Он создается в приложении. Для этого создадим новую папку templatetags (рисунок 5)

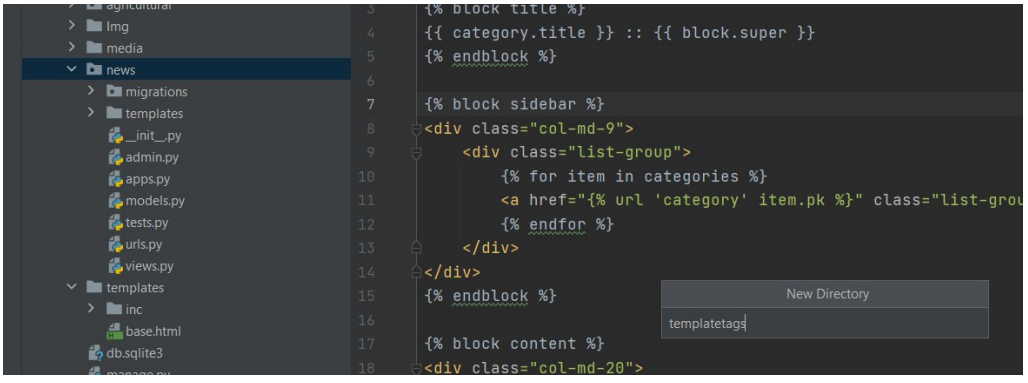


Рисунок 5

Внутри папки templatetags создаем файл __init__.py И в этой же папке создадим файл news_tags.py, в котором мы будем создавать пользовательские теги и в дальнейшем импортировать.

В этом файле импортируем модуль template. Поскольку мы будем работать с моделью Category, то необходимо нам импортировать и ее .

```
from django import template
from news.models import Category
```

Для того, чтобы все заработало нам необходимо зарегистрировать наш templatetag и это делается строчкой кода, вызовом метода Library()

```
register=template.Library()
```

Теперь создадим функцию, которая будет возвращать наши категории. Функция имеет следующий код:

```
def get_categories():
return Category.objects.all()
```

Чтобы все это отработало, то используется понятие декораторы. Декораторы – это возможность изменить поведение функции.

Декораторы - это обёртки вокруг Python-функций (или классов), которые изменяют работу того, к чему они применяются. Декоратор максимально абстрагирует собственные механизмы. А синтаксические конструкции, используемые при применении декораторов, устроены так, чтобы они как можно меньше влияли бы на код декорируемых сущностей. Разработчики могут создавать код для решения своих специфических задач так, как привыкли, а декораторы могут использовать исключительно для расширения функционала своих разработок.

Здесь обращаемся к декоратору `simple_tag` (рисунок 7)

```python
from django import template
from news.models import Category

register=template.Library()
@register.simple_tag()

def get_categories():
    return Category.objects.all()
```

Рисунок 7

В simple_tag () необходимо поставить скобки, которые мы потом заполним аргументами и нужны они для того, чтобы изменить название нашего тега. По умолчанию это будет `get_categories()`, но если нам нужно будет другое название, то мы можем в эту функцию simple_tag () передать другой параметр или аргумент (name) и там назвать наш тег.

Вернемся в наши шаблоны. Здесь нам необходимо оптимизировать код, потому что есть дублирование кода, который мы делаем как в index.html так и в category.html. Для того чтобы избежать дублирования кода, логично создать какой-нибудь подключаемый шаблон и в него вынести этот код.

В templatet в inc создадим новый шаблон _sidebar.html (рисунок 8).



Рисунок 8

И из шаблона category перенесем часть кода, который формирует список категорий и подключим к шаблону.

```
    <div                                class="list-group">
  {%        for        item        in        categories%}
  <a  href="{%  url  'category'  item.pk  %}"  class="list-
group-item list-group-item-action">{{ item.title }}</a>
```

Теперь воспользуемся созданным тегом и для этого необходимо его подключить к шаблону с помощью тега `include`

```
{% include 'inc/_sidebar.html' %}
```

Вставленого на место, вырезанной части кода в разделе sidebar (рисунок 9)



Рисунок 9

Тоже самой проделаем с шаблоном index (рисунок 10).

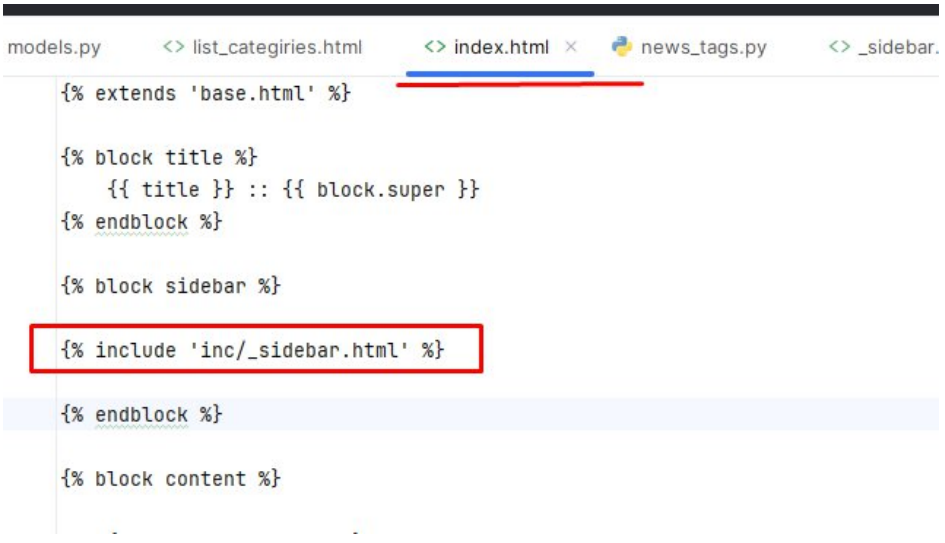


Рисунок 10

Теперь используем, созданный кастомный тег `get_categories()`.

Для этого необходимо его импортировать, т.е. подключить его к шаблону (загрузить) в шаблон.

И воспользуемся тегом load, который загружает настраиваемый набор тегов шаблона (подробнее смотрите в документации (рисунок 11). В нашем случае, мы импортируем из файла news_tags тег `get_categories()`

{% load news_tags %} (рисунок 12)

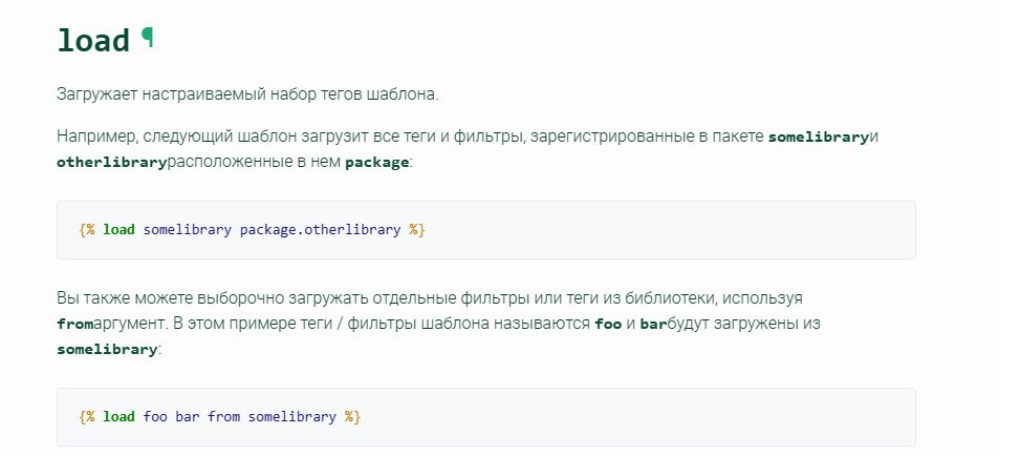


Рисунок 11



Рисунок 12

После импортирования, рекомендуется перезагрузить сервер, чтобы Djangoувидел новый тег.

Теперь нам необходимо передать названия категорий в цикл. Для этого мы можем использовать псевдоним и присвоить переменную Categories (рисунок 13). Более подробно можно посмотреть в документации (рисунок 14)

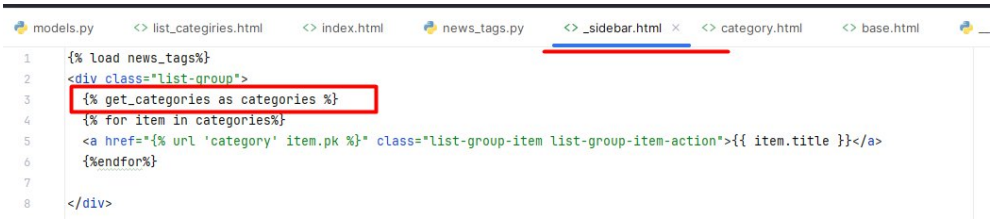


Рисунок 13

Результаты тега можно хранить в переменной шаблона, а не в непосредственно выводя его. Это делается с помощью аргумента, за которым следует имя переменной. Это позволит вам самостоятельно выводить контент там, где Вы считаете нужным:**as**

```
{% current_time "%Y-%m-%d %I:%M %p" as the_time %}
<p>The time is {{ the_time }}.</p>
```

Рисунок 14

Теперь из views мы можем удалить дублирующие строки кода (рисунок 15)

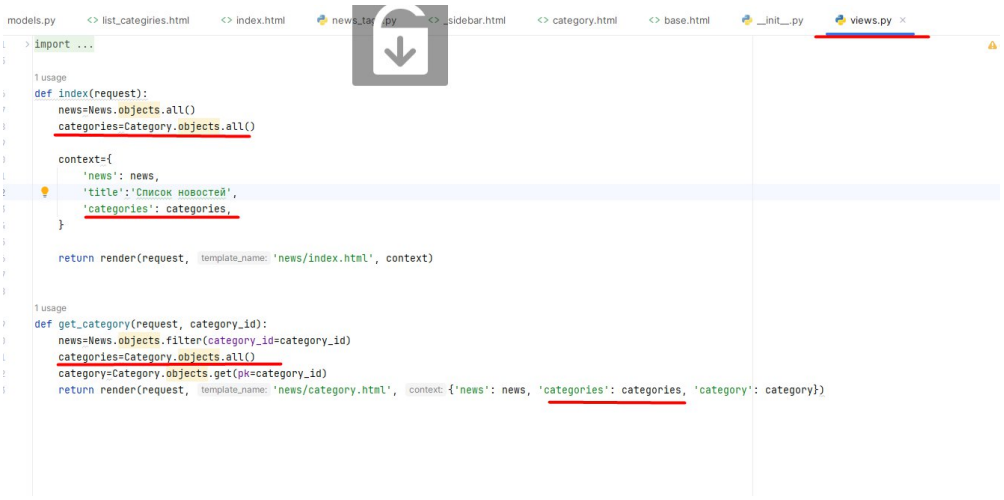


Рисунок 15

В результате имеем оптимизированный код (рисунок 16)

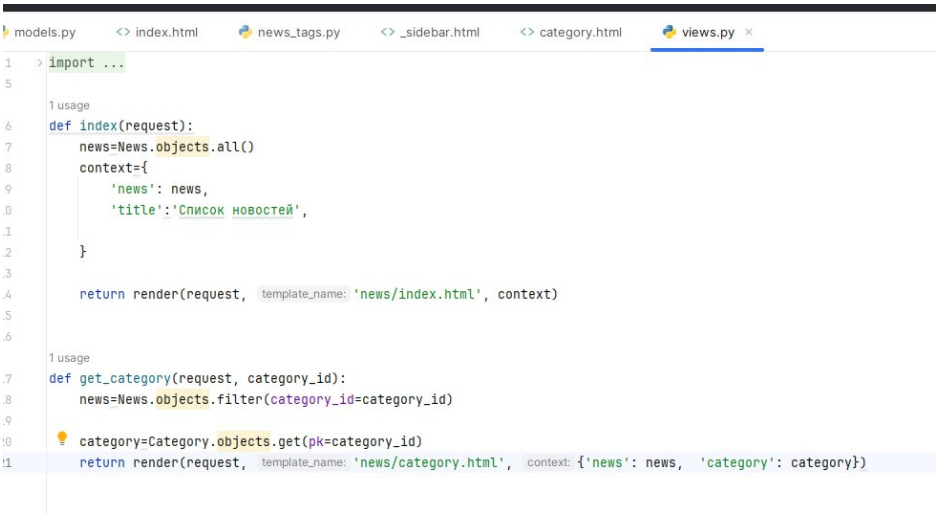


Рисунок 16

Таким образом, мы оптимизировали код. Он не дублируется, а определен в одном пользовательском теге (кастомном теге). В нашем случае мы получаем список категорий с помощью пользовательского тега 'get_categories'

Можем задать имя нашему списку категорий в строке регистрации пользовательского тега в пустых скобках (рисунок 17)

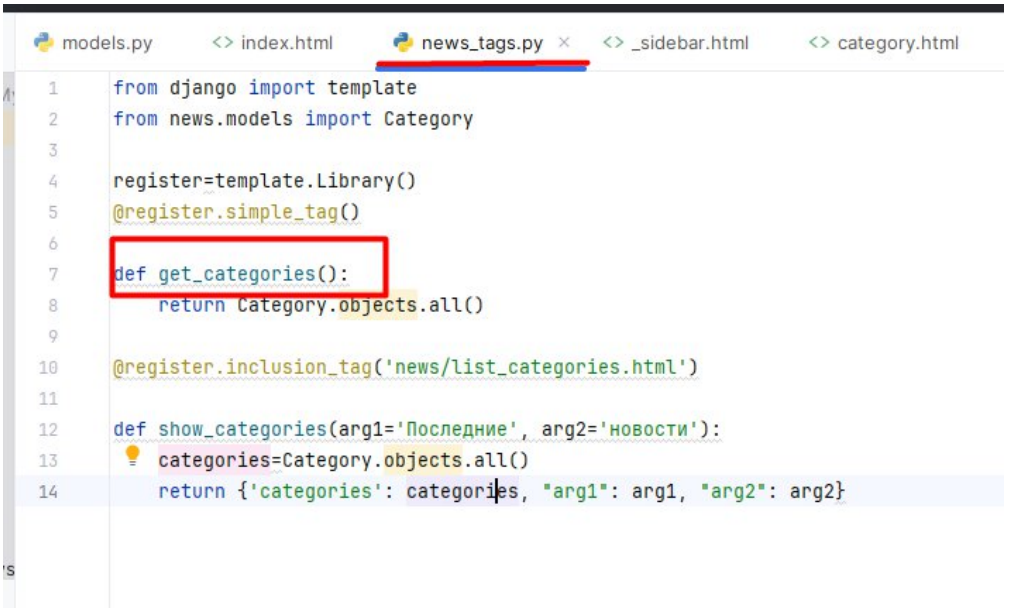


Рисунок 17

Добавив name='get_list_categories' мы это должны учесть в дальнейшем (рисунок 18)



Рисунок 18

Но теперь мы по это имя должны переписать в файле sidebar.html в пользовательском теге (рисунок 19).



Рисунок 19

Перейдем к созданию второго типа тега – это тег включения, который позволяет не только возвращать данные, но и рендерить данные в шаблон.

Для начала создадим необходимый шаблон в приложении news в папке templates/news /. Дадим имя шаблону list_categories.html (рисунок 20)

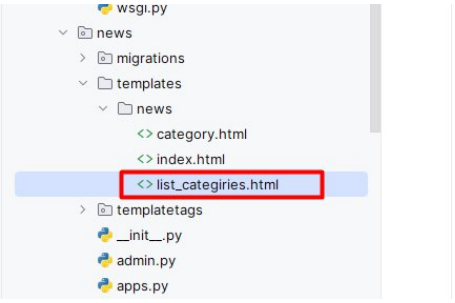


Рисунок 20

В этот файл мы передадим категории наших статей. Для упрощения, давайте скопируем содержимое файла sidebar.html и удалим некоторые строки, оставляя только цикл передачи категорий (рисунок 21), которые будет получать наш тег включения.

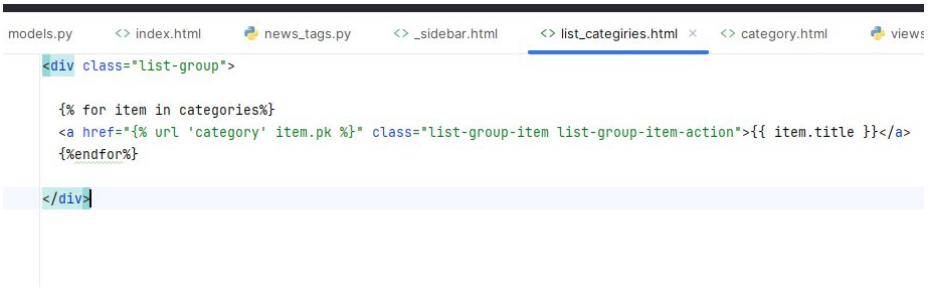


Рисунок 21

И в этот шаблон будут передаваться данные - список категорий и эти данные будет получать тег

Напишем его в файле news_tegs.py. Поскольку этот тег будет не только возвращать эти данные, но и показывать эти данные, то назовем его `show_categories.` Данный тег может принимать аргументы, которые определим чуть позже. Но пока получим переменные строкой `categories=Category.objects.all()` и вернем словарь с ключем `categories` и значением `categories` (рисунок 22)

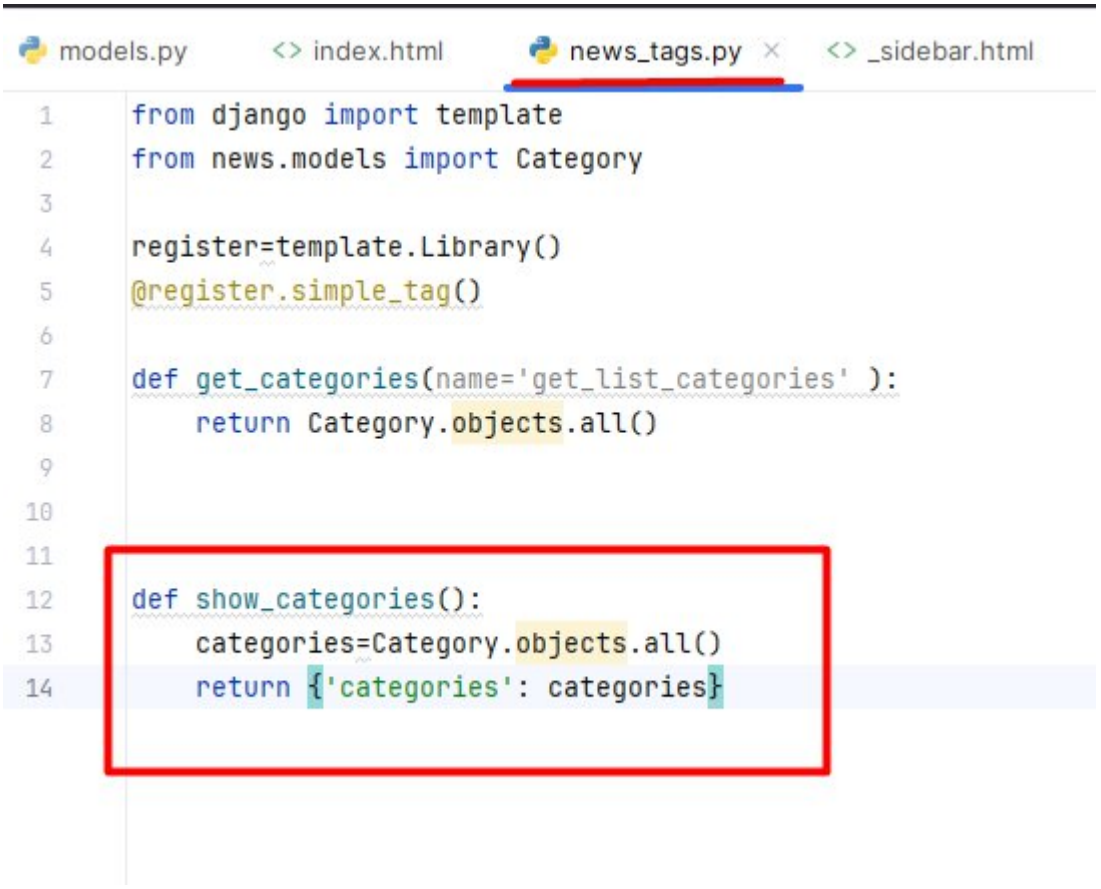


Рисунок 22

Сам шаблон, который необходим для рендеринга необходимо указать при регистрации тега.

При регистрации указываем inclusion_tag('news/list_categories.html') аргументом которого будет указан файл тега приложения/

Строка регистрации имеет вид:

news @register.inclusion_tag('news/list_categories.html')

Смотрите рисунок 23.



Рисунок 23

Теперь перейдем в _sidebare.html, в котором с помощью `{% load news_tags%}` подключили `news_tags`

И выведем здесь, полученные категории.

Теперь в шаблоне _sidebare.html вызовем этот тег (рисунок 24)



Рисунок 24

Можно передавать любые аргументы. Это необходимо, когда в нам необходимо вывести какие-то дополнительные данные. Например, в нашем

случае, это может быть вывод количества каких-нибудь последних 10-15 комментариев.

Для этого через тег включения show_categories(): давайте передадим каких-нибудь аргумента (рисунок 25)

```
def show_categories(arg1='Последние', arg2='новости'):
    categories=Category.objects.all()
    return {'categories': categories, "arg1": arg1, "arg2": arg2}
```

Рисунок 25

Как известно, эти данные доступны в шаблоне _list_categories. В этом файле добавим вывод этих аргументов. Результат вывода аргументов на рисунке 26

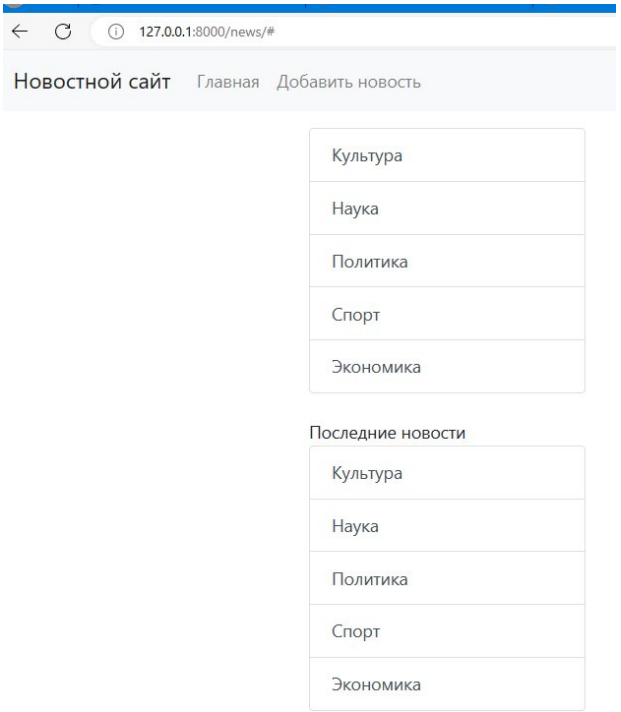


Рисунок 26

Лабораторная работа № 10.

Обратное разрешение адресов. Статические файлы

Цель работы: Изучить механизмы обратного разрешения адресов (reverse URL resolving) в Django, научиться использовать метод get_absolute_url для формирования ссылок на объекты моделей, а также освоить принципы работы со статическими файлами (CSS, JS) и их сборки на проекте.

1. Обратное разрешение адресов

Распространенной потребностью при работе над проектом Django является возможность получения URL-адресов в их окончательных формах либо для встраивания в сгенерированный контент (URL-адреса представлений и ресурсов, URL-адреса, показываемые пользователю, и т. д.), либо для обработки потока навигации на сервере, перенаправления и т. д.

Основная информация, доступная для получения URL-адреса, - это идентификация (например, имя) представления, отвечающего за его обработку.

Django предоставляет решение, в котором преобразователь URL-адресов является единственным хранилищем дизайна URL-адресов. Вы загружаете его своим URLconf, и затем его можно использовать в обоих направлениях:

- Начиная с URL-адреса, запрошенного пользователем / браузером, он вызывает правильное представление Django, предоставляя любые аргументы, которые могут ему понадобиться, с их значениями, извлеченными из URL-адреса.
- Начиная с идентификации соответствующего представления Django и значений аргументов, которые будут переданы ему, получите связанный URL.

Django предоставляет инструменты для реверсирования URL-адресов, соответствующие различным слоям, на которых требуются URL-адреса:

- В шаблонах: с помощью url тега шаблона.

- В коде Python: использование reverse()функции.

- В коде более высокого уровня, относящемся к обработке URL-адресов экземпляров модели Django: The get_absolute_url()method.

Продолжим изучать возможности Django и продолжим улучшать наш сайт.

На сайте мы с вами формируем различные ссылки. Ссылки на категории, ссылки на отдельную категорию и формируем их с помощью имен и url_адресов.

В файле urls приложения мы даем имя адресу, которое используем в шаблоне (рисунок 1).

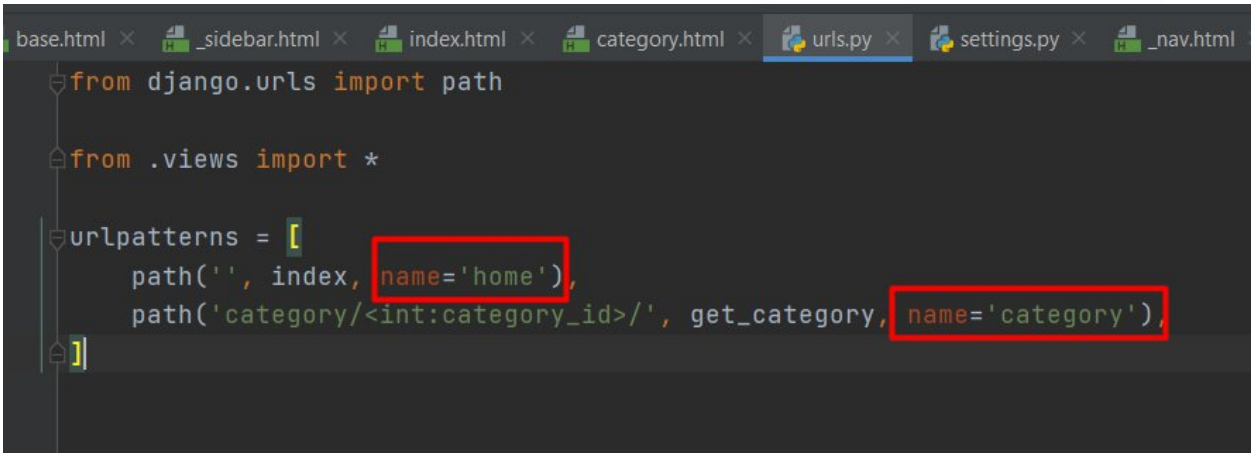


Рисунок 1

В параметре в шаблоне url мы передаем название нашего маршрута (рисунок 2).

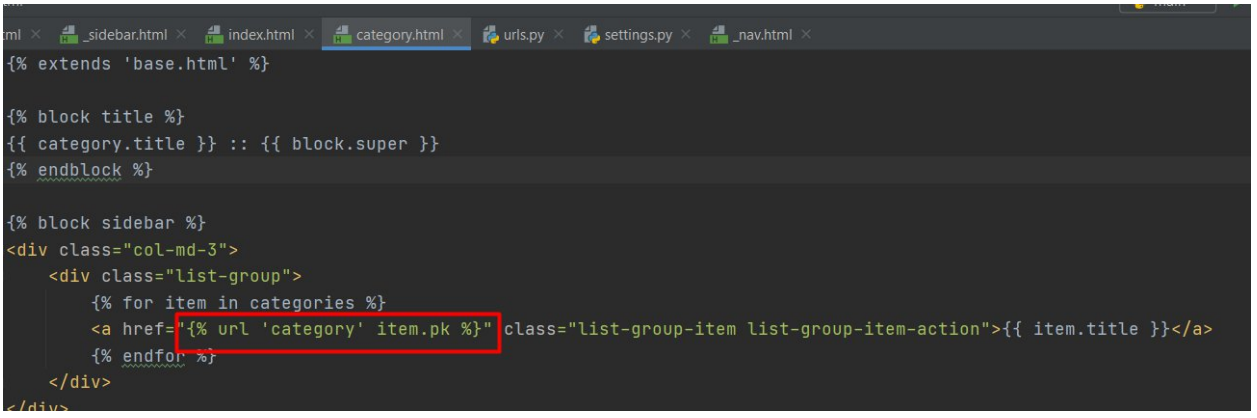


Рисунок 2

Но это не единственный вариант, который рекомендует Django для построения ссылок на отдельные экземпляры модели. В нашем случае на отдельную Категорию и отдельную Новость.

Но для начала закомментируем часть кода, который дублирует вывод списка категорий.

Это например, можно сделать с помощью тега

{%comment %}

{% endcomment %}

Для этого необходимо написать специальный метод с помощью которого Django сам выстроит ссылку, который будет указывать на конкретный объект модели, т.е. на конкретную категорию и построит нужный вариант ссылки (рисунок 3).

Для этого откроем файл модели models приложения и для класса Category напишем этот метод.

Дать название методу можно любой, но если следовать конвенции Django, то зададим этому методу имя **get_absolute_url.**

Если мы назовем этот метод именно так, то Django этот метод сможет использовать по умолчанию. Например, он его использует при создании формы объекта модели, например, будет автоматом переходить на созданный объект благодаря этому методу get_absolute_url

Рассмотрим работу данного метода в админке. Когда мы создадим данный метод, то в админке он будет использован для того чтобы перейди из админки на страницу конкретной категории.

Для написания метода, нам потребуется специальная функция, которая называется **revers**. Эта функция из модуля Django urls. Импортируем reverse (рисунок 3)

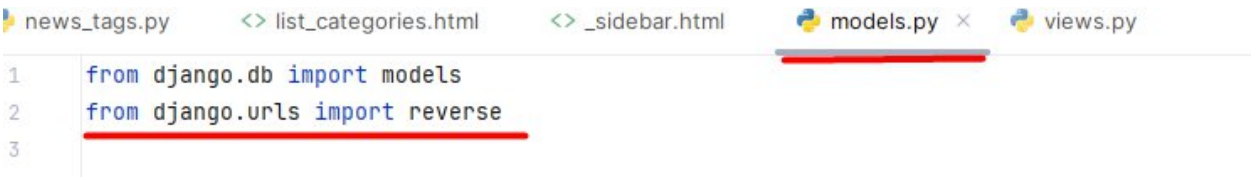


Рисунок 3

Записываем метод и возвращаем функцию reverse.

В этой функции первый параметр принимает название url маршрута, а вторым параметром будем использовать kwargs, который будет из себя представлять словарь kwargs={'category_id': self.pk} и он является, этот второй параметр, параметром для построения данного маршрута (рисунок 4).



Рисунок 4

Используя данный метод get_absolute_url, Django сам выстроит ссылку.

Если мы обратимся к list_categories.html и сравним работу функции reverse и работу тега url в этом шаблоне, то увидим, что они очень похожи (рисунок 5).

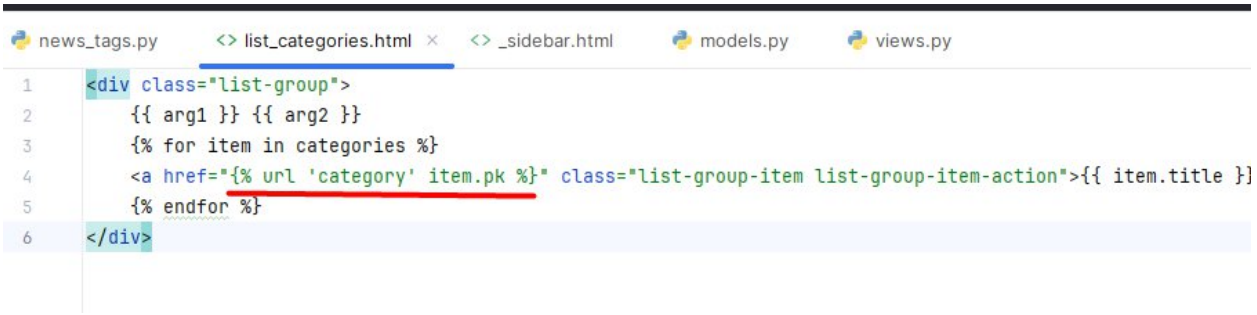


Рисунок 5

Отсюда мы можем понять, что функция reverse и тег url – это одна и та же функция по сути, только url – это template teg, который предназначен для работы в шаблонах, а функция reverse – это таже функция, но только в python файлах и делает абсолютно тоже самое, т.е. строит ссылку, но только в python файлах.

Воспользуемся данным методом. У нас есть.

item – это конкретный экземпляр модели categories. Поэтому берем этот самый item и вызываем его метод

Таким образом, мы обратились к модели, вызвали его метод get_absolute_url, который как мы поняли, возвращает нам ссылку на конкретный объект. Эту ссылку он берут из наименования маршрута и берет каждый конкретный идентификатор категории, которые крутятся в цикле.

Удаляем не нужную нам ссылку url (рисунок 6, 7)

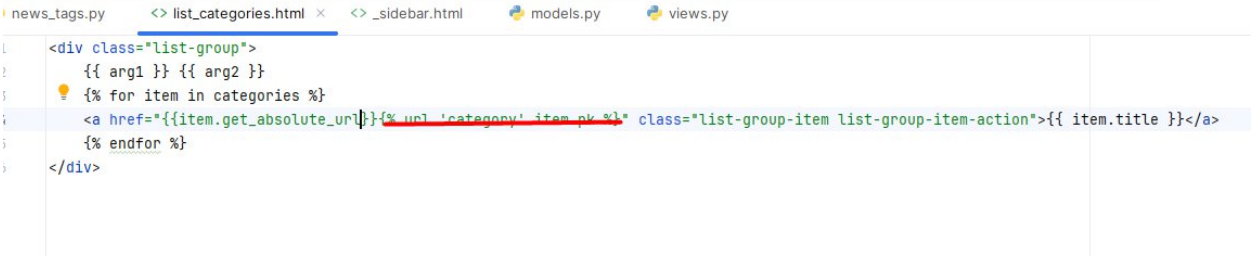


Рисунок 6

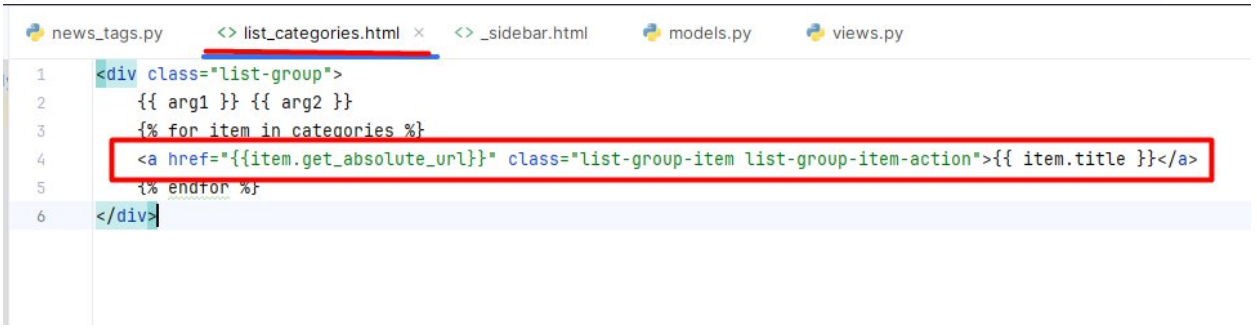


Рисунок 7

Тоже можно проделываем в шаблонах index.html (рисунок 8).



Рисунок 8

Этот метод get_absolute_url позволяет добавить кнопку предпросмотра страницы сайта в админке, которой не было до использования этого метода (рисунок 9) и просматривать ту страницу с той категорией, которая в настоящий момент редактируется.

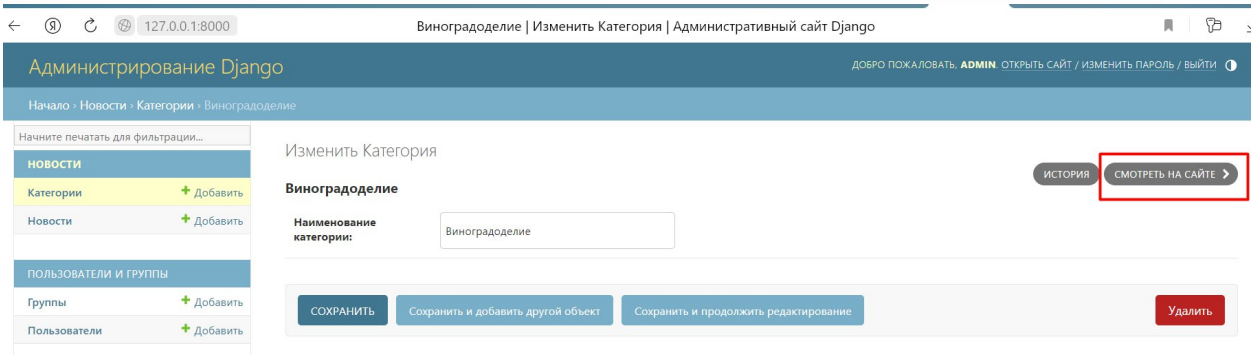


Рисунок 9

Если мы закомментируем этот метод, то после обновления, увидим, что этой кнопки нет.

Поэтому при создании этого метода и рекомендуется давать названия в соответствии с конвенцией Django.

Реализация кнопки Read more…

Перейдем к реализации кнопки просмотра отдельной записи Read more…

Для этого в первую очередь пропишем маршрут (рисунок 10).



Рисунок 10

По этой ссылке мы должны видеть конкретную новость. И здесь мы должны прописать новую функцию, которую создадим в файле views, т.е. в файле нашего контроллера.

И создадим функцию views_news для просмотра отдельной новости. Первым параметром она будет принимать request, а вторым параметром принимает news_id.

def view_news(request, news_id):

Теперь по идентификатору, запрошенную новость. Для этого создаем переменную, в которую будем получать эту новость через обращение к модели News

news_item = News.objects.get(pk=news_id)

и далее рендерим:

return render(request, 'news/view_news.html', {"news_item": news_item})

В итоге, получаем новый контроллер (рисунок 11):



Рисунок 11

Следующим шагом создадим шаблон для вывода отдельной страницы просмотра статьи категории view_news.html в папке templates в нашем приложении news.

Скопируем в него шаблон из файла index.html и внесем изменения.



Рисунок 12

Таким образом, будет выводиться название открываемой новости в заголовке.

Поскольку будет только один объект, то цикл нам не нужен, поэтому строки с тегом цикла удаляем.

Везде item заменим на news_item.

и удалим строку

Read more...

Полный текст следующий:

```
{% extends 'base.html' %}

{% block title %}
    {{ news_item.title }} :: {{ block.super }}
{% endblock %}

{% block sidebar %}
{% include 'inc/_sidebar.html' %}
{% endblock %}
```

```
{% block content %}
<div class="col-md-9">

    <div class="card mb-3">
        <div class="card-header">
            Категория: <a
href="{{news_item.category.get_absolute_url}}">{{ news_item.category }}</a>
        </div>
        <div class="card-body">
            <div class="media">
                {% if news_item.photo %}
                <img src="{{ item.photo.url }}" alt="" width="350" class="mr-
3">
                {% else %}
                <img src="https://picsum.photos/id/1060/350/235/?blur=2"
alt="" class="mr-3">
                {% endif %}
                <div class="media-body">
                    <h5 class="card-title">{{ news_item.title }}</h5>
                    <p class="card-
text">{{ news_item.content|safe|linebreaks}}</p>
                                    </div>
            </div>

        </div>
        <div class="card-footer text-muted">
            {{ news_item.created_at|date:"Y-m-d H:i:s" }}
        </div>
    </div>

</div>
{% endblock %}
```

Теперь необходимо прописать ссылки в шаблонах index.html и category.html

Перейдем в файл модели и создадим функцию подобную get_absolute_url, но для модели News.

```
def get_absolute_url(self):
    return reverse('view_news', kwargs={'news_id': self.pk})
```

И пропишем эту ссылку в шаблонах index.html и category.html.



Рисунок 13

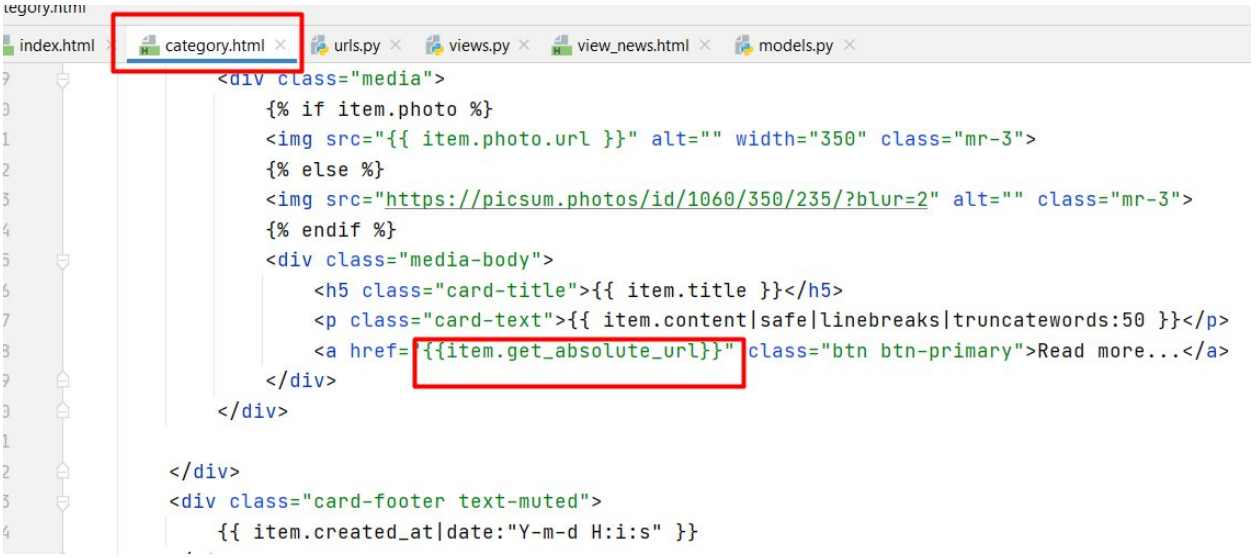


Рисунок 14

Рассмотрим с вами следующую возможность Django. Допустим мы вами перешли на страницу статьи категории из какой-нибудь закладки, сохраненной раннее или открываемой вами из журнала браузера. Но при этом статья администратором была удалена. И ссылка у нас оказалась «битой».

Например, давайте перейдем к статье по номеру не существующей статьи и в результате видим ошибку о том, что такой страницы нет (рисунок 15).

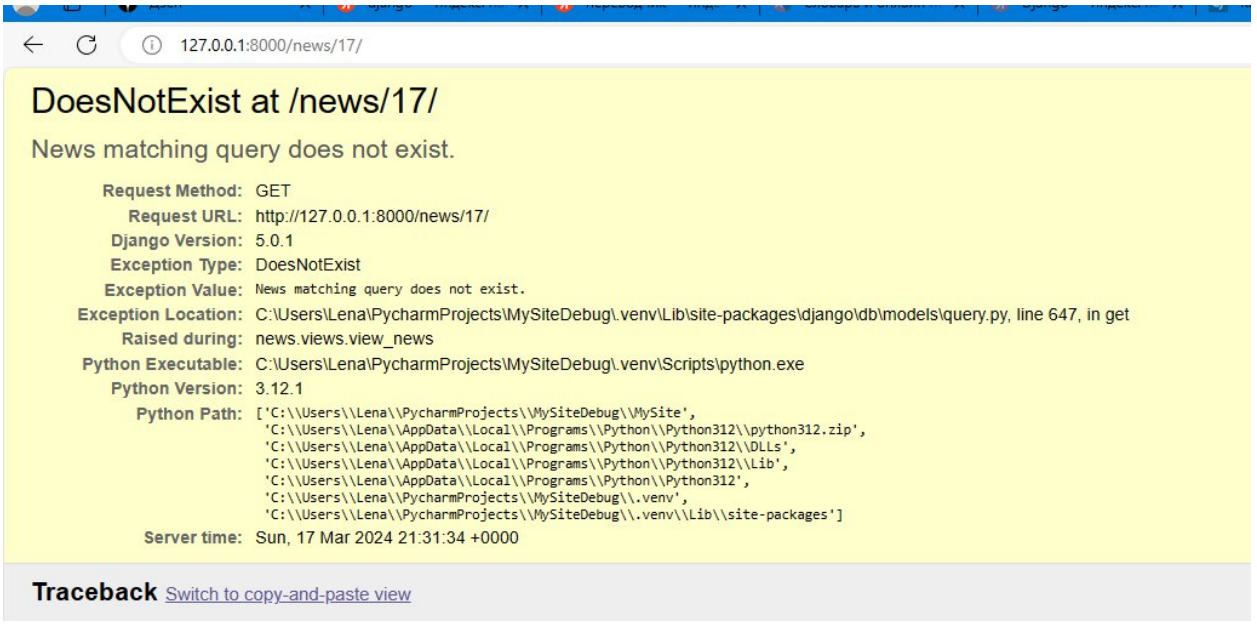


Рисунок 15.

Обратимся к методу **'view_news**

Как видим из строки return reverse(**'view_news'**, kwargs={**'news_id'**: self.pk}), то здесь ожидается один вполне конкретный объект.

Если вдруг случается, что он ни чего не достанет, либо достанет, более чем один объект, то он выбрасывает исключение, которое мы с вами видим (рисунок 15).

Если мы посмотрим код в браузере (рисунок 16)

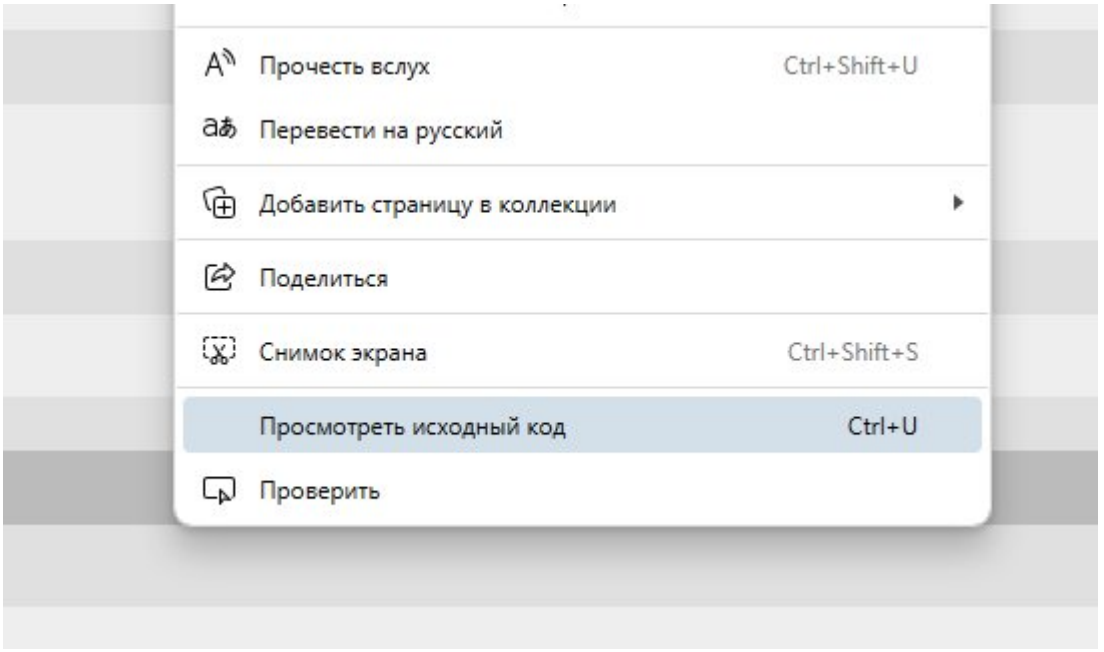


Рисунок 16

Консоль разработчика открывается «Ctrl + Shift + I».

Перейдем во вкладку разработчика на вкладку Сеть и после обновления увидим ошибку 500, но не 404 о которой можно было бы предположить (рисунок 17).

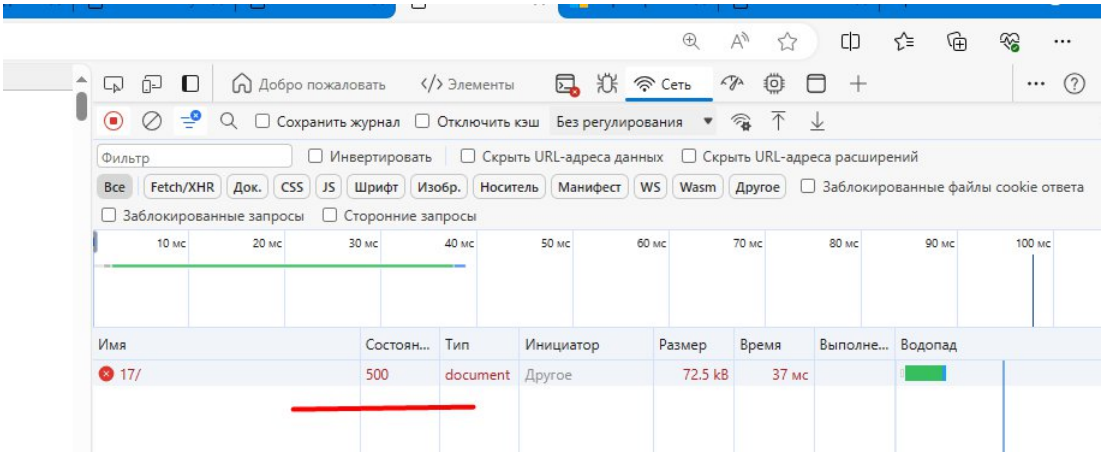


Рисунок 17

500 – это ошибка сервера.

Как можно избежать этой ситуации? Для этого используем метод Django get_object_or_404 и импортируем его в файл контроллеров (рисунок 18)

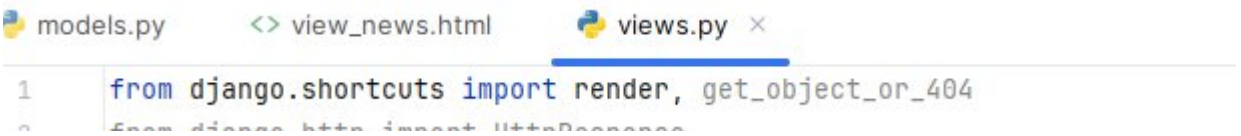


Рисунок 18

Используем эту функцию (рисунок 19)



Рисунок 19

В результате получим уже 404 ошибку (рисунок 20).

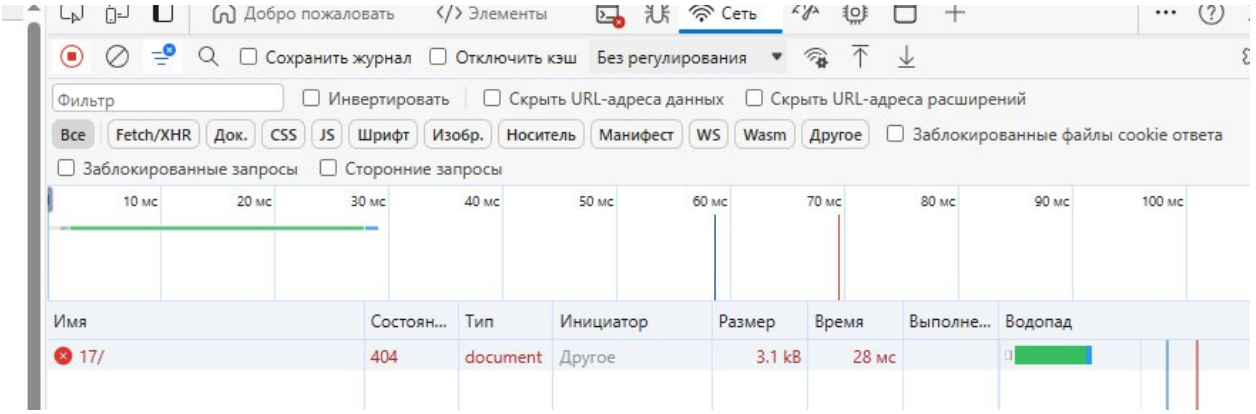


Рисунок 20

И получим ответную страницу с соответствующей ошибкой

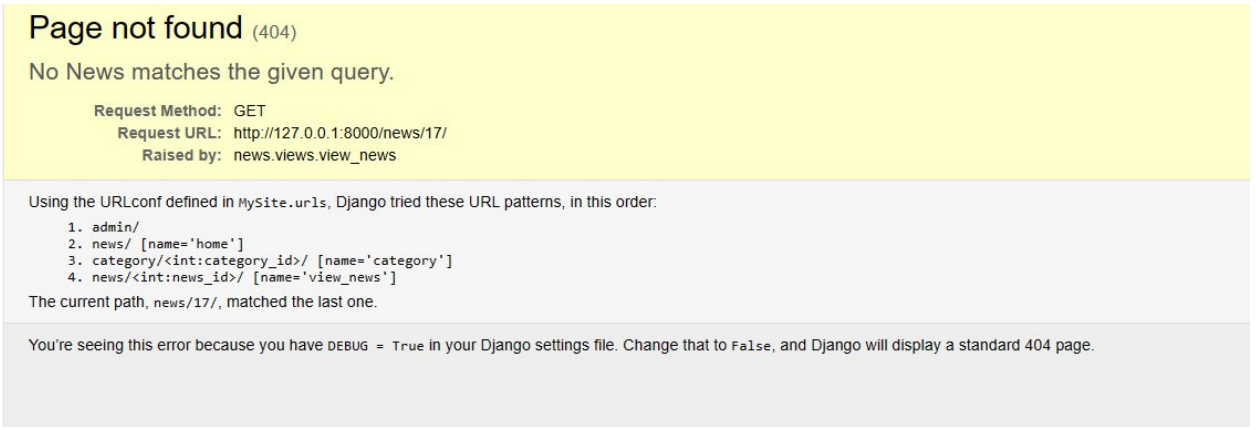


Рисунок 20

2. Статические файлы

Перейдем к работе со статикой. Статика в терминологии Django – это любой статический файл типа CSS, JS либо медиа файлы, все то, что включается в шаблоны для его красивого оформления.

В данном проекте мы имеем базовый шаблон base.html, то здесь мы подключаем bootstrap, jquery, popper. Это хорошая практика подключения. Но многие файлы могут подключаться локально с вашего сервера. Разберем как это сделать.

Django по умолчанию ищет всю статику в папке static приложения. Это происходит благодаря настройки (рисунок 21)



Рисунок 21

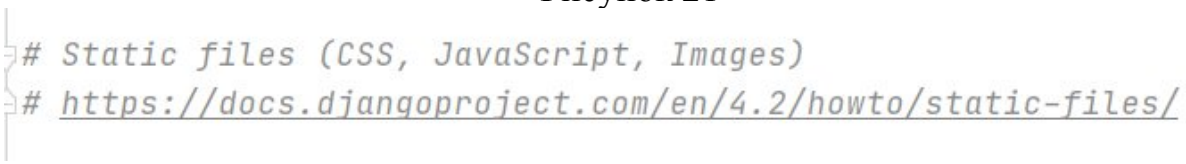


Рисунок 22 – Ссылка как работать со статикой

Благодаря этой настройки Django ищет эту папку со статикой этого приложения. Но на практике у нас может быть более чем одно приложение. В каждом приложении будет своя статика. Но будет соответственно общая статика. Для этого необходимо создать общую папку, которую создадим в корне проекта. Мы знаем, что у нас есть отладочный сервер и есть боевой сервер, где должен работать наш сайт.

На отладочном сервере Django ищет папки со статикой и показывает ее, т.е. умеет обрабатывать папки со статикой приложения. Сторонний сервер этого делать не умеет. Для него необходимо, чтобы статика находилась в одной папке, например, в корне нашего приложения. Для этого в корне приложения создается папка, которая создается самим Django.

Подключим https://getbootstrap.com/ локально.

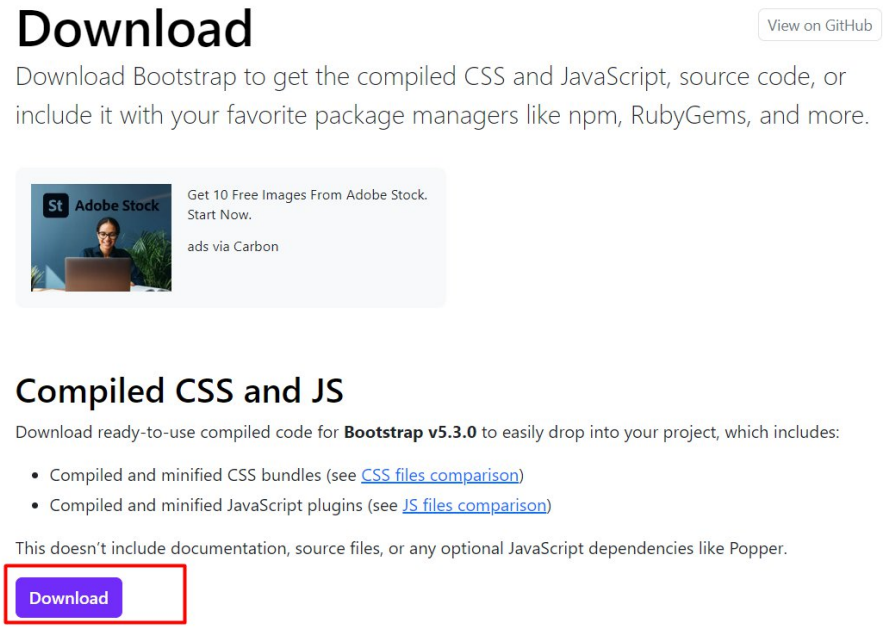


Рисунок 23

В конфигурационной папке создадим папку static (рисунок 24)

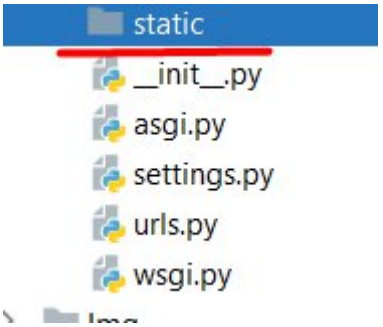


Рисунок 24

И в эту папку распакуем bootstrap. И создадим внутри папки static папку css и внутри нее файл style.css (рисунок 25)

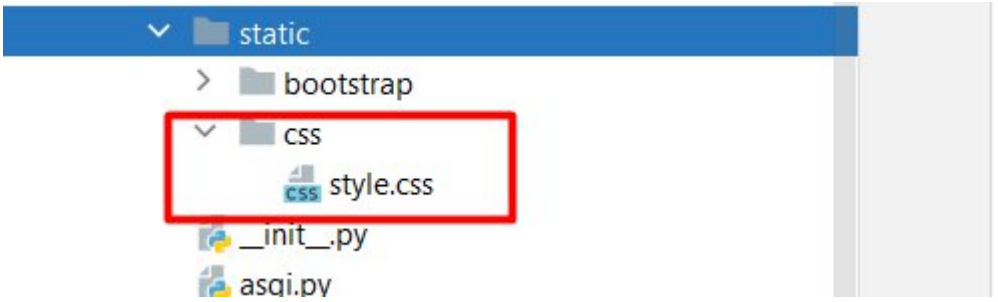


Рисунок 25

И создадим некоторый стиль и запустим его вывод (рисунок 26).

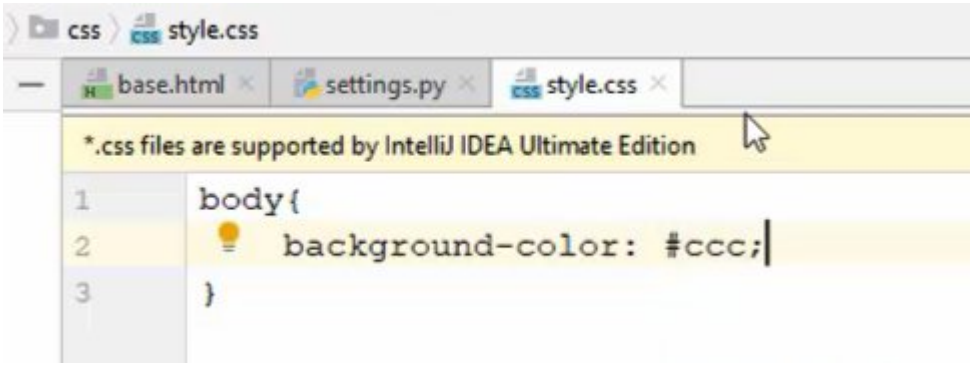


Рисунок 26

Чтобы эта статика заработала, нам необходимо использовать три константы. Уже используемая константа STATIC_URL.

Эта константа добавляет префис к адресу статического файла. Встретив этот адрес Django будет понимать, что ему нужно подключить статику и где ее искать.

И дополнительно создаем еще две константы:

- STATIC_ROOT – хранит путь к папке static. Суть этой константы собрать все статические файлы с одном месте;
- STATIFILES_DIRS – хранит дополнительные пути к папкам со статикой. Т.е. когда мы выполним команду, которая соберет всю статику в единое место, это делается на боевом сервере, то Django пройдется по всем папкам приложения, где есть статика и закинет по пути os.path.join(DASE_DIR…). Создаст папку статика в корне приложения и кроме того он возьмет статику еще из третьей константы (рисунок 27).

```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/4.2/howto/static-files/

STATIC_URL = '/static/'
STATIC_ROOT=os.path.join(BASE_DIR, 'static')
STATIFILES_DIRS=[
        os.path.join(BASE_DIR, 'MySite/static')
]
```

Рисунок 27

Для сбора всей статики в терминале выполним следующую команду

python manage.py collectstatic

Джанго найдет всю статику и скопирует ее по указанному адресу.

```
(venv) C:\Users\Елена\PycharmProjects\MyProject\MySite>python manage.py
collectstatic

125 static files copied to 'C:\Users\Елена\PycharmProjects\MyProject\ MySite
\static'.
```

125 файлов появилось.

В папке проекта появилась папка static (рисунок 28)

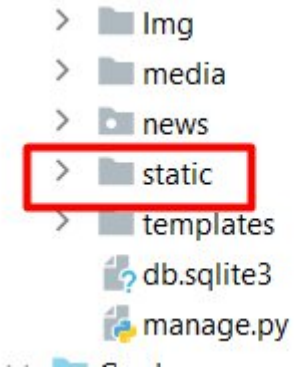


Рисунок 28

В корне проекта появилась папка static, содержащая папку admin (рисунок 29)

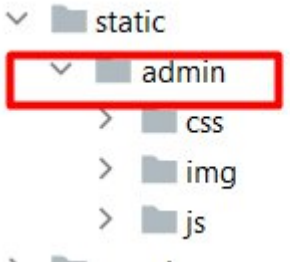


Рисунок 29

В этой папке хранится вся статика, которая находится в админке.

А в конфигурационной папке проекта тоже появилась папка static, содержащая статику (рисунок 30).

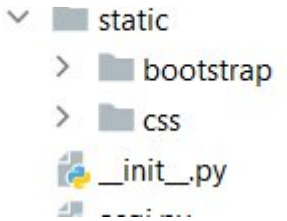


Рисунок 30

Статика у нас теперь есть, и теперь необходимо подключать наши файлы расположенные локально. Это делается в base.html (рисунок 31).

Удаляем ссылку (рисунок 31)

```
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js"
        integrity="sha384-
wfSDF2E50Y2D1uUdj0O3uMBJnjuUD4Ih7YwaYd1iqfktj0Uod8GCExl3Og8ifwB6"
        crossorigin="anonymous"></script>
```

```
<script src="https://code.jquery.com/jquery-3.4.1.slim.min.js"
        integrity="sha384-J6qa4849blE2+poT4WnyKhv5vZF5SrPo0iEjwBvKU7imGFAV0wwj1yYfoRSJoZ+n"
        crossorigin="anonymous"></script>
<script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"
        integrity="sha384-Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo"
        crossorigin="anonymous"></script>
```

Рисунок 31

В заголовке файла прописываем ссылки (рисунок 32, 33)

```
ttings.py    <> base.html ×
<!doctype html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <link rel="stylesheet" href="{% static 'bootstrap/css/bootstrap.min.css' %}">
    <link rel="stylesheet" href="{% static 'css/style.css' %}">
    <title>{% block title %}Новости со всего мира{% endblock %}</title>
</head>
<body>

{% include 'inc/_nav.html' %}
```

Рисунок 32

Второй строкой прописала доступ к нашему файлу css

И используем специальный тег, который называется static, который подключаем вначале файла (рисунок 33)

{% load static %}

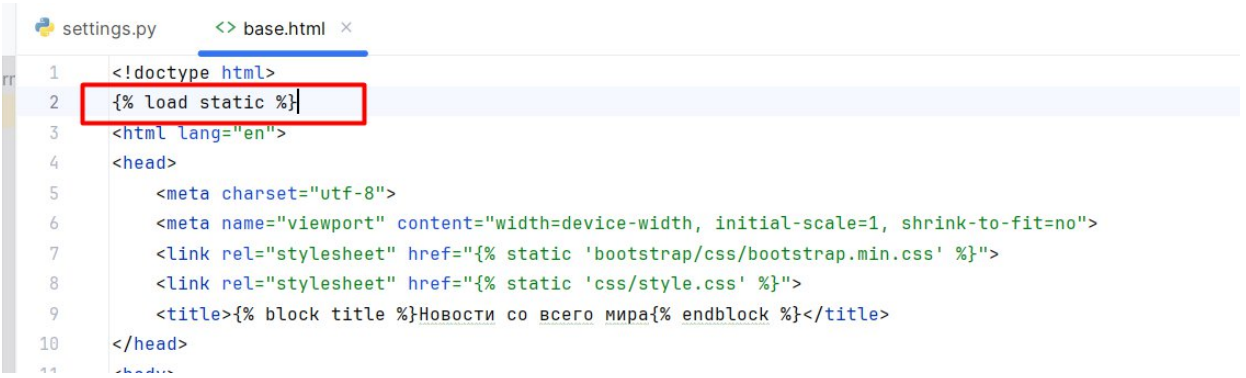


Рисунок 33

Подключим bootstrap внизу страницы (рисунок 34)

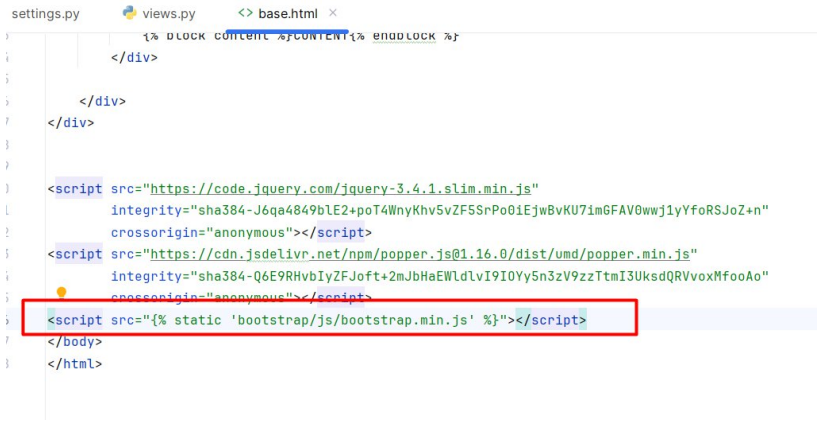


Рисунок 34

Если файл изменился, то для его применения, необходимо его записать и повторно выполнить команду python manage.py collectstatic.

Практические задания

Задание 1. Реализация get_absolute_url для модели

1. В существующем приложении откройте файл models.py.

2. Для модели Category (или аналогичной) реализуйте метод get_absolute_url, который будет возвращать ссылку на страницу категории, используя reverse и идентификатор записи.

3. В шаблоне list_categories.html замените явное использование тега {% url %} на вызов метода get_absolute_url через объект категории.

4. Проверьте, что ссылки работают корректно.

Задание 2. Детальная страница новости

1. Создайте маршрут (URL) для просмотра отдельной новости, который будет принимать news_id.

2. В файле views.py создайте функцию view_news, которая с помощью get_object_or_404 получает объект новости или возвращает 404 ошибку, если новость не найдена.

3. Создайте шаблон view_news.html для отображения заголовка, текста, даты и категории новости.

4. В модели News реализуйте метод get_absolute_url, возвращающий ссылку на эту детальную страницу.

5. В шаблонах index.html и category.html замените кнопку "Read more..." на ссылку, сформированную через метод get_absolute_url.

Задание 3. Работа со статическими файлами

1. Создайте в корне проекта папку static, а в ней - папку css и файл style.css, если еще не создана.

2. Пропишите в файле style.css любой стиль (например, изменить цвет фона или шрифта).

3. В настройках settings.py укажите пути до папок со статикой (STATICFILES_DIRS и STATIC_ROOT).

4. В базовом шаблоне base.html подключите тег {% load static %} и добавьте ссылку на созданный CSS-файл.

5. Выполните команду сбора статики и убедитесь, что все файлы скопировались в папку STATIC_ROOT.

Задание 4. Локальное подключение Bootstrap

1. Скачайте дистрибутив Bootstrap с официального сайта.

2. Распакуйте файлы в папку static вашего проекта.

3. В шаблоне base.html замените CDN-ссылку на Bootstrap локальным путём через тег {% static %}.

4. Перезапустите сервер и проверьте, что стили Bootstrap применяются корректно.

Задание 5. Динамический breadcrumb (навигационная цепочка)

1. Создайте шаблонный тег или добавьте логику в представления для отображения "хлебных крошек" (breadcrumbs) на странице новости.

2. На странице view_news.html добавьте навигацию вида: "Главная > Категория > Название новости", где каждый элемент — активная ссылка, сформированная через get_absolute_url.

3. Если категория или новость не существуют, ссылка не должна отображаться или должна вести на 404.

Задание 6. Кастомная страница 404

1. Создайте собственный шаблон 404.html в корне папки templates.

2. Настройте Django для использования этого шаблона при возникновении ошибки 404.

3. Добавьте в шаблон ссылку на главную страницу, сформированную с помощью reverse (например, через контекстный процессор или напрямую в шаблоне с помощью {% url %}).

4. Проверьте работу, перейдя по несуществующему адресу.

Задание 7. Рефакторинг ссылок в меню категорий

1. В файле inc/_sidebar.html (или аналогичном шаблоне сайдбара) используется цикл для вывода списка категорий.

2. Перепишите код так, чтобы ссылка на каждую категорию формировалась не через {% url %}, а через вызов метода get_absolute_url у объекта категории.

3. Убедитесь, что после рефакторинга ссылки работают так же, как и раньше.

Задание 8. Использование reverse в представлениях для редиректа

1. Создайте простое представление, которое после выполнения какого-либо действия (например, имитация отправки формы) перенаправляет пользователя на страницу конкретной категории.

2. Используйте функцию redirect из django.shortcuts в сочетании с reverse для формирования URL.

3. Пример: return redirect(reverse('category_view', kwargs={'category_id': some_id}))

4. Проверьте, что редирект работает корректно.

Задание 9. Работа с медиафайлами (изображения)

1. Настройте в settings.py константы MEDIA_URL и MEDIA_ROOT для загрузки пользовательских изображений.

2. Добавьте в модель News поле image (или используйте существующее поле photo), которое будет загружаться через админку.

3. В шаблоне view_news.html настройте отображение изображения через {{ news_item.photo.url }}, предусмотрите заглушку, если изображения нет.

4. Убедитесь, что на этапе разработки медиафайлы обслуживаются корректно (пропишите + static() в urls.py проекта).

Задание 10. Автоматическое обновление статики при разработке

1. Установите и настройте пакет django-browser-reload (из django-extensions или аналогичный) для автоматической перезагрузки страницы при изменении статических файлов.

2. Или настройте режим отладки так, чтобы изменения в CSS (например, в style.css) применялись без ручного перезапуска сервера и выполнения collectstatic.

3. Внесите несколько изменений в локальный CSS-файл и убедитесь, что они сразу видны на сайте без дополнительных действий.

Контрольные вопросы

1. Что такое обратное разрешение адресов (reverse URL resolving) и для каких целей оно используется в Django?

2. Чем отличается использование тега {% url %} в шаблоне от функции reverse() в Python?

3. Для чего используется метод get_absolute_url() в моделях Django? Какое преимущество даёт соблюдение конвенции именования этого метода?

4. Как метод get_absolute_url() влияет на интерфейс административной панели Django?

5. Какую функцию выполняет reverse() и какие параметры она принимает?

6. Какое исключение возникает, если запрашиваемый объект не найден в базе данных? Как обработать эту ситуацию с помощью get_object_or_404()?

7. В чём разница между ошибками 404 и 500, и как Django позволяет управлять отображением этих ошибок?

8. Что такое статические файлы в терминологии Django?

9. Объясните назначение следующих настроек в settings.py:

- STATIC_URL
- STATICFILES_DIRS
- STATIC_ROOT

10. Какую команду нужно выполнить для сбора всей статики в единую директорию и зачем это нужно на боевом сервере?

Лабораторная работа №11.

Работа с формами

Цель работы: научиться создавать формы

Теоретическое обоснование

1. Обзор. Создание формы, не связанной с моделью

В процессе выполнения лабораторной формы мы познакомимся с основами работы с формами в Django. Для получения более полной информации по теме следует обратиться к документации: https://django.fun/ru/docs/django/4.1/#forms

В Django существует 2 вида форм:

- связанные с моделью
- несвязанные с моделью.

В процессе работы мы познакомимся со всеми вариантами, однако стоит отметить, что в практике чаще необходимо использовать формы, связанные с моделью.

При использовании связанной с моделью формы необходимо меньший объем кода. В противном случае программисту потребуется создать саму

форму, указать для нее поля и их тип, провалидировать, принять введенные данные и сохранить их.

Естественно, библиотека forms поставляется с набором классов **Field**, которые представляют общие потребности валидации. Для каждого поля мы описываем виджет по умолчанию, используемый, если вы не указали widget. Подробнее про каждый виджет:

https://django.fun/ru/docs/django/4.1/ref/forms/fields/

Итак, начнем знакомство с несвязанной с моделью формы. В нашем случае это будет страница с добавлением новой новости.

Первым шагом, добавим в файл news/url.py новый маршрут. (рисунок 1)



Рисунок 1

Затем в файле views.py создадим саму функцию add_news (на предыдущем этапе название функции подчеркивалось красным цветом, так как ее пока не существует). (рисунок 2)



Рисунок 2

В функции render вторым параметром мы передали название шаблона, теперь нам необходимо его создать. Добавляем в news/templates/news html-файл с названием add_news. (рисунок 3)

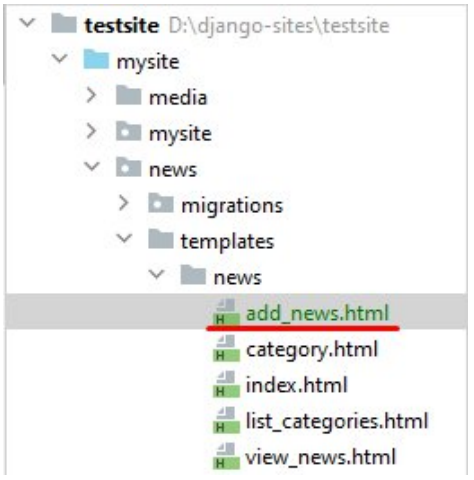


Рисунок 3

Для экономии времени скопируйте в созданный файл содержимое view_news.html, затем измените title на «Добавление новости» и уберите содержимое блока контента. После добавьте заголовок страницы аналогичный title. (рисунок 4)

Теперь, чтобы появилась возможность просмотра создаваемой нами формы, добавим в панели навигации ссылку на нее (_nav.html). Для этого заменим {% url ‘home’%} на {% url ‘add_news’ %}, где «add_news» - название маршрута. (рисунок 5)

В итоге, на данном этапе мы получим следующий результат. (рисунок 6, рисунок 7)

urls.py | views.py | add_news.html

```
{% extends 'base.html' %}

{% block title %}
Добавление новости :: {{block.super}}
{% endblock %}

{% block sidebar %}
{% include 'inc/_sidebar.html' %}
{% endblock %}

{% block content %}

<h1>Добавление новости</h1>

{% endblock %}
```

Рисунок 4

urls.py views.py add_news.html _nav.html

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <a class="navbar-brand" href="{% url 'home' %}">Navbar</a>
    <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupportedContent"
            aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
    </button>

    <div class="collapse navbar-collapse" id="navbarSupportedContent">
        <ul class="navbar-nav mr-auto">
            <li class="nav-item"><a class="nav-link" href="{% url 'home' %}">Главная</a></li>
            <li class="nav-item"><a class="nav-link" href="{% url 'home add_news' %}">Добавить новость</a></li>
        </ul>
    </div>
</nav>
```

Рисунок 5

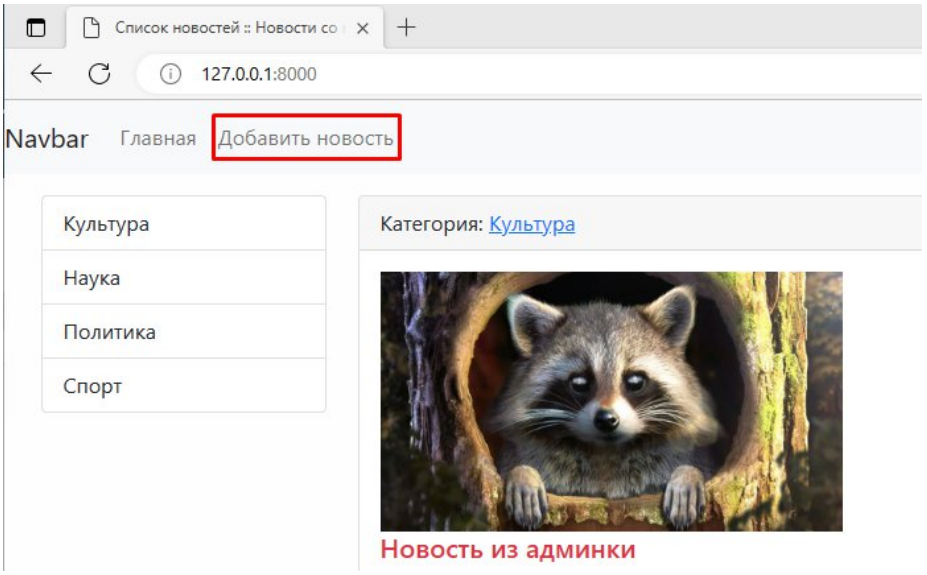


Рисунок 6

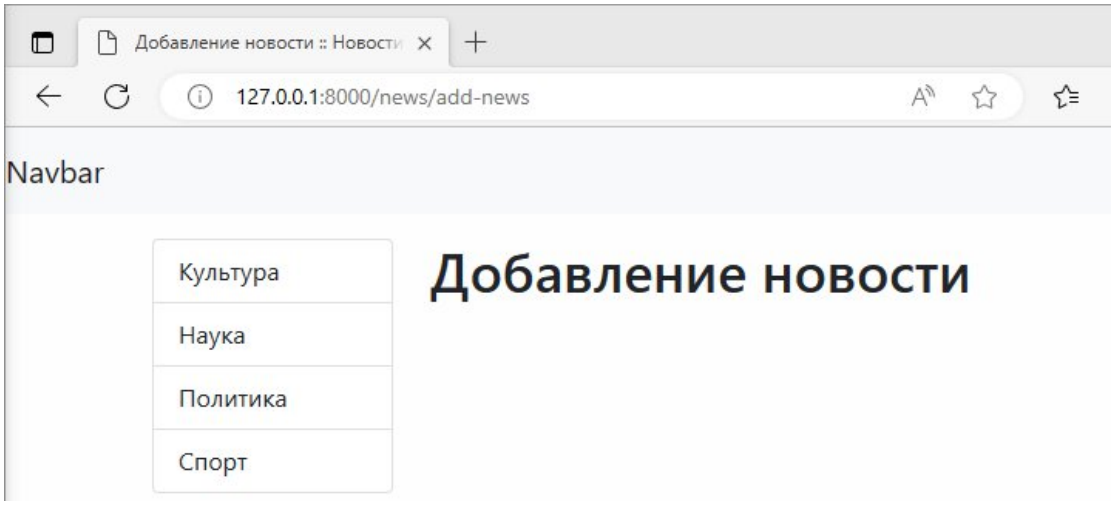
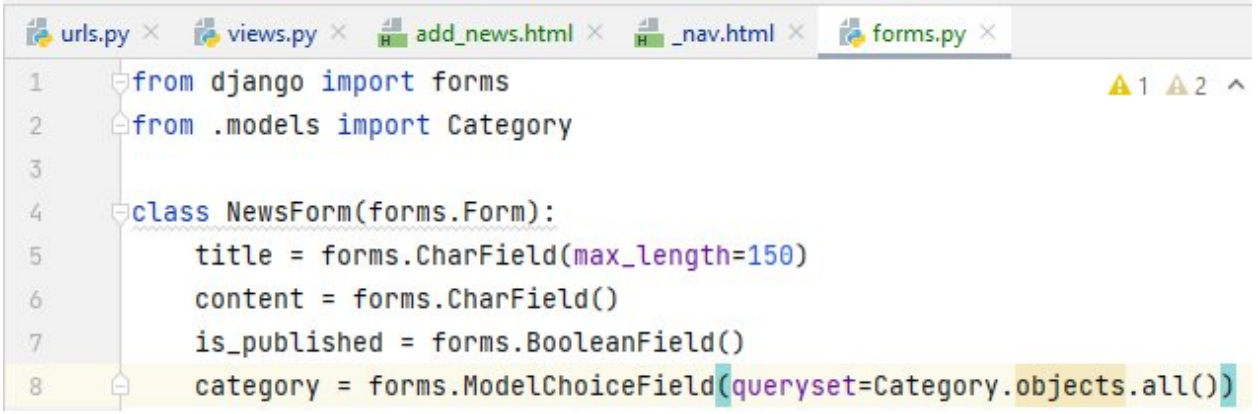


Рисунок 7

Теперь приступим непосредственно к созданию формы. Для начала создадим в папке приложения news pyhton-файл с названием forms. В нем импортируем модуль для работы с формами и модель Category (она нужна, чтобы была возможность выбора категории при добавлении новости). Далее создаем класс **NewsForm**, он будет наследником класса **forms.Form**. Если бы форма была связана с моделью, то класс был бы наследником



forms.ModelForms. Затем для созданного класса прописываем поля аналогично тому, как мы делали это для модели News. (рисунок 8)

Рисунок 8

Обратите внимание, что у класса **ModelChoiceField** есть обязательный аргумент **queryset**, где указывается, из какой модели будут взяты данные.

Теперь необходимо добавить созданную модель в шаблон. Для этого в файле views.py импортируем ее и дополним ранее созданную функцию add_news. (рисунок 9). А в файле add_news.html пропишем тег формы и кнопку отправки формы (рисунок 10).

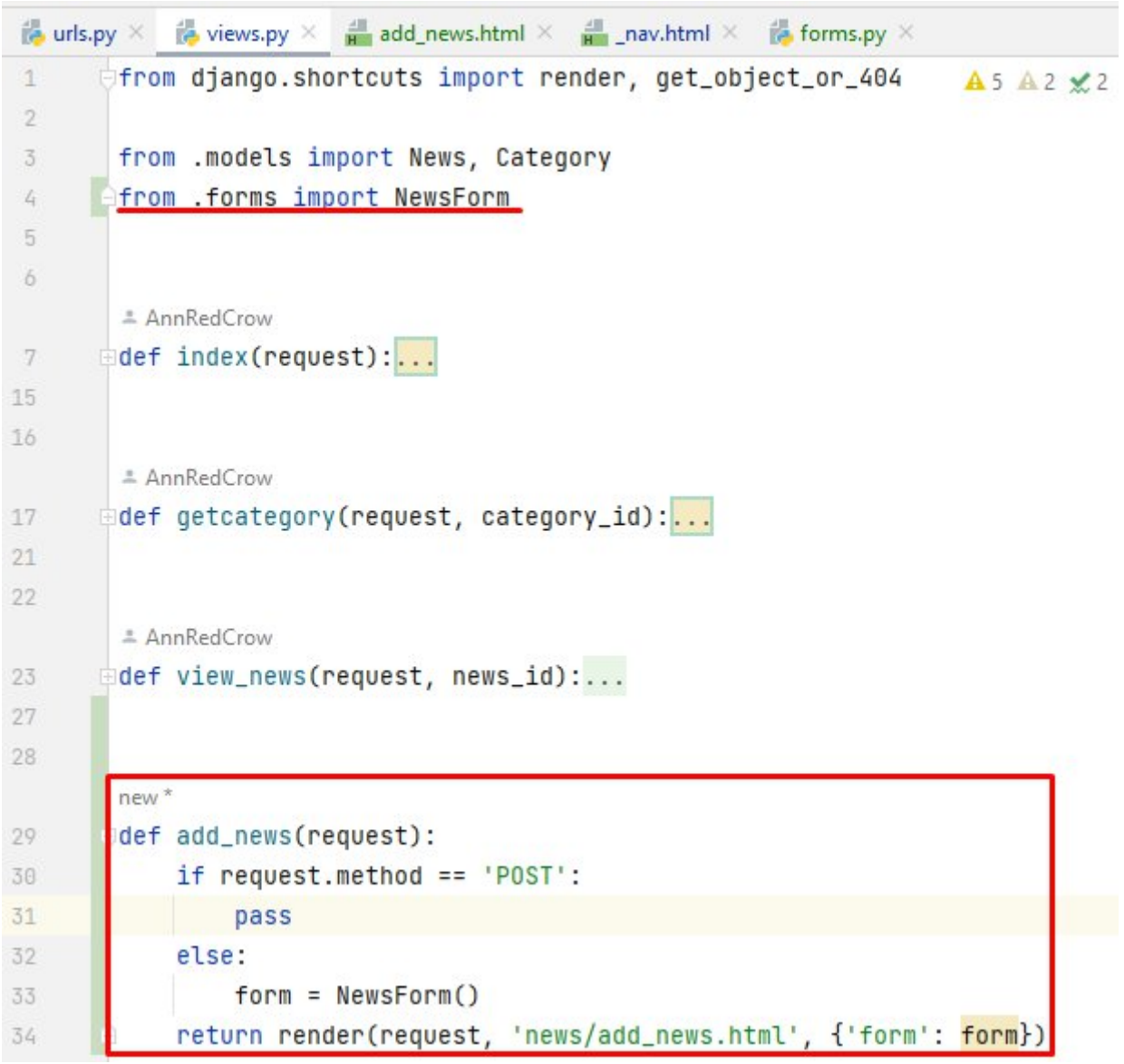

urls.py
views.py
add_news.html
_nav.html
forms.py
from django.shortcuts import render, get_object_or_404
from .models import News, Category
from .forms import NewsForm
AnnRedCrow
def index(request):...
AnnRedCrow
def getcategory(request, category_id):...
AnnRedCrow
def view_news(request, news_id):...
new *
def add_news(request):
if request.method == 'POST':
pass
else:
form = NewsForm()
return render(request, 'news/add_news.html', {'form': form})

Рисунок 9

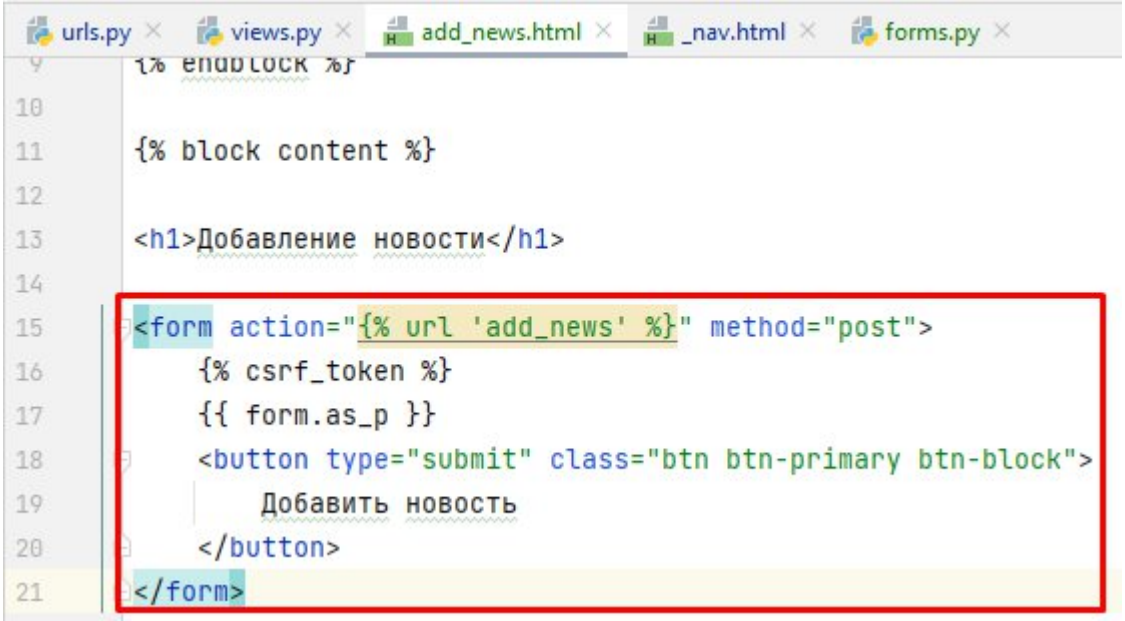


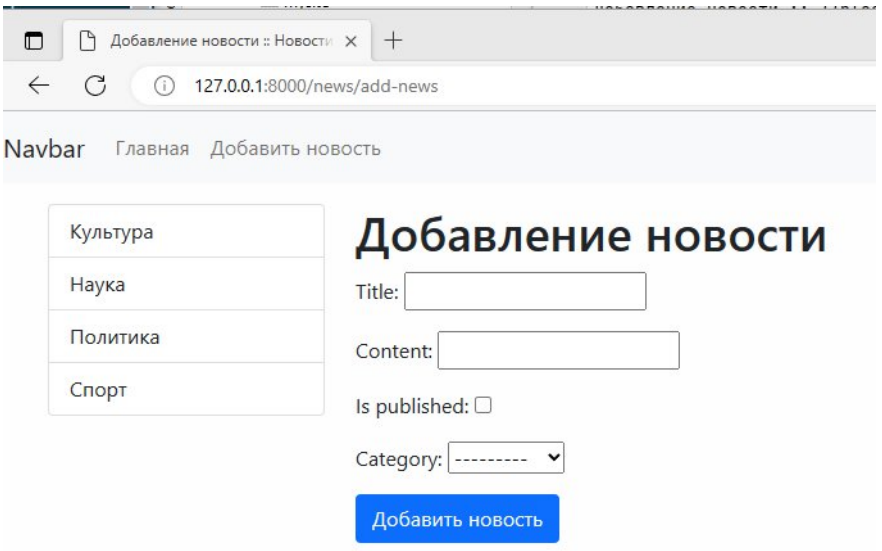
Рисунок 10

Обратим внимание на несколько моментов в коде:

- тег {{ form.as_p }} оборачивает каждое поле формы в параграф <p>
- {% csrf_token %} защищает форму от подделки межсайтовых запросов. Его значение каждый раз будет разным.

Результат работы этих тегов можно посмотреть в коде страницы(рисунок 12)

В результате наших действий форма будет отображаться на сайте.



(рисунок 11)

Рисунок 11

Рисунок 12

2. Настройка внешнего вида формы, сохранение данных

Первым шагом настроим поля формы: заменим английские названия на русские (атрибут **label**), укажем, что поле формы **context** является необязательным к заполнению (по умолчанию, в несвязанных формах все создаваемые поля обязательны к заполнению), зададим начальное значение для поля **is_published**. Также нам необходимо настроить выпадающий список. (рисунок 13) По умолчанию виджет **<select>**,

```
<h1>Добавление новости</h1>

<form action="/news/add-news" method="post">
    <input type="hidden" name="csrfmiddlewaretoken" value="TYO3vaogdzRiFGMgIu9OHsJTiVluVAfPZjH0i6DOdCDMmaM8kgXMWdR6zTMiSv2G">
    <p>
    <label for="id_title">Title:</label>
    <input type="text" name="title" maxlength="150" required id="id_title">


  </p>


  <p>
    <label for="id_content">Content:</label>
    <input type="text" name="content" required id="id_content">


  </p>


  <p>
    <label for="id_is_published">Is published:</label>
    <input type="checkbox" name="is_published" required id="id_is_published">


  </p>


  <p>
    <label for="id_category">Category:</label>
    <select name="category" required id="id_category">
  <option value="" selected>---------</option>

  <option value="1">Культура</option>

  <option value="4">Наука</option>

  <option value="3">Политика</option>

  <option value="2">Спорт</option>

</select>
```

используемый **ModelChoiceField**, будет иметь пустой выбор в верхней части списка. Чтобы изменить текст этой метки (который по умолчанию равен **"---------"**) необходимо использовать атрибут **empty_label**. (https://django.fun/ru/docs/django/4.1/ref/forms/fields/#modelchoicefield).

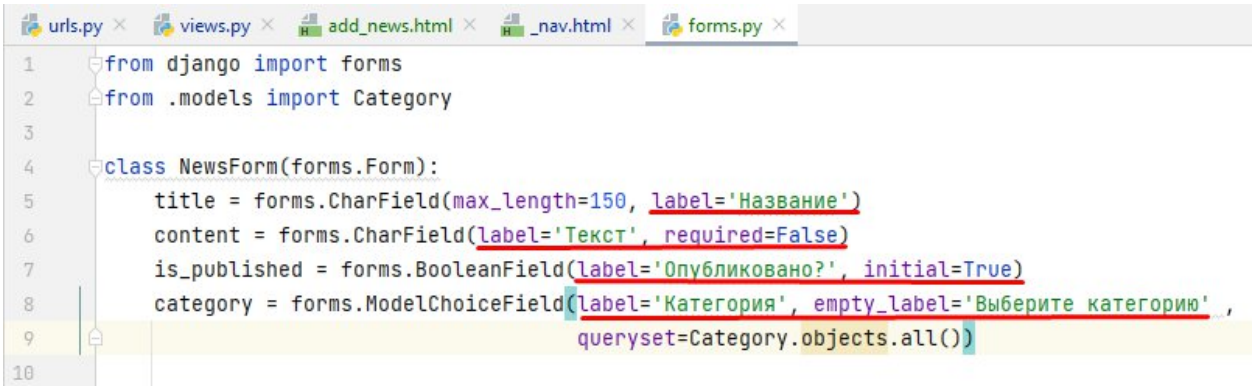


Рисунок 13

Далее приступим к оформлению внешнего вида формы, для этого будем использовать Bootstrap. Обратите внимание, что шаблоны могут отличаться в зависимости от используемой версии инструментария, обращайтесь к документации. В данном примере использовалась версия 5.0.2. Вы можете найти шаблоны по ссылке: https://bootstrap-4.ru/docs/5.0/forms/overview/.

Первым делом, закомментируйте автоматический рендеринг формы {{ form.as_p }}, далее мы рассмотрим ручной и полуавтоматический варианты. Начнем с первого, для этого в файле add_news.html необходимо прописать каждое поле отдельно. За красивый внешний вид формы отвечает класс «form-control», указываемый для input. Также обратите внимание на div «invalid-feedback», здесь мы указываем {{ form.title.errors }}. Таким образом, при некорректном заполнении полей будут выводиться ошибки валидации. Для вывода ошибок, не связанных с видимыми полями, используем директиву {{ form.non_field.errors }}. (рисунок 14) Представленный вариант рендеринга хоть и требует большого количества кода, но при этом делает настройку более гибкой.

Если вы сейчас обновите страницу, то обнаружите, что внешний вид формы не изменился. Чтобы это исправить, необходимо настроить атрибут **widget**

```
{% comment %}
{{ form.as_p }}
{% endcomment %}

{{ form.non_field.errors }}
<div class="mb-3">
    <label for="{{ form.title.id_for_label }}">Название</label>
    {{ form.title }}
    <div class="invalid-feedback">
        {{ form.title.errors }}
    </div>
</div>

<div class="mb-3">
    <label for="{{ form.content.id_for_label }}">Текст: </label>
    {{ form.content }}
    <div class="invalid-feedback">
        {{ form.content.errors }}
    </div>
</div>

<div class="mb-3">
    <label for="{{ form.is_published.id_for_label }}">Опубликовано?</label>
    {{ form.is_published }}
    <div class="invalid-feedback">
        {{ form.is_published.errors }}
    </div>
</div>

<div class="mb-3">
    <label for="{{ form.category.id_for_label }}">Категория: </label>
    {{ form.category }}
    <div class="invalid-feedback">
        {{ form.category.errors }}
    </div>
</div>
```

Рисунок 14

для полей формы. Обратите внимание, что для каждого встроенного поля назначен widget по умолчанию. Итак, передадим необходимые атрибуты в виде словаря **attrs**, например, класс «form-control» для применения настроек внешнего вида. (рисунок 15) Важно отметить, что для

полей title и content мы указывали встроенный класс **CharField**, но при этом

add_news.html forms.py

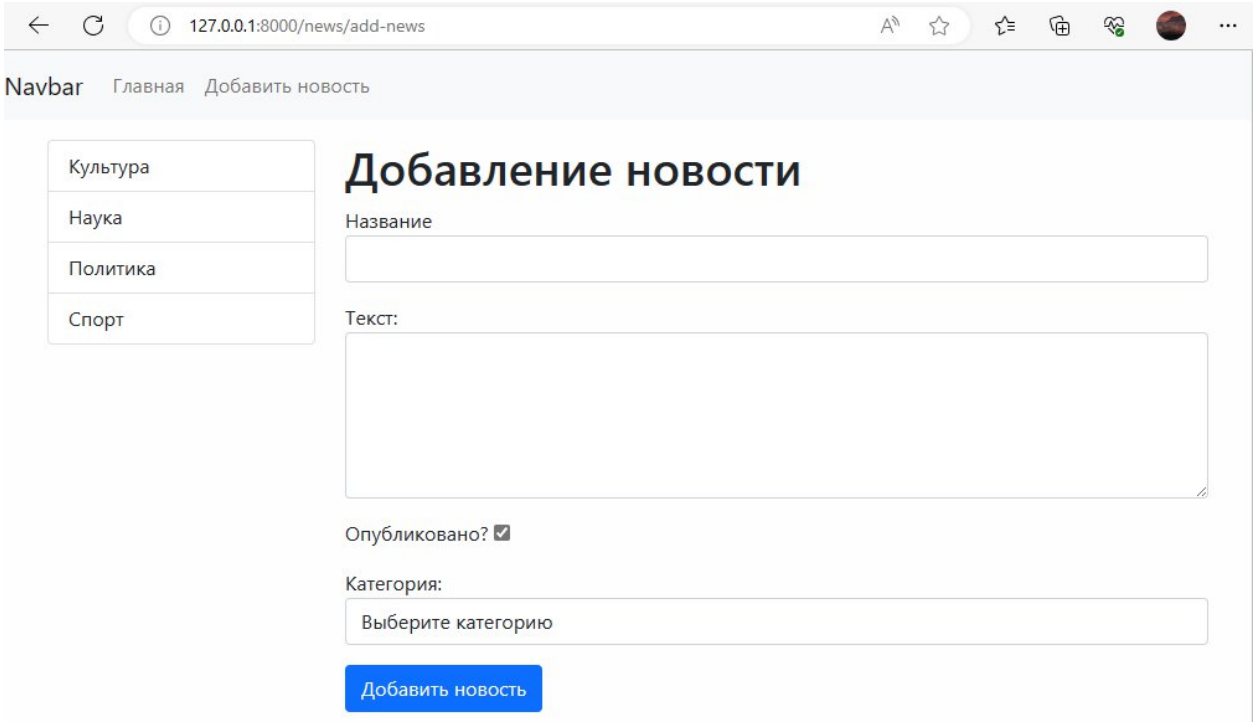
```python
from django import forms
from .models import Category

class NewsForm(forms.Form):
    title = forms.CharField(max_length=150, label='Название',
                            widget=forms.TextInput(attrs={"class": "form-control"}))
    content = forms.CharField(label='Текст', required=False,
                              widget=forms.Textarea(attrs={
                                  "class": "form-control",
                                  "rows": 5
                              }))
    is_published = forms.BooleanField(label='Опубликовано?', initial=True)
    category = forms.ModelChoiceField(label='Категория', empty_label='Выберите категорию' ,
                                      queryset=Category.objects.all(),
                                      widget=forms.Select(attrs={"class": "form-control"}))
```

прописали разные виджеты. **TextInput** для **title** (используется по умолчанию для всех полей класса **CharField**) и **Textarea** для **content**.

Рисунок 15

Теперь, обновив страницу, мы увидим, что изменения вступили в силу.



(рисунок 16)

Рисунок 16

Теперь, закомментируем рассмотренный нами ручной вариант рендеринга и перейдем к рассмотрению полуавтоматического. (рисунок 17) Для этого необходимо использовать цикл for. (рисунок 18)

В итоге, давайте остановимся на самом первом, полностью автоматическом варианте {{ **form.as_p** }} рендеринга, а остальные закомментируем.

add_news.html × forms.py ×

```
14
15  <form action="{% url 'add_news' %}" method="post">
16      {% csrf_token %}
17
18      {% comment %}
19      {{ form.as_p }}
20      {% endcomment %}
21
22      {% comment %}
23      {{ form.non_field.errors }}
24      <div class="mb-3"...>
31
32      <div class="mb-3"...>
39
40      <div class="mb-3"...>
47
48      <div class="mb-3"...>
55      {% endcomment %}
56
57      <button type="submit" class="btn btn-primary btn-block"...>
60  </form>
61
62  {% endblock %}
```

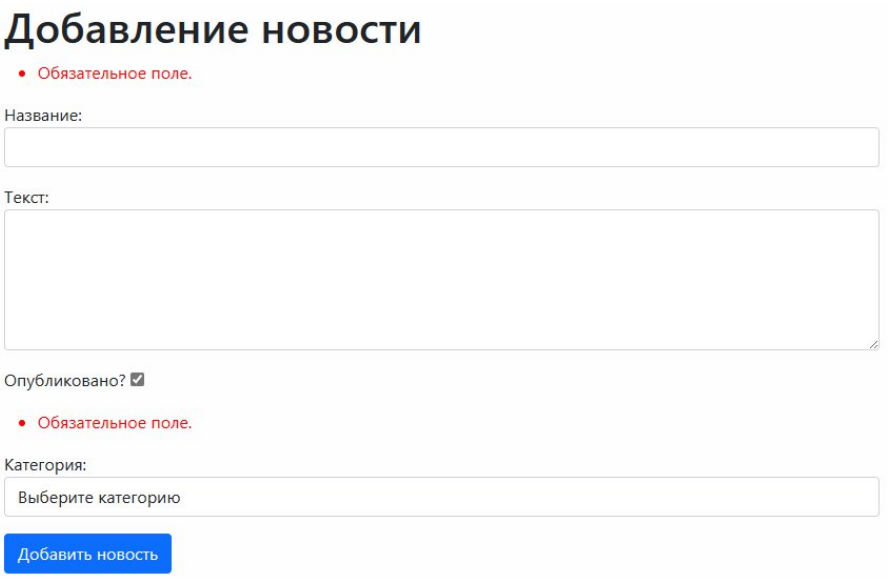
Рисунок 17

```
{% for field in form %}
    <div class="mb-3">
        {{field.label_tag}}
        {{field}}
        <div class="invalid-feedback">
            {{ field.errors }}
        </div>
    </div>
{% endfor %}
```

Рисунок 18

Осталось дописать функцию add_news в views.py, чтобы появилась возможность принимать данные заполненной формы. Запишем полученные данные в переменную **form**, укажем условие для проверки валидации

form.is_valid. В случае соблюдения условия, у формы появится атрибут



form.cleaned_data. В противном случае форма выведет ошибки валидации. (рисунок 19)

Рисунок 19

В завершение сохраним полученные данные. В данном случае нам необходимо приписывать это вручную, так как мы работаем с несвязанной с моделью формой. Вызовем метод **create** модели **News**. Для того, чтобы не прописывать каждое значение самостоятельно, воспользуемся распаковкой словарей, для это укажем ** перед form.cleaned_data. Уже на данном этапе создаваемая новость будет сохраняться. В дополнение настроим переход на страницу нового объекта. Для этого необходимо импортировать и применить метод **redirect**. (рисунок 20, рисунок 21)

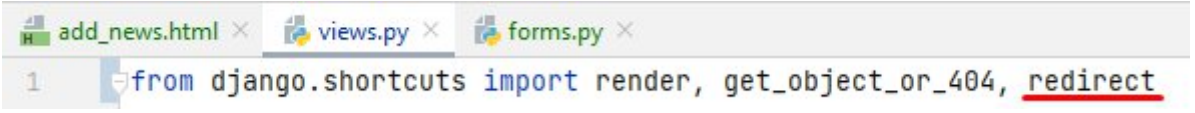


Рисунок 20

```
def add_news(request):
    if request.method == 'POST':
        form = NewsForm(request.POST)
        if form.is_valid():
            # print(form.cleaned_data)
            news = News.objects.create(**form.cleaned_data)
            return redirect(news)
    else:
        form = NewsForm()
    return render(request, 'news/add_news.html', {'form': form})
```

Рисунок 21

Теперь мы можем создавать новые новости через форму. (рисунок 22, рисунок 23, рисунок 24)



Рисунок 22



Рисунок 23

	ID	НАИМЕНОВАНИЕ	КАТЕГОРИЯ	ДАТА ПУБЛИКАЦИИ	ОБНОВЛЕНО	ОПУБЛИКОВАНО
☐	8	Новость из формы	Культура	28 июня 2023 г. 20:33	28 июня 2023 г. 20:33	☑
☐	7	Новость из админки	Культура	22 июня 2023 г. 14:51	23 июня 2023 г. 8:05	☑
☐	5	News 5	Наука	22 июня 2023 г. 10:07	23 июня 2023 г. 8:05	☐
☐	4	Новость 4	Политика	21 июня 2023 г. 19:43	23 июня 2023 г. 15:11	☑
☐	3	Новость 3	Спорт	21 июня 2023 г. 19:40	23 июня 2023 г. 8:09	☑
☐	2	Новость 2	Политика	21 июня 2023 г. 19:37	23 июня 2023 г. 8:06	☑
☐	1	Новость 1	Наука	21 июня 2023 г. 19:26	23 июня 2023 г. 10:53	☑

Рисунок 24

3. Создание формы, связанной с моделью

Теперь наша задача состоит в том, чтобы переделать ранее созданную форму, сделать ее связанной с моделью News.

Перейдем в файл forms.py, ранее мы были вынуждены самостоятельно прописывать поля формы, фактически дублируя код из модели. Теперь нам достаточно импортировать модель News и написать подкласс Meta для класса NewsForm. Удалите ранее написанный код, обратите внимание, что теперь класс NewsForm должен являться наследником класса **forms.ModelForm**).

add_news.html | views.py | forms.py | models.py

```
1  from django import forms
2  from .models import News
3
4
5  class NewsForm(forms.ModelForm):
6      class Meta:
7          model = News
8          fields = '__all__'
```

Для получения всех полей можно использовать волшебную строку ‘__all__’, но это считается плохой практикой. (рисунок 25, рисунок 26)

Рисунок 25

Рисунок 26

Рекомендуется прописывать названия полей самостоятельно. (рисунок 27, рисунок 28)

```python
class NewsForm(forms.ModelForm):
    class Meta:
        model = News
        # fields = '__all__'
        fields = ['title', 'content', 'is_published', 'category']
```

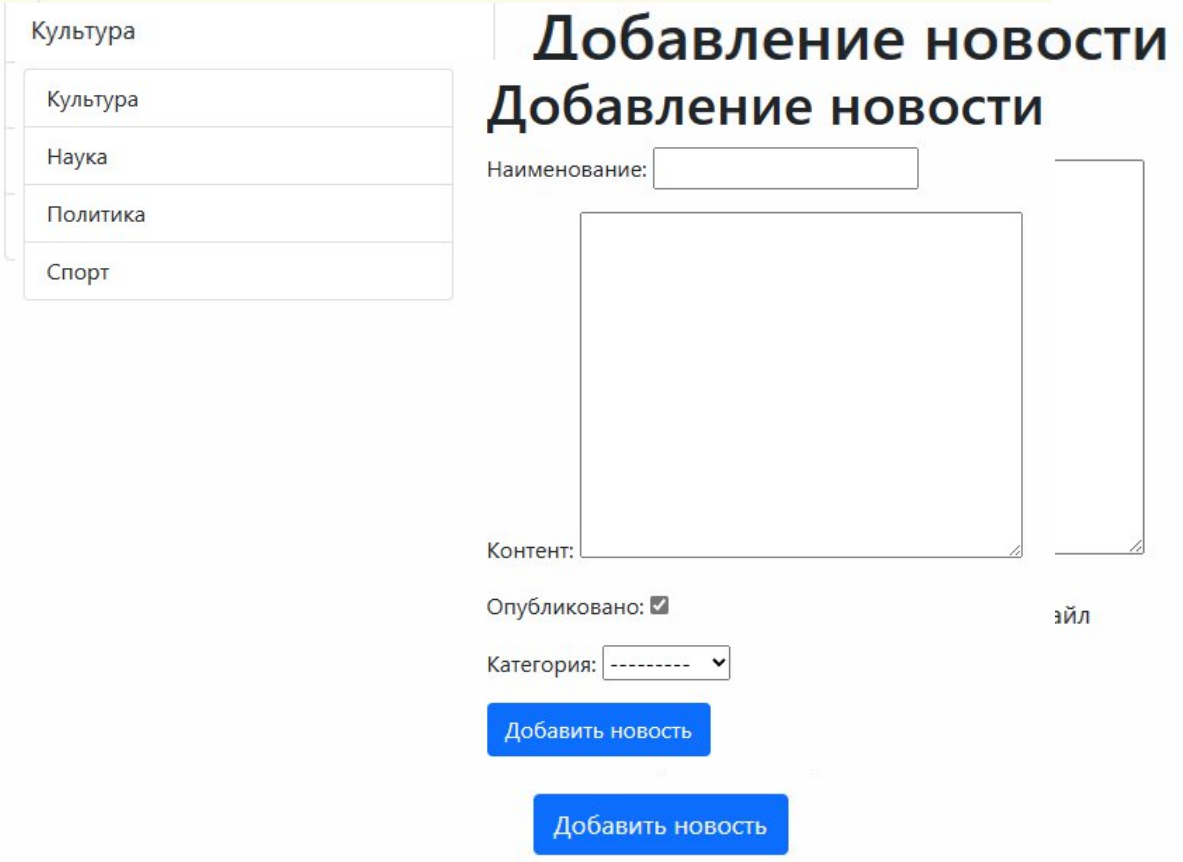


Рисунок 27

Рисунок 28

Для настройки отображения используем атрибут **widgets**, представленный в виде словаря. Здесь аналогично прошлому примеру

```
class NewsForm(forms.ModelForm):
    class Meta:
        model = News
        # fields = '__all__'
        fields = ['title', 'content', 'is_published', 'category']
        widgets = {
            'title': forms.TextInput(attrs={"class": "form-control"}),
            'content': forms.Textarea(attrs={
                                      "class": "form-control",
                                      "rows": 5
                                  }),
            'category': forms.Select(attrs={"class": "form-control"})
        }
```

прописываем классы полей и другие необходимые атрибуты. (рисунок 29)

Рисунок 29

Обновив страницу, убедитесь, что оформление аналогично предыдущему примеру.

Затем перейдем во views.py и изменим механизм сохранения данных, теперь нам не нужно прописывать все вручную, достаточно воспользоваться

```
def add_news(request):
    if request.method == 'POST':
        form = NewsForm(request.POST)
        if form.is_valid():
            # print(form.cleaned_data)
            # news = News.objects.create(**form.cleaned_data)
            news = form.save()
            return redirect(news)
    else:
        form = NewsForm()
    return render(request, 'news/add_news.html', {'form': form})
```

встроенным методом save(). (рисунок 30)

Рисунок 30

В завершение работы создала еще одну новость, убедилась, что она сохранилась через админку. (рисунок 31)



Рисунок 31

4. Кастомные валидаторы

Иногда бывают ситуации, когда стандартных валидаторов недостаточно и необходимо каким-то дополнительно провалидировать поле формы. Для решения этой задачи можно создавать пользовательские(кастомные) валидаторы.

Напишем собственный валидатор для поля **title** формы NewsForm, проверяющий, чтобы введенное значение не начиналось с цифры. Он будет отрабатывать после стандартных проверок. Откроем файл forms.py, импортируем модуль для работы с регулярными выражениями и ошибку валидации из модуля **django.core.exceptions**, напишем саму функцию. (рисунок 32)

forms.py

```python
from django import forms
from .models import News
import re
from django.core.exceptions import ValidationError


AnnRedCrow *
class NewsForm(forms.ModelForm):
    AnnRedCrow
    class Meta:...

    new *
    def clean_title(self):
        title = self.cleaned_data['title']
        if re.match(r'\d', title):
            raise ValidationError('Название не должно начинаться с цифры')
        return title
```

Рисунок 32

Теперь при попытке добавления новости, чей заголовок начинается с цифры, мы будем получать сообщение об ошибке, и создаваемый объект не будет сохраняться в БД. (рисунок 33)

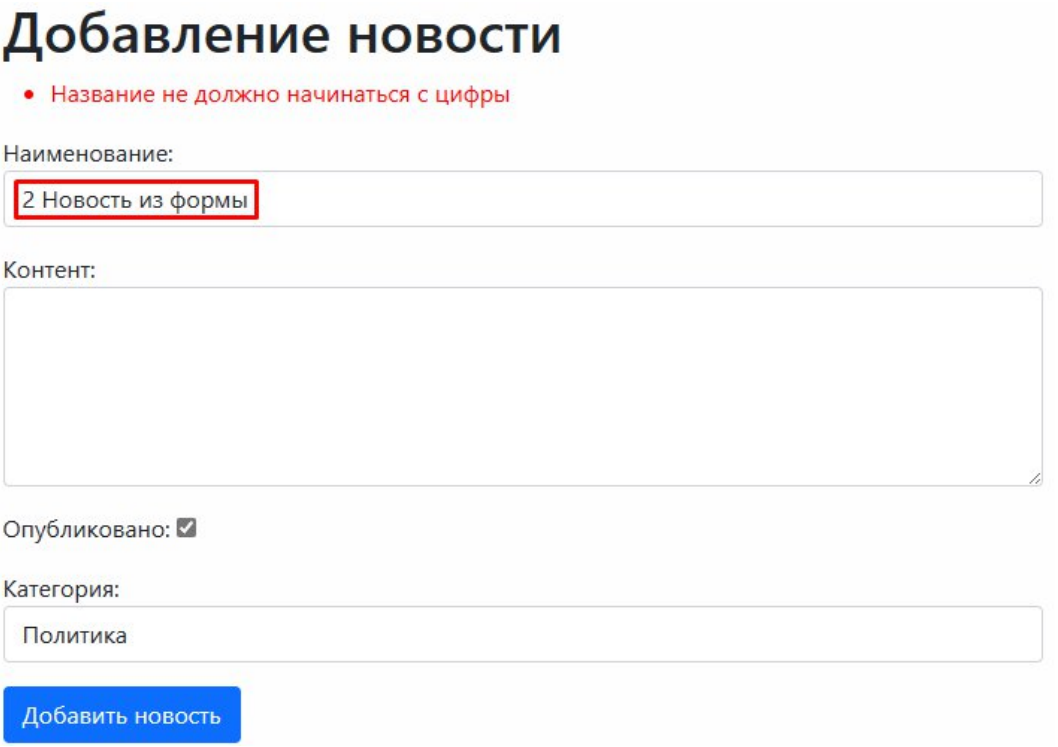


Рисунок 33

Лабораторная работа № 12.
Классы-контроллеры

Цель работы: Изучить и освоить создание представлений на основе классов (классов-контроллеров) в Django, включая использование встроенных классов ListView, DetailView, CreateView. Научиться настраивать шаблоны, контекстные данные, фильтрацию queryset, передачу дополнительных параметров и обработку форм без написания функций-контроллеров в views.py.

Теоретическое обоснование
Представления на основе классов

Ранее мы использовали контроллеры-функции для обработки получаемых запросов и возврата отрисованных и заполненных шаблонов, их

мы прописывали в файле views.py. Теперь же рассмотрим вариант написания представлений на основе классов.

Ознакомьтесь с документацией для более полного понимания темы:

https://django.fun/ru/docs/django/4.1/topics/class-based-views/

Встроенный API представлений на основе классов | Документация Django 4.1 | Все о фреймворке Джанго и его библиотеках

1. Класс ListView

В данной лабораторной работе мы рассмотрим напишем контроллеры-классы, работающие со списками и заменяющие ранее написанные функции **index()** и **get_category()**. Для начала работы импортируйте класс **ListView** в файле **views.py**. Теперь мы можем создать свой класс, наследующий **ListView**, и переопределить и\или использовать атрибуты и функции родителя. Укажем модель-источник, из которой будут браться данные. (рисунок 1)



Рисунок 1

Далее отредактируем ссылку в файле **urls.py**, теперь вместо названия функции **index** напишем название созданного класса и вызовем метод **as_view()**(не забудьте указать скобки). (рисунок 2)

Этого количества кода уже может быть достаточно для корректной работы контроллера, однако, если вы попробуете обновить страницу

браузера, то столкнетесь с ошибкой, где будет указано, что шаблон не найден. Дело в том, что он ищет шаблон по умолчанию.



Рисунок 2

Название шаблона строится автоматически. В нашем примере это будет **news/news_list.html**, где первый news – название приложения, второй – название модели в нижнем регистре и _list – постфикс. Если вы создадите шаблон с таким названием, то ошибка исчезнет. Перенесем код из шаблона index.html, заменим в цикле for **news** на **object_list**. Ранее мы в функции указывали контекст, где в переменной news записывали результат запроса к модели, получающий все записи, сейчас же нам достаточно будет обратиться к встроенному объекту **object_list.** (рисунок 3)

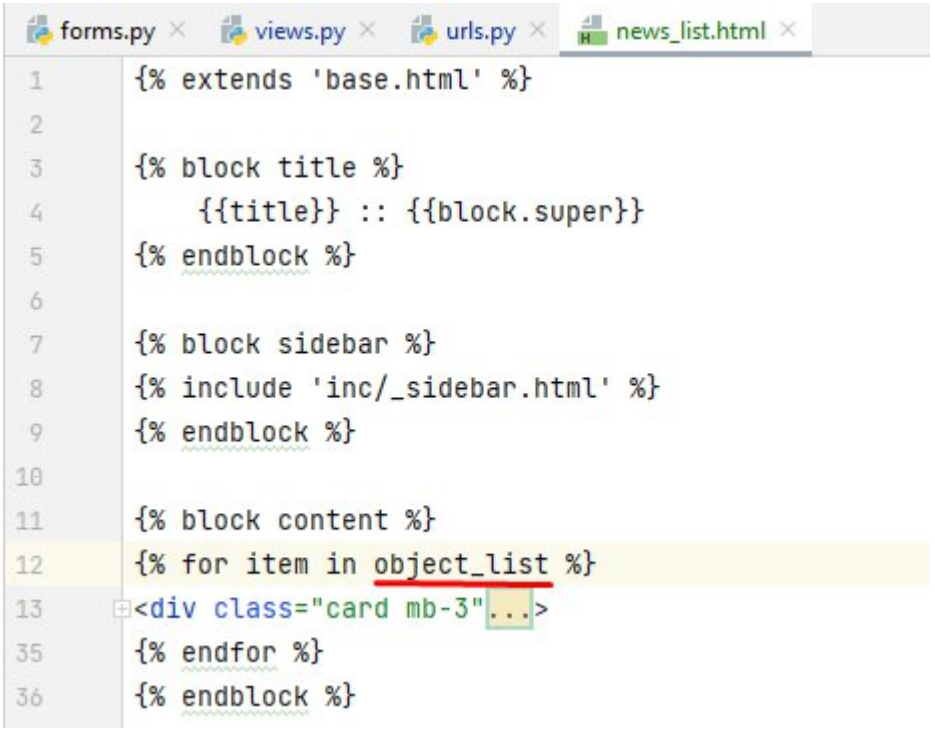


Рисунок 3

Однако, если вам необходимо использовать шаблон с другим названием, то достаточно указать его в переменной **template_name**. Вы также можете передавать и контекст в атрибут **context_object_name**. Создадим новый шаблон home_news_list.html, перенесем тот же код, что на рисунке 3, но теперь вернем **news** вместо **object_list**. (рисунок 4) Передадим название шаблона и контекст в атрибуты класса. (рисунок 5) Теперь при обновлении страницы браузера вы увидите тот же результат, что и при работе функции **index**.

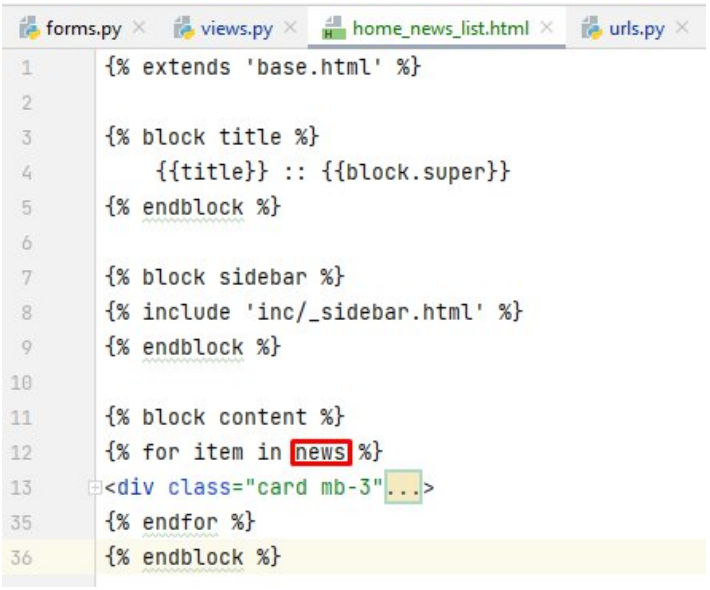


Рисунок 4

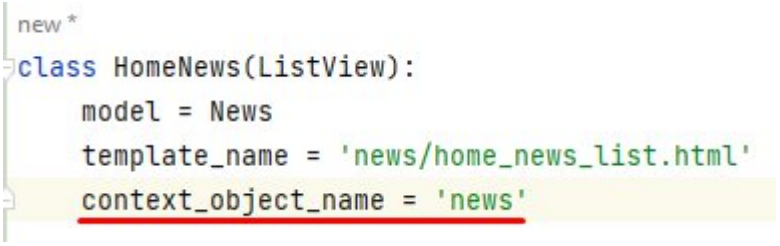


Рисунок 5

Зачастую бывают ситуации, когда необходимо получать какие-то дополнительные данные помимо самого списка. Для этого можно использовать как атрибут класса **extra_context**, так и метод **get_context_data**. Атрибут рекомендуется использовать только со статическими данными. В методе мы записываем в переменную **context** все данные, хранившиеся в контексте до переопределения метода, далее дополняем контекст его своими значениями. В нашем случае мы добавили title для страницы. В конце метода возвращаем измененный контекст. (рисунок 6)

На данном этапе у нас на главной странице отображаются все новости независимо от того, какое значение имеет атрибут **is_published**. Давайте исправим это таким образом, чтобы отображались только опубликованные.

```python
class HomeNews(ListView):
    model = News
    template_name = 'news/home_news_list.html'
    context_object_name = 'news'
    # extra_context = {'title': 'Главная'}

    new *
    def get_context_data(self, *, object_list=None, **kwargs):
        context = super().get_context_data(**kwargs)
        context['title'] = 'Главная страница'
        return context
```

Рисунок 6

Для этого используем метод get_queryset, где применим фильтр к данным, получаемым из модели. (рисунок 7)

```python
class HomeNews(ListView):
    model = News
    template_name = 'news/home_news_list.html'
    context_object_name = 'news'
    # extra_context = {'title': 'Главная'}

    new *
    def get_context_data(self, *, object_list=None, **kwargs):
        context = super().get_context_data(**kwargs)
        context['title'] = 'Главная страница'
        return context

    new *
    def get_queryset(self):
        return News.objects.filter(is_published=True)
```

Рисунок 7

Теперь пользователь будет видеть только опубликованные новости, остальные же будут скрыты. (рисунок 8, рисунок 9)

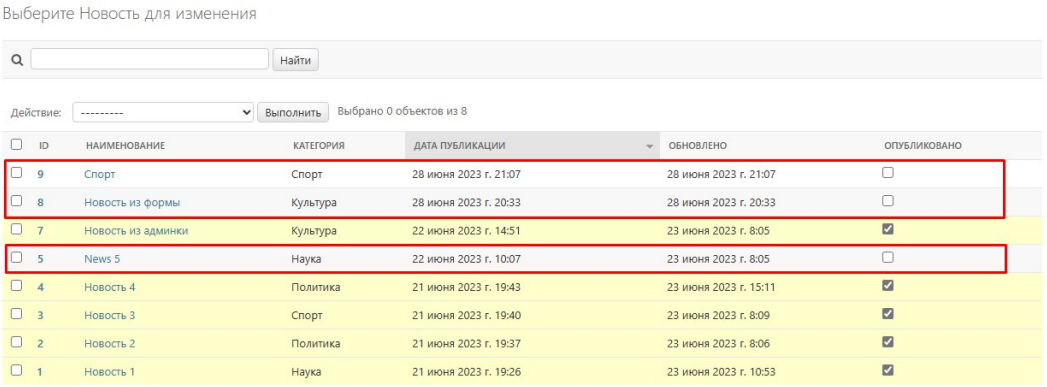


Рисунок 8

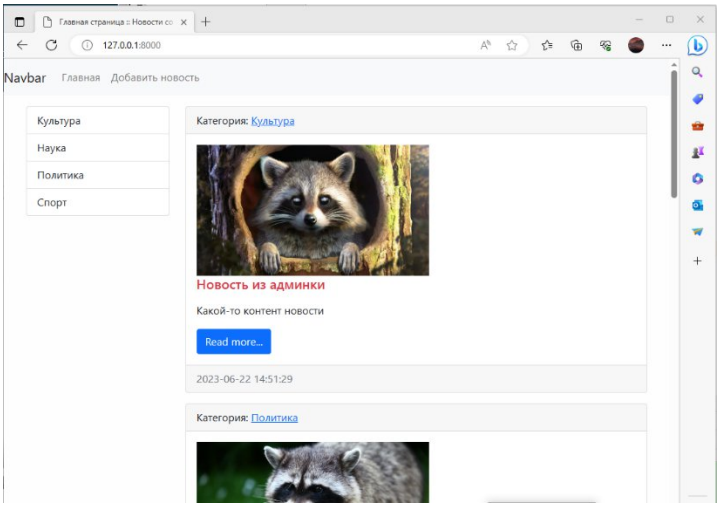


Рисунок 9

Аналогичным способом напишем класс, заменяющий функцию get_category, назовем его **NewsByCategory**. Для атрибутов укажем те же значения, что и в прошлом примере. Здесь мы также напишем метод get_queryset, где будем фильтровать данные модели по идентификатору категории и значению поля is_published. (рисунок 10)

```
new *
class NewsByCategory(ListView):
    model = News
    template_name = 'news/home_news_list.html'
    context_object_name = 'news'

    new *
    def get_queryset(self):
        return News.objects.filter(category_id=self.kwargs['category_id'], is_published=True)
```

Рисунок 10

Стоит отметить, что вы можете указывать дополнительный контекст не только как атрибут или метод, но и как параметр метода as_view. (рисунок 11)

Если определенной категории новостей не существует или она не содержит никаких записей, то вы можете воспользоваться атрибут **allow_empty** и установить его значение в **False**. (рисунок 12, на нем также показано переписывание метода получения контекста)



Рисунок 11



Рисунок 12

В результате мы будем получать только опубликованные записи, относящиеся к выбранной категории. Контроллеры-функции, используемые ранее уберите или закомментируйте.

2. Класс **DetailView**

Пока представление на основе этого класса выполняется, в **self.object** будет содержать объект, над которым оно работает. Создадим свой класс, наследующий класс DetailView, для замены контроллера-функции **view_news(request, news_id)**.

Начнем с того, что импортируем класс в файле **views.py**. (рисунок 13)

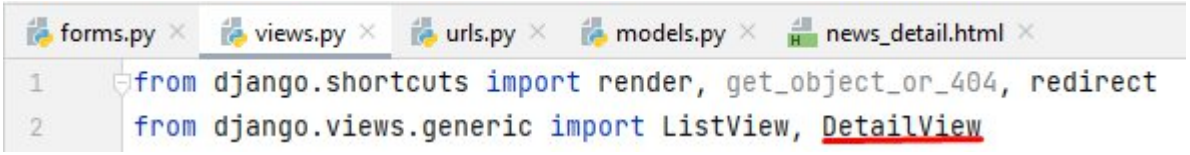


Рисунок 13

Напишем новый класс **ViewNews**, укажем модель **News** как источник данных. (рисунок 14)

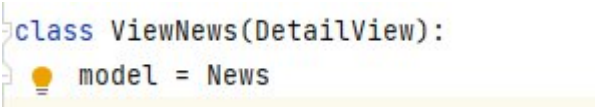


Рисунок 14

Если вы сейчас обновите страницу браузера, то столкнетесь с ошибкой **AttributeError**, где указано, что требуется объект pk или slug для вызова класса (рисунок 15). Ранее мы не сталкивались с подобными ошибками.

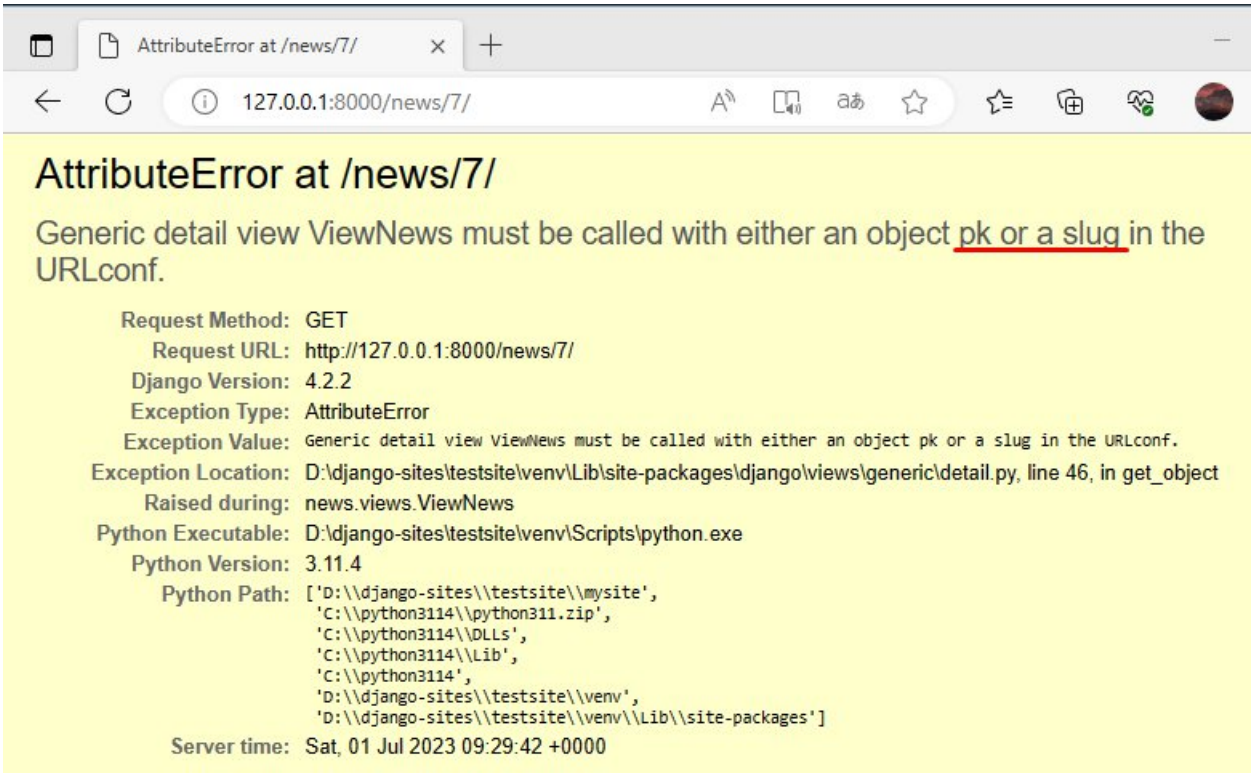


Рисунок 15

Самым быстрым и эффективным решением будет заменить передаваемый объект с **category_id** на **pk** в файлах **urls.py** и **models.py**(метод **get_absolute_url**). (рисунок 16, рисунок 17)



Рисунок 16



Рисунок 17

Теперь, обновив страницу браузера, мы столкнемся с уже знакомой нам ошибкой, указывающей на отсутствие искомого шаблона. Для ее решения создадим шаблон news_detail.html, куда полностью скопируем содержимое файла view_news.html. В итоге получим отображение пустого шаблона. (рисунок 18)

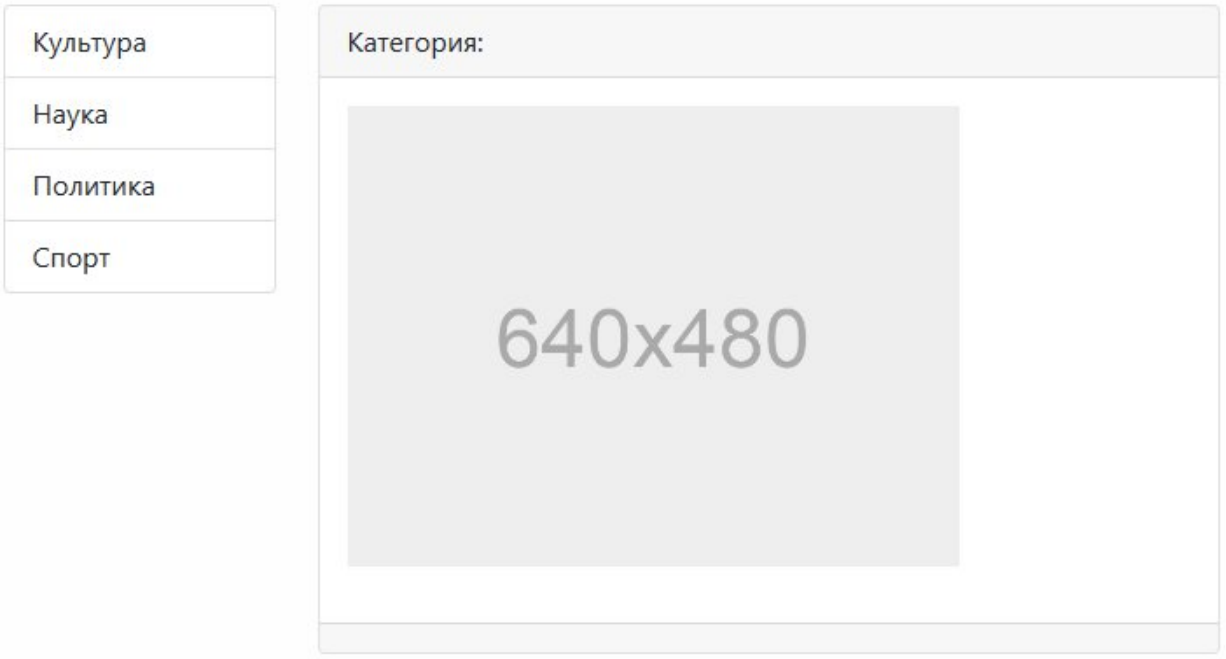


Рисунок 18

Чтобы заполнить шаблон данными есть 2 способа: заменить в html-файле все **news_item** на **object**(именно так при использовании DetailView можно обратиться к объекту), либо прописать значение атрибута **context_object_name** класса, переопределив значение по умолчанию. Второй вариант будет более быстрым и эффективным с учетом уже написанного шаблона. (рисунок 19)

```
new *
class ViewNews(DetailView):
    model = News
    context_object_name = 'news_item'
    # pk_url_kwarg = 'news_id'
    # template_name = 'news/news_detail.html'
```

Рисунок 19

В результате получится тот же результат, что и при использовании функции-контроллер. Уберите или закомментируйте view_news(request, news_id).

3. Класс **CreateView**

Этот класс отображает форму для создания объекта, ошибки валидации(при наличии) и сохраняет объект в базе данных.

https://djangodoc.ru/3.2/ref/class-based-views/generic-editing/#createview

В нашей работе мы создадим класс, заменяющий функцию **add_news()**, написанную ранее. Первым шагом импортируем класс **CreateView**. (рисунок 20)

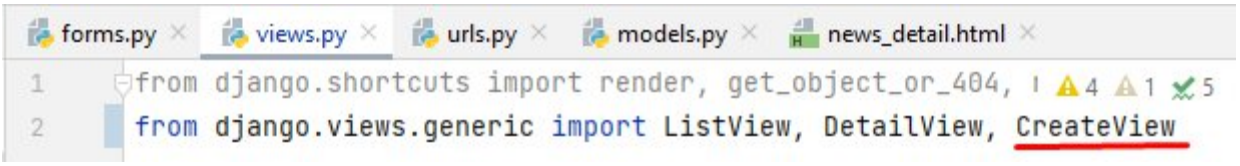


Рисунок 20

Далее создадим новый класс **CreateNews**, укажем его родителя. В отличии от 2 прошлых классов, здесь мы должны указать не модель, а название формы. Также сразу откажемся от использования шаблона по умолчанию и определим свой, уже созданный вариант. (рисунок 21)

```
new *
class CreateNews(CreateView):
    form_class = NewsForm
    template_name = 'news/add_news.html'
```

Рисунок 21

Также поправим ссылку, заменив название функции add_news на метод as_view() класса CreateNews. (рисунок 22)

```
urlpatterns = [
    # path('', index, name='home'),
    path('', HomeNews.as_view(), name='home'),
    # path('category/<int:category_id>/', getcategory, name='category'),
    path('category/<int:category_id>/', NewsByCategory.as_view(), name='c
    # path('news/<int:news_id>/', view_news, name="view_news"),
    path('news/<int:pk>/', ViewNews.as_view(), name="view_news"),
    # path('news/add-news', add_news, name='add_news'),
    path('news/add-news', CreateNews.as_view(), name='add_news'),
]
```

Рисунок 22

На данном этапе уже можно закончить работу с созданным классом, он будет реализовывать все возложенные на него задачи. (рисунок 23, рисунок 24)

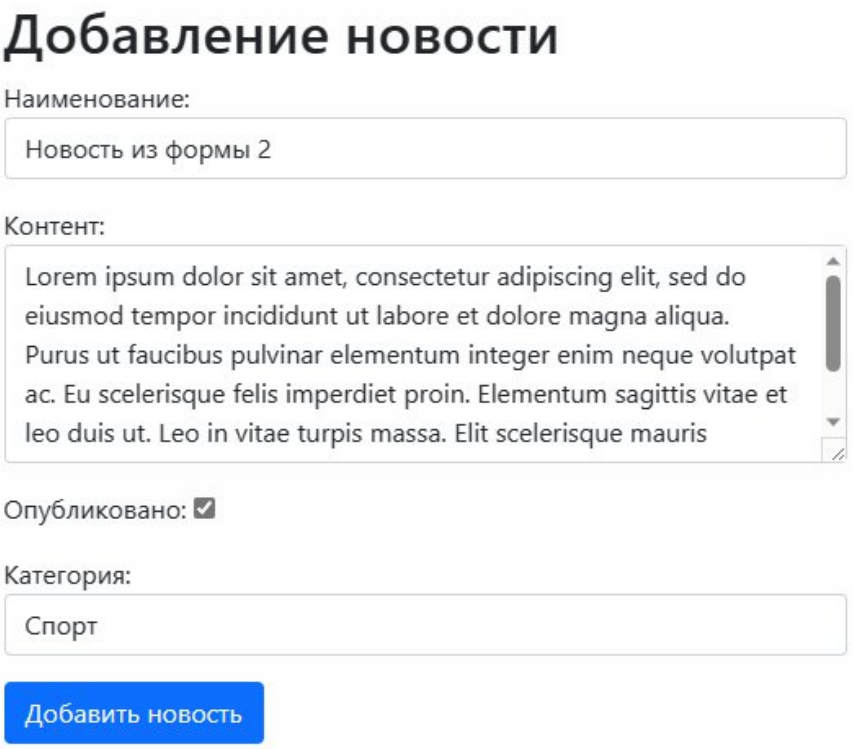


Рисунок 23



Рисунок 24

Обратите внимание, что после добавления новости пользователь автоматически переадресовывается на страницу отображения созданного объекта. Ранее для получения такого эффекта мы пользовались функцией redirect. В случае с классом этот переход реализован по умолчанию. Чтобы изменить страницу, на которую переадресовывают пользователя, вы можете воспользоваться функцией reverse_lazy. (рисунок 25, рисунок 26)

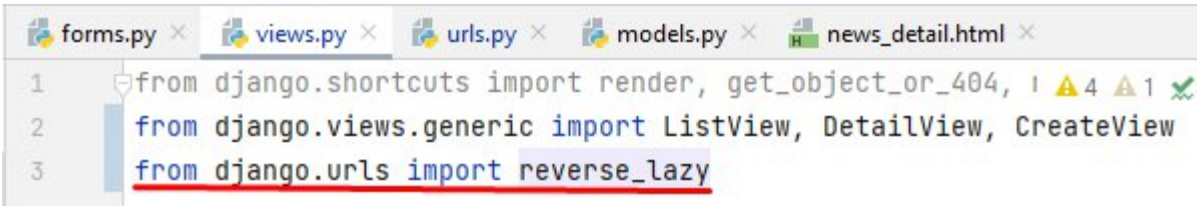


Рисунок 25

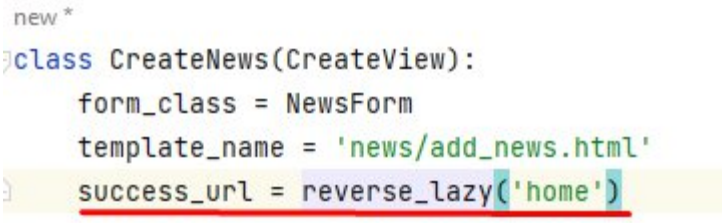


Рисунок 26

Варианты индивидуальных заданий

Вариант 1

Создайте ListView для модели Category, который выводит список всех категорий. Переопределите template_name на catalog/categories.html, а context_object_name — на categories. Добавьте в контекст заголовок страницы «Список категорий».

Вариант 2

Реализуйте ListView для новостей, который выводит только те записи, у которых is_published = True и дата публикации не позже текущей (использовать фильтр по дате). Используйте метод get_queryset().

Вариант 3

Создайте DetailView для модели Category. Настройте шаблон category_detail.html. В контексте объект должен называться category. Добавьте внизу страницы ссылку на список новостей этой категории.

Вариант 4

Реализуйте CreateView для модели Category с формой, содержащей поля name и slug. После создания перенаправляйте на список категорий через reverse_lazy.

Вариант 5

Измените NewsByCategory так, чтобы он дополнительно выводил количество новостей в категории (добавить в extra_context или через get_context_data).

Вариант 6

Создайте ListView, который отображает только последние 5 опубликованных новостей. Используйте сортировку по дате убывания и ограничение выборки ([:5] в get_queryset).

Вариант 7

Реализуйте DetailView для новостей, где вместо pk используется slug для идентификации новости. Настройте urls.py соответствующим образом.

Вариант 8

Создайте класс UserNewsListView, который наследуется от ListView и выводит только те новости, которые созданы текущим авторизованным пользователем (использовать request.user в get_queryset).

Вариант 9

Добавьте в CreateView для новостей автоматическую подстановку текущего пользователя в поле author (переопределите form_valid).

Вариант 10

Реализуйте UpdateView для модели News (изучите документацию). Разрешите редактирование только опубликованных новостей. Используйте тот же шаблон, что и для CreateView.

Вариант 11

Создайте DeleteView для новостей с подтверждением удаления. После удаления перенаправляйте на главную страницу. Добавьте кнопку удаления на страницу детального просмотра.

Вариант 12

Используйте ListView с пагинацией (атрибут paginate_by = 3). Добавьте в шаблон навигацию по страницам.

Вариант 13

Создайте ListView, который выводит новости, отфильтрованные по поисковому запросу (использовать request.GET в get_queryset и фильтр по заголовку).

Вариант 14

В CreateView для новостей добавьте проверку: если новость с таким заголовком уже существует — показывать ошибку валидации (переопределить метод form_valid или использовать clean_title в форме).

Вариант 15

Создайте миксин (класс-примесь) PublishedFilterMixin, который добавляет в get_queryset фильтр по is_published = True. Примените его к ListView новостей и к NewsByCategory.

Контрольные вопросы

1. В чем основное различие между представлениями-функциями и представлениями на основе классов в Django?

2. Для каких целей предназначен класс ListView и какие атрибуты (model, template_name, context_object_name) он позволяет переопределять?

3. Как Django автоматически определяет имя шаблона для ListView, если не указан template_name? Приведите пример.

4. Зачем нужен метод get_queryset() в классе-представлении и как с его помощью реализовать фильтрацию данных?

5. Какими двумя способами можно добавить дополнительные данные в контекст шаблона для классового представления?

6. Что делает метод as_view() при подключении классового представления в urls.py?

7. Какой атрибут или параметр позволяет запретить отображение пустого списка в ListView?

8. Для чего используется класс DetailView и как он идентифицирует запрашиваемый объект (через pk или slug)?

9. Чем отличается работа с контекстным именем объекта в DetailView (по умолчанию object) от ListView?

10. Как выполняется перенаправление после успешного создания объекта в CreateView и как его изменить с помощью reverse_lazy?

Лабораторная работа №13

Работа с Django ORM

Для начала разберемся, что из себя представляет ORM. ORM – это поход в программировании, позволяющий взаимодействовать с базой данных представляя данные в виде объектов. Он скрывает сложности SQL-запросов и позволяет разработчикам работать с данными на более высоком уровне абстракции, упрощая разработку и поддержку приложений.

Ранее мы уже немного сталкивались с использованием этого подхода. Например, для получения всех новостей из таблицы News в консоли приложения вызывали команду News.objects.all(), а не писали SQL-запрос SELECT * FROM news в среде разработки базы данных.

Документация Django на русском и английском языках:

https://django.fun/ru/docs/django/4.1/ref/models/querysets/

https://docs.djangoproject.com/en/4.2/ref/models/querysets/

В ходе выполнения лабораторной работы мы рассмотрим возможности использования ORM в Django. Прим. Во всех примерах мы будем обращаться к созданным нами моделям News и Category.

Для начала вызовите в терминале командную оболочку Django и импортируйте используемые модели:

python manage.py shell

from news.models import News, Category

Теперь перейдем к получению данных.

News.objects.all() – получение всех записей таблицы

Обратите внимание, что ранее мы настроили сортировку объектов по дате создания через атрибут ordering класса Meta модели News.

News.objects.order_by() – получение записей, отсортированных по заданному полю. (рисунок 1)

```
>>> News.objects.all()
<QuerySet [<News: Новость из формы 2>, <News: Спорт>, <News: Новость из формы>, <News: Новость из админки>
, <News: News 5>, <News: Новость 4>, <News: Новость 3>, <News: Новость 2>, <News: Новость 1>]>
>>> News.objects.order_by('pk')
<QuerySet [<News: Новость 1>, <News: Новость 2>, <News: Новость 3>, <News: Новость 4>, <News: News 5>, <Ne
ws: Новость из админки>, <News: Новость из формы>, <News: Спорт>, <News: Новость из формы 2>]>
>>> News.objects.order_by('-pk')
<QuerySet [<News: Новость из формы 2>, <News: Спорт>, <News: Новость из формы>, <News: Новость из админки>
, <News: News 5>, <News: Новость 4>, <News: Новость 3>, <News: Новость 2>, <News: Новость 1>]>
```

Рисунок 1

Знак «-» перед названием поля обозначает сортировку в обратном порядке.

Также для изменения порядка на обратный можно воспользоваться методом reverse. (рисунок 2)

News.objects.all().reverse()

```
>>> News.objects.all().reverse()
<QuerySet [<News: Новость 1>, <News: Новость 2>, <News: Новость 3>, <News: Новость 4>, <News: News 5>, <Ne
ws: Новость из админки>, <News: Новость из формы>, <News: Спорт>, <News: Новость из формы 2>]>
```

Рисунок 2

News.objects.get(pk=1) – получение **одной** записи, соответствующей заданному параметру. В нашем примере мы получаем запись со значением первичного ключа pk равного 1, однако могут использоваться любые поля, например, title. (рисунок 3)

Зачастую в качестве параметра передаются **первичный ключ** или **slug**, при отсутствии записей или нахождении более одной записи будет выдаваться ошибка.

```
>>> News.objects.get(pk=1)
<News: Новость 1>
```

Рисунок 3

Результат запроса вы можете записать в переменную и просматривать значения ее атрибутов. Обратите внимание, что также возможно получение связанных данных, например категории. При вызове news5.category будет получен объект. (рисунок 4) Обратите внимание на отображение объекта, оно было настроено нами через метод __str__ модели Category.

```
>>> news5 = _
>>> news5
<News: Новость 1>
>>> news5.is_published
True
>>> news5.category
<Category: Наука>
>>> news5.category.pk
4
```

Рисунок 4

В последней строке news5.category.pk мы обратились к первичной модели Category через вторичную News через их связь. Также можно пользоваться обратной связью между моделями. Например, получить все новости, относящиеся к конкретной категории. Для этого воспользуемся атрибутом первичной модели (рисунок 5):

<Название первичной модели>.<Название связанной модели в нижнем регистре>_set

```
>>> cat4 = Category.objects.get(pk=4)
>>> cat4
<Category: Наука>
>>> cat4.news_set.all()
<QuerySet [<News: News 5>, <News: Новость 1>]>
```

Рисунок 5

Атрибут **_set** можно переназначить. Для этого во вторичной модели для атрибута внешнего ключа необходимо указать параметр **related_name**. (рисунок 6) Мы оставим атрибут по умолчанию.

```python
class News(models.Model):
    title = models.CharField(max_length=150, verbose_name='Наименование')
    content = models.TextField(blank=True, verbose_name='Контент')
    created_at = models.DateTimeField(auto_now_add=True, verbose_name='Дата публикации')
    updated_at = models.DateTimeField(auto_now=True, verbose_name='Обновлено')
    photo = models.ImageField(upload_to='photos/%Y/%m/%d', verbose_name='Фото', blank=True)
    is_published = models.BooleanField(default=True, verbose_name='Опубликовано')
    category = models.ForeignKey('Category', on_delete=models.PROTECT,
                                 null=True, verbose_name='Категория', related_name='get_news')
```

Рисунок 6

Операторы фильтрации

Для более сложных запросов можно использовать фильтры полей. Они имеют следующий синтаксис: **<имя поля>__<фильтр>**. Это своеобразный аналог WHERE в SQL Все фильтры можно посмотреть в документации: https://django.fun/ru/docs/django/4.1/ref/models/querysets/#field-lookups

Рассмотрим некоторые из них:

News.objects.filter(pk__gt=5) – возвращает все записи, где значение поля больше указанного параметра.

News.objects.filter(pk__gte=5) – возвращает все записи, где значение поля больше или равно указанному параметру. (рисунок 7)

```
>>> News.objects.filter(pk__gt=5)
<QuerySet [<News: Новость из формы 2>, <News: Спорт>, <News: Новость из формы>, <News: Новость из админки>
]>
>>> News.objects.filter(pk__gte=5)
<QuerySet [<News: Новость из формы 2>, <News: Спорт>, <News: Новость из формы>, <News: Новость из админки>
, <News: News 5>]>
```

Рисунок 7

News.objects.filter(title__contains='<какая-то строка>') – независимый от регистра фильтр, ищущий записи, в чьих полях содержится указанная строка.

News.objects.filter(title__icontains='<какая-то строка>') – зависимый от регистра фильтр, ищущий записи, в чьих полях содержится указанная строка.

Тут есть одна очень важная особенность, связанная с используемой нами базой данных Sqlite3. Дело в том, что данная БД нечувствительна к

регистру строк, написанных ASCII символами, таким образом обе команды будут возвращать одинаковые результаты. В то же время Sqlite3 будет учитывать регистр символов, не входящих в ASCII(кириллицы), всегда, и итог выполнения запросов будет также одинаковым. (рисунок 8)

Подробнее: https://django.fun/ru/docs/django/4.1/ref/databases/#sqlite-string-matching

```
>>> News.objects.filter(title__contains='new')
<QuerySet [<News: News 5>]>
>>> News.objects.filter(title__icontains='new')
<QuerySet [<News: News 5>]>
>>> News.objects.filter(title__contains='Формы')
<QuerySet [<News: Новость из Формы 3>]>
>>> News.objects.filter(title__icontains='Формы')
<QuerySet [<News: Новость из Формы 3>]>
>>> News.objects.filter(title__icontains='формы')
<QuerySet [<News: Новость из формы 2>, <News: Новость из формы>]>
```

Рисунок 8

News.objects.filter(pk__in=[значение1, значение2, значение3]) – находит записи, чьи поля соответствуют одному из указанных значений. (рисунок 9)

```
>>> News.objects.filter(pk__in=[9, 10, 11])
<QuerySet [<News: Новость из Формы 3>, <News: Новость из формы 2>, <News: Спорт>]>
```

Рисунок 9

Мы можем объединять несколько операторов фильтрации, однако обратите внимание, что если вы используете перечисление, то все условия будут связаны логическим И. (Рисунок 10)

```
>>> News.objects.filter(pk__in=[9, 10, 11], title__contains='формы')
<QuerySet [<News: Новость из формы 2>]>
```

Рисунок 10

Также обратите внимание, что в качестве параметра при фильтрации может быть передан Queryset. Например, могут отбираться новости не по одной категории, а по нескольким. (рисунок 11)

```
>>> cats = Category.objects.filter(pk__in=[1, 3])
>>> cats
<QuerySet [<Category: Культура>, <Category: Политика>]>
>>> News.objects.filter(category__in=cats)
<QuerySet [<News: Новость из формы>, <News: Новость из админки>, <News: Новость 4>, <News: Новость 2>]>
```

Рисунок 11

Методы

Методы, возвращающие первый/последний элемент выборки.

News.first()

News.last()

Преимущество данных методов заключается в том, что они могут выбирать первый/последний элемент из списка, к которому был применены различные фильтры. (рисунок 12)

Обратите внимание, мы устанавливали через атрибут модели, как должны быть отфильтрованы объекты(от последней добавленной к первой добавленной), поэтому, если применить метод first() к News.objects.all(), то получим последнюю добавленную новость.

```
>>> cat2 = Category.objects.get(pk=2)
>>> news = cat2.news_set.all()            )
>>> news
<QuerySet [<News: Новость из формы 2>, <News: Спорт>, <News: Новость 3>]>
>>> news = cat2.news_set.filter(pk__gt=3)
>>> news
<QuerySet [<News: Новость из формы 2>, <News: Спорт>]>
>>> news = cat2.news_set.filter(pk__gt=3).first()
>>> news
<News: Новость из формы 2>
```

Рисунок 12

Методы earliest() и latest(), возвращают самый ранний/поздний элемент по полю типа datatime. Применим эти методы с параметром **updated_at**. (рисунок 13, рисунок 14)

```
>>> News.objects.earliest('updated_at')
<News: Новость из админки>
>>> News.objects.latest('updated_at')
<News: Новость 3>
```

Рисунок 13

	id	title	content	created at	updated at
1	1	Новость 1	Django ...	2023-06-21 19:26:15.935424	2023-06-23 10:53:28.734898
2	2	Новость 2	Контент новости 2	2023-06-21 19:37:37.477978	2023-06-23 08:06:03.486257
3	3	Новость 3	Контент новости 3 :)	2023-06-21 19:40:52.550142	2023-07-04 12:31:25.778043
4	4	Новость 4	Контент новости 4	2023-06-21 19:43:09.637843	2023-06-23 15:11:19.313961
5	5	News 5	Content about news 5 :-)	2023-06-22 10:07:46.677974	2023-06-23 08:05:32.327772
6	7	Новость из админки	Какой-то контент новости	2023-06-22 14:51:29.420335	2023-06-23 08:05:20.336365
7	8	Новость из формы	Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore ...	2023-06-28 20:33:38.941696	2023-06-30 16:27:57.942211
8	9	Спорт	Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore ...	2023-06-28 21:07:05.268785	2023-06-30 16:27:57.938245
9	10	Новость из формы 2	Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore ...	2023-07-01 10:45:55.339460	2023-07-01 10:45:55.339460
10	11	Новость из Формы 3	Odio eu feugiat pretium nibh ipsum consequat nisl vel pretium. Ut tortor pretium viverra suspendisse potenti ...	2023-07-04 11:44:30.351386	2023-07-04 11:44:30.351386

Рисунок 14

Метод **exists()** возвращает значение Boolean, показывающее, содержит ли выборка элементы, а метод **count()** возвращает количество объектов. Эти методы будут очень полезны нам в будущем, запомните их, они очень быстрые и могут использоваться для проверок. Создадим новую категорию. (рисунок 15)

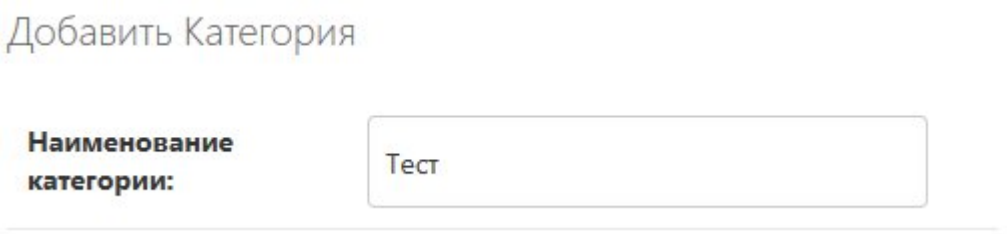


Рисунок 15

Проверим, существуют ли новости с данной категорией, вернем их количество. (рисунок 16)

```
>>> cat1 = Category.objects.get(pk=1)
>>> cat5 = Category.objects.get(pk=5)
>>> cat1.news_set.exists()
True
>>> cat5.news_set.exists()
False
>>> cat1.news_set.count()
2
>>> cat5.news_set.count()
0
>>> News.objects.count()
10
```

Рисунок 16

Достаточно интересные методы это get_previous_by_foo и get_next_by_foo. Они доступны только в том случае, если в таблице есть поле типа datatime. Методы возвращают предыдущую/последующую запись от текущей по указанному признаку. (рисунок 17)

<название переменной>.get_previous_by_created_at

<название переменной>.get_next_by_created_at

```
>>> news = News.objects.get(pk=5)
>>> news
<News: News 5>
>>> news.get_previous_by_created_at()
<News: Новость 4>
>>> news.get_next_by_created_at()
<News: Новость из админки>
```

Рисунок 17

Не забывайте, что методы и фильтры можно комбинировать. (рисунок 18)

```
>>> news.get_next_by_created_at(pk__gt=8)
<News: Спорт>
>>> news.get_next_by_created_at(pk__gt=8, title__contains='Форм')
<News: Новость из Формы 3>
```

Рисунок 18

Класс Q. Фильтрация по значениям полей связанных записей

Для того, чтобы получить данные вторичной модели по значению поля первичной модели, мы должны построить свой запрос следующим образом:

<имя поля внешнего ключа>__<имя поля первичной модели>

Получим новости, относящейся к определенной категории, используя не ее первичный ключ, а поле title. (рисунок 19)

News.objects.filter(category__title='Политика')

```
>>> News.objects.filter(category__title='Политика')
<QuerySet [<News: Новость 4>, <News: Новость 2>]>
```

Рисунок 19

Для того, чтобы решить обратную задачу нам необходимо построить свой запрос следующим образом: <имя вторичной модели в нижнем регистре>__<название поля первичной модели>__фильтр. Например, нам необходимо вывести категорию каждой новости, советующей заданному условию. (рисунок 20)

```
>>> Category.objects.filter(news__title__contains='формы')
<QuerySet [<Category: Культура>, <Category: Спорт>]>
>>> Category.objects.filter(news__title__contains='формы').count()
2
```

Рисунок 20

В нашем случае получилось 2 новости с разными категориями. Однако, если бы новостей было больше, то названия новостей дублировались бы. Например, <QuerySet [<Category: Культура>, <Category: Спорт>, <Category: Спорт>, <Category: Культура>] >.

Для того, чтобы этого избежать и выводить название категории только 1 раз, т.е выводить уникальные значения, можно использовать метод distinct(). (рисунок 21)

```
>>> Category.objects.filter(news__title__contains='формы').distinct()
<QuerySet [<Category: Культура>, <Category: Спорт>]>
```

Рисунок 21

Теперь вспомним, как мы применяла несколько фильтров к данным, при использовании перечисления между условиями по умолчанию устанавливалось логическое И. Однако нам также могут понадобиться логические ИЛИ и НЕ. Для решения этой задачи необходимо использовать класс **Q**. Теперь заключив каждое условие в скобки с Q в начале мы можем устанавливать между ними **&**(И), |(ИЛИ), ~(НЕ). (рисунок 22)

```
>>> from django.db.models import Q
>>> News.objects.filter(pk__in=[5, 7, 10], title__contains='2')
<QuerySet [<News: Новость из формы 2>]>
>>> News.objects.filter(pk__in=[5, 7], title__contains='2')
<QuerySet []>
>>> News.objects.filter(Q(pk__in=[5, 7]) | Q(title__contains='2'))
<QuerySet [<News: Новость из формы 2>, <News: Новость из админки>, <News: News 5>, <News: Новость 2>]>
>>> News.objects.filter(Q(pk__in=[5, 7]) | Q(title__contains='2') & ~ Q(pk__gt=4))
<QuerySet [<News: Новость из админки>, <News: News 5>, <News: Новость 2>]>
```

Рисунок 22

Лимитирование количества записей. Агрегатные функции

Для того, чтобы осуществить выборку определённого количества записей можно использовать механизм срезов. Он работает также, как и в классическом Python, за исключением того, что Django не поддерживает отрицательные значения. (рисунок 23)

```
>>> News.objects.all()[:3]
<QuerySet [<News: Новость из Формы 3>, <News: Новость из формы 2>, <News: Спорт>]>
>>> News.objects.all()[7:]
<QuerySet [<News: Новость 3>, <News: Новость 2>, <News: Новость 1>]>
>>> News.objects.all()[3:7]
<QuerySet [<News: Новость из формы>, <News: Новость из админки>, <News: News 5>, <News: Новость 4>]>
```

Рисунок 23

Агрегатная функция выполняет вычисление над набором значений и возвращает одно значение. В табличной модели данных это значит, что функция берет ноль, одну или несколько строк для какой-то колонки и возвращает единственное значение.

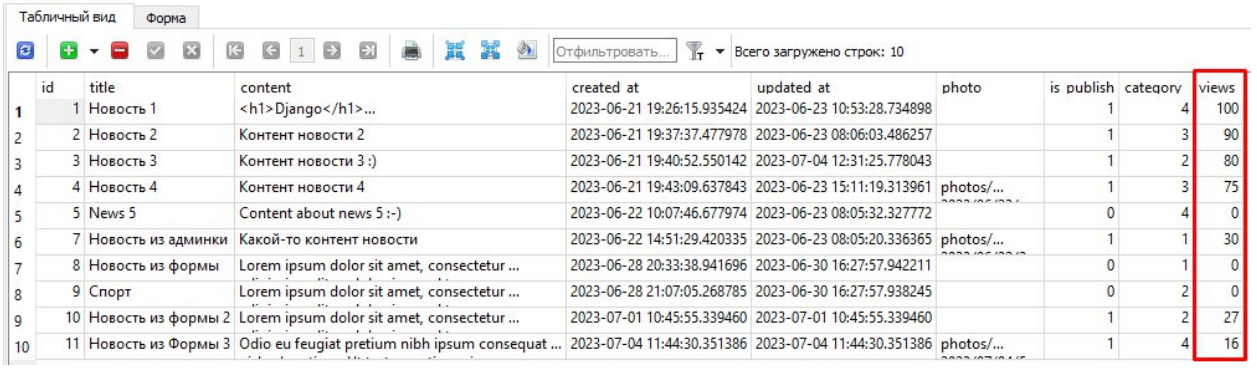
Документация:
https://django.fun/ru/docs/django/4.1/ref/models/querysets/#id6

Для демонстрации работы с агрегатными функциями нам понадобиться добавить еще одно поле к модели News – views, показывающее количество просмотров. (рисунок 24)

```
class News(models.Model):
    title = models.CharField(max_length=150, verbose_name='Наименование')
    content = models.TextField(blank=True, verbose_name='Контент')
    created_at = models.DateTimeField(auto_now_add=True, verbose_name='Дата публикации')
    updated_at = models.DateTimeField(auto_now=True, verbose_name='Обновлено')
    photo = models.ImageField(upload_to='photos/%Y/%m/%d', verbose_name='Фото', blank=True)
    is_published = models.BooleanField(default=True, verbose_name='Опубликовано')
    category = models.ForeignKey('Category', on_delete=models.PROTECT, null=True, verbose_name='Категория')
    views = models.IntegerField(default=0)
```

Рисунок 24

Не забудьте провести миграции для применения изменений. Далее заполните новую колонку произвольными значениями. (рисунок 25)



Табличный вид | Форма

Отфильтровать... | Всего загружено строк: 10

	id	title	content	created at	updated at	photo	is publish	category	views
1	1	Новость 1	Django ...	2023-06-21 19:26:15.935424	2023-06-23 10:53:28.734898		1	4	100
2	2	Новость 2	Контент новости 2	2023-06-21 19:37:37.477978	2023-06-23 08:06:03.486257		1	3	90
3	3	Новость 3	Контент новости 3 :)	2023-06-21 19:40:52.550142	2023-07-04 12:31:25.778043		1	2	80
4	4	Новость 4	Контент новости 4	2023-06-21 19:43:09.637843	2023-06-23 15:11:19.313961	photos/...	1	3	75
5	5	News 5	Content about news 5 :-)	2023-06-22 10:07:46.677974	2023-06-23 08:05:32.327772		0	4	0
6	7	Новость из админки	Какой-то контент новости	2023-06-22 14:51:29.420335	2023-06-23 08:05:20.336365	photos/...	1	1	30
7	8	Новость из формы	Lorem ipsum dolor sit amet, consectetur ...	2023-06-28 20:33:38.941696	2023-06-30 16:27:57.942211		0	1	0
8	9	Спорт	Lorem ipsum dolor sit amet, consectetur ...	2023-06-28 21:07:05.268785	2023-06-30 16:27:57.938245		0	2	0
9	10	Новость из формы 2	Lorem ipsum dolor sit amet, consectetur ...	2023-07-01 10:45:55.339460	2023-07-01 10:45:55.339460		1	2	27
10	11	Новость из Формы 3	Odio eu feugiat pretium nibh ipsum consequat ...	2023-07-04 11:44:30.351386	2023-07-04 11:44:30.351386	photos/...	1	4	16

Рисунок 25

Импортируйте агрегатные функции из модуля **django.db.models** в консоли. Далее рассмотрим некоторые из них: Min, Max, Sum, Avg(среднее арифметическое), Count(количество записей). Обратите внимание, что над ними можно проводить дополнительные вычисления. (рисунок 26, рисунок 27)

```
>>> News.objects.aggregate(Sum('views'))
{'views__sum': 418}
>>> News.objects.aggregate(Avg('views'))
{'views__avg': 41.8}
>>> News.objects.aggregate(Count('views'))
{'views__count': 10}
```

Рисунок 26

```
>>> from django.db.models import *
>>> News.objects.aggregate(Min('views'), Max('views'))
{'views__min': 0, 'views__max': 100}
>>> News.objects.aggregate(min_views=Min('views'), max_views=Max('views'))
{'min_views': 0, 'max_views': 100}
>>> News.objects.aggregate(diff=Min('views')-Max('views'))
{'diff': -100}
```

Рисунок 27

Метод annotate()

Документация:

https://django.fun/ru/docs/django/4.1/ref/models/querysets/#annotate

Используя более простое объяснение, можно сказать, что метод позволяет производить агрегатные вычисления по группам записей. Более подробно рассмотрим на практических примерах.

Применим выражение Count к каждой категории, чтобы получить количество новостей в них. (рисунок 28)

```
>>> cats = Category.objects.annotate(Count('news'))
>>> cats
<QuerySet [<Category: Культура>, <Category: Спорт>, <Category: Политика>, <Category: Наука>, <Category: Тест>]>
```

Рисунок 28

Выведем название каждой категории и результат работы агрегатной функции. По умолчанию он будет записан в атрибут news__count, однако мы можем изменить его название, используя позиционный аргумент. (рисунок 29)

```
>>> cats = Category.objects.annotate(Count('news'))
>>> for item in cats:
...     print(item.title, item.news__count)
...
Культура 2
Спорт 3
Политика 2
Наука 3
Тест 0
>>> cats = Category.objects.annotate(cnt=Count('news'))
>>> for item in cats:
...     print(item.title, item.cnt)
...
Культура 2
Спорт 3
Политика 2
Наука 3
Тест 0
```

Рисунок 29

Аналогичным образом можно применять и другие агрегатные вычисления. (рисунок 30)

```
>>> cats = Category.objects.annotate(max_views=Max('news__views'))
>>> for item in cats:
...     print(item.title, item.max_views)
...
Культура 30
Спорт 80
Политика 90
Наука 100
Тест None
>>> cats = Category.objects.annotate(sum_views=Sum('news__views'))
>>> for item in cats:
...     print(item.title, item.sum_views)
...
Культура 30
Спорт 107
Политика 165
Наука 116
Тест None
```

Рисунок 30

Обратите внимание, что для категории Тест выводится None. Будет логично скрывать категорию, не содержащую новостей. Для этого необходимо применить фильтр. (рисунок 31)

```
>>> cats = Category.objects.annotate(cnt=Count('news')).filter(cnt__gt=0)
>>> for item in cats:
...     print(item.title, item.cnt)
...
Культура 2
Спорт 3
Политика 2
Наука 3
```

Рисунок 31

Давайте теперь используем полученные знания, чтобы скрыть от пользователя категории, не содержащие новостей, тем самым исключив переход на пустые или несуществующие страницы.

Сначала необходимо в методе **show_categories()**, находящемся в файле **news_tags.py**, изменить получение всех записей на получение релевантных записей. (рисунок 32)

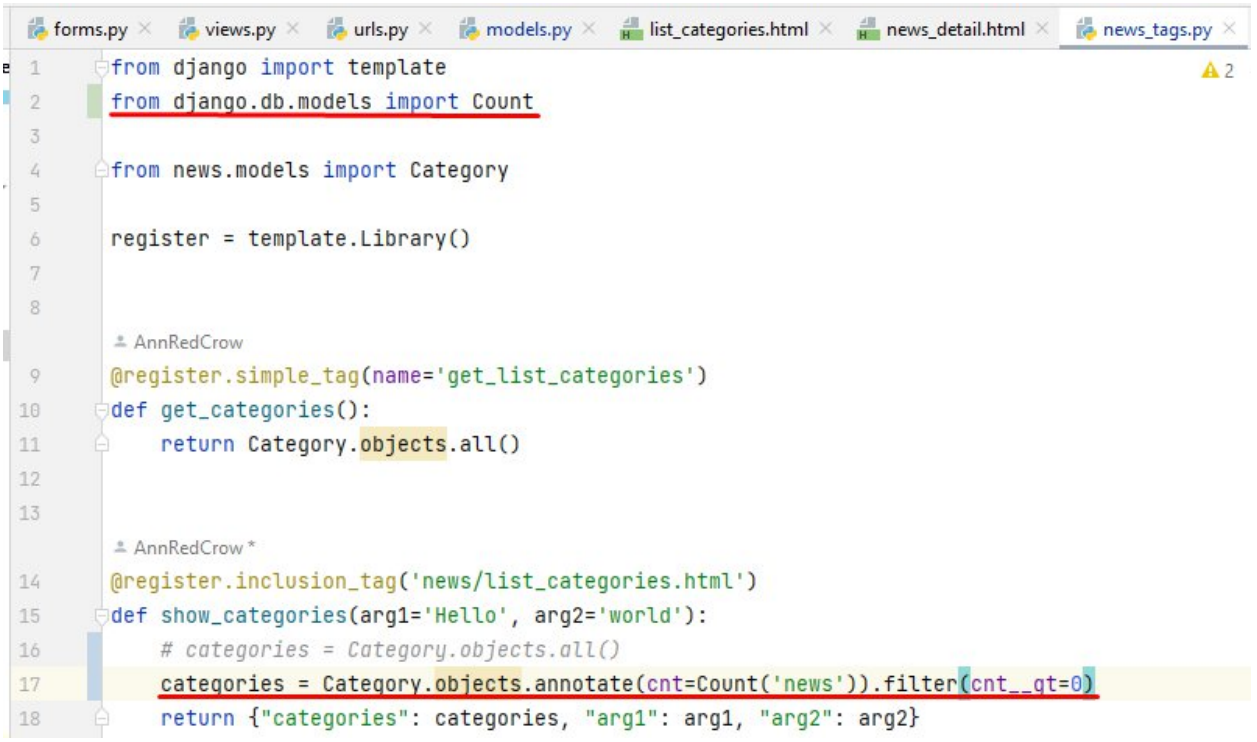


Рисунок 32

Теперь, обновив страницу браузера, вы увидите только категории, содержащие новости.

Далее давайте добавим вывод количества новостей рядом с названием категории. Скопируйте код (отображение списка со значками):

14

(https://getbootstrap.com/docs/5.0/components/list-group/#with-badges)

Поместите его в ссылку шаблона list_categories.html, замените **14** на **{{item.cnt}}**. Обратите внимание, мы сами назвали атрибут **cnt**, если вы не использовали позиционный аргумент, то вам необходимо писать имя по умолчанию. (рисунок 33)

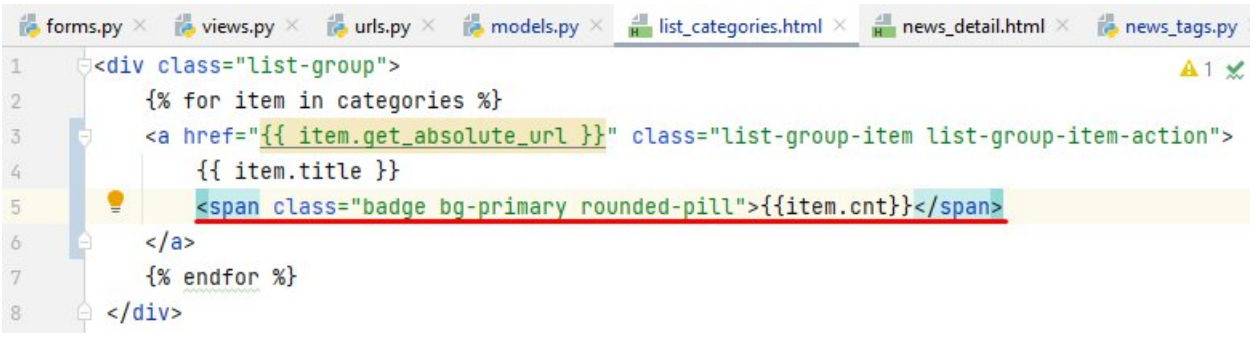


Рисунок 33

В результате страница пользователя будет выглядеть следующим образом (рисунок 34):

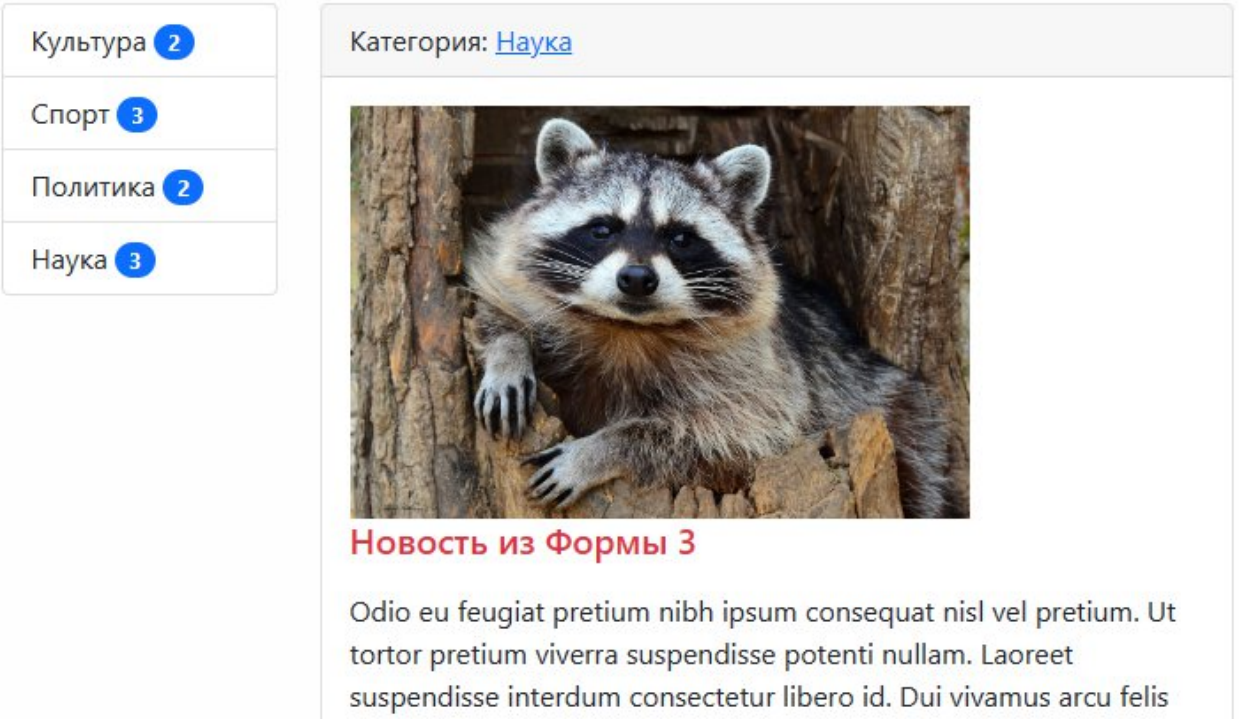


Рисунок 34

Небольшое напоминание, если вам необходимо считать записи с уникальными полями, то установите параметр distinct в True. (рисунок 35)

```
>>> News.objects.aggregate(cnt=Count('views'))
{'cnt': 10}
>>> News.objects.aggregate(cnt=Count('views', distinct=True))
{'cnt': 8}
```

Рисунок 35

Метод values()

Этот метод возвращает QuerySet, который возвращает словари, а не экземпляры модели, когда используется как итеративный. Каждый из этих словарей представляет объект с ключами, соответствующими именам атрибутов объектов модели.

Документация:
https://django.fun/ru/docs/django/4.1/ref/models/querysets/#values

Говоря простыми словами, данный метод позволяет получить не все поля объекта, а только указанные. Он будет эффективен при работе с

реальными базами данных, где число полей может быть большим, позволит избежать излишней перегрузки памяти.

Также давайте изучим, какие SQL-запросы будут применены. Импортируйте из django.db метод connection для просмотра и метод reset_queries для очистки списка. Примените методы, убедитесь, что список стал пустым. (рисунок 36, рисунок 37)

```
>>> reset_queries()
>>> connection.queries
[]
>>> news = News.objects.values('title', 'views', 'category__title')
>>> connection.queries
[]
```

Рисунок 36

```
>>> from django.db import connection
>>> news = News.objects.values('title', 'views', 'category__title')
>>> connection.queries
[{'sql': 'SELECT "news_category"."id", "news_category"."title", COUNT("news_news"."id") AS "news__count" FROM
 "news_category" LEFT OUTER JOIN "news_news" ON ("news_category"."id" = "news_news"."category_id") GROUP BY "
news_category"."id", "news_category"."title"', 'time': '0.000'}, {'sql': 'SELECT "news_category"."id", "news_
category"."title", COUNT("news_news"."id") AS "cnt" FROM "news_category" LEFT OUTER JOIN "news_news" ON ("new
s_category"."id" = "news_news"."category_id") GROUP BY "news_category"."id", "news_category"."title"', 'time'
: '0.000'}, {'sql': 'SELECT "news_category"."id", "news_category"."title", MAX("news_news"."views") AS "max_v
iews" FROM "news_category" LEFT OUTER JOIN "news_news" ON ("news_category"."id" = "news_news"."category_id")
GROUP BY "news_category"."id", "news_category"."title"', 'time': '0.000'}, {'sql': 'SELECT "news_category"."i
d", "news_category"."title", SUM("news_news"."views") AS "sum_views" FROM "news_category" LEFT OUTER JOIN "ne
ws_news" ON ("news_category"."id" = "news_news"."category_id") GROUP BY "news_category"."id", "news_category"
."title"', 'time': '0.000'}, {'sql': 'SELECT "news_category"."id", "news_category"."title", COUNT("news_news"
."id") AS "cnt" FROM "news_category" LEFT OUTER JOIN "news_news" ON ("news_category"."id" = "news_news"."cate
gory_id") GROUP BY "news_category"."id", "news_category"."title" HAVING COUNT("news_news"."id") > 0', 'time':
```

Рисунок 37

Обратите внимание, вы выполнили запрос, выбрали нужные поля, но при этом список остался пустым. Напоминаем, так происходит потому, что запросы в Django «ленивые», они не выполняются до тех пор, пока мы не запрашиваем данные. Давайте распечатаем элементы переменной news. Не забывайте, что метод values() возвращает **словари**. (рисунок 38)

```
>>> reset_queries()
>>> connection.queries
[]
>>> news = News.objects.values('title', 'views', 'category__title')
>>> connection.queries
[]
>>> for item in news:
...     print(item['title'], item['category__title'])
...
Новость из Формы 3 Наука
Новость из формы 2 Спорт
Спорт Спорт
Новость из формы Культура
Новость из админки Культура
News 5 Наука
Новость 4 Политика
Новость 3 Спорт
Новость 2 Политика
Новость 1 Наука
>>> connection.queries                          '])
[{'sql': 'SELECT "news_news"."title", "news_news"."views", "news_category"."title" FROM "news_news" LEFT OUTER JOIN "news_category" ON (
"news_news"."category_id" = "news_category"."id") ORDER BY "news_news"."created_at" DESC', 'time': '0.000'}]
```

Рисунок 38

Обратите внимание, что в результате был выполнен 1 оптимальный запрос, никаких дополнительный нагрузок на базу данных оказано не было.

Класс F

Объект **F()** представляет значение поля модели, преобразованное значение поля модели или аннотированный столбец. Он позволяет ссылаться на значения полей модели и выполнять операции с базой данных, используя их без необходимости извлекать их из базы данных в память Python.

Вместо этого Django использует объект **F()** для генерации SQL-выражения, которое описывает требуемую операцию на уровне базы данных.

Применим этот класс для добавления единицы к существующему значению views при просмотре новости. (рисунок 39) Конечно, мы могли бы реализовать это и без класса F, записав объект в переменную, внеся изменения и сохранив, но при этом мы сделаем наше решение уязвимым при образовании коллизий.

```
>>> from django.db.models import F
>>> news1 = News.objects.get(pk=1)
>>> news1.views
100
>>> news.views = F('views') + 1
>>> news1.views = F('views') + 1
>>> news1.views
<CombinedExpression: F(views) + Value(1)>
>>> news1.save()
```

Рисунок 39

Убедимся, что значение в БД изменилось. (рисунок 40)

	id	title	content	created at	updated at	photo	is publish	category	view
1	1	Новость 1	Django ...	2023-06-21 19:26:15.935424	2023-07-05 11:52:35.379867		1	4	
2	2	Новость 2	Контент новости 2	2023-06-21 19:37:37.477978	2023-06-23 08:06:03.486257		1	3	
3	3	Новость 3	Контент новости 3 :)	2023-06-21 19:40:52.550142	2023-07-04 12:31:25.778043		1	2	

Рисунок 40

Еще один пример, где мы выбираем новость, чей заголовок содержится в контентной части. (рисунок 41)

```
>>> News.objects.filter(content__icontains=F('title'))
<QuerySet [<News: News 5>]>
```

Рисунок 41

SQL-Функции

Это не функции, которые выполняет сам Django, а «обертки» над функциями, которые будут выполняться СУБД.

Документация: https://django.fun/ru/docs/django/4.1/ref/models/database-functions/

Мы не будем детально рассматривать эту тему в рамках лабораторной работы, всю необходимую информация вы сможете найти в документации.

Для использования функций необходимо импортировать их из модуля **django.db.models.functions**.

Применим класс **Length**. (рисунок 42)

Он принимает одно текстовое поле или выражение и возвращает количество символов в значении. Если выражение равно null, то длина также будет равна null.

```
>>> from django.db.models.functions import *
>>> news = News.objects.annotate(length=Length('title')).all()
>>> for item in news:
...     print(item.title, item.length)
...
Новость из Формы 3 18
Новость из формы 2 18
Спорт 5
Новость из формы 16
Новость из админки 18
News 5 6
Новость 4 9
Новость 3 9
Новость 2 9
Новость 1 9
```

Рисунок 42

Мы с вами рассмотрели основные возможности Django ORM. Теперь изучим возможности фреймворка по работе с чистым SQL.

Выполнение необработанных SQL-запросов (raw SQL queries)

Документация: https://django.fun/ru/docs/django/4.1/topics/db/sql/

Метод менеджера **raw()** может использоваться для выполнения необработанных SQL-запросов, возвращающих экземпляры модели. (рисунок 43)

Этот метод принимает необработанный SQL-запрос, выполняет его и возвращает экземпляр django.db.models.query.RawQuerySet. Этот экземпляр **RawQuerySet** можно перебирать как обычный QuerySet для предоставления экземпляров объектов. (рисунок 44)

```
>>> News.objects.raw("SELECT * FROM news_news")
<RawQuerySet: SELECT * FROM news_news>
>>> news = _
```

Рисунок 43

```
>>> for item in news:
...     print(item.title, item.pk, item.is_published)
...
Новость 1 1 True
Новость 2 2 True
Новость 3 3 True
Новость 4 4 True
News 5 5 False
Новость из админки 7 True
Новость из формы 8 False
Спорт 9 False
Новость из формы 2 10 True
Новость из Формы 3 11 True
```

Рисунок 44

Естественно, мы с вами не будем изучать язык SQL, но рассмотрим некоторые нюансы работы с ним в Django.

При выборе загружаемых полей мы всегда должны указывать поле первичного ключа, в противном случае возникнет ошибка. (рисунок 45)

Отложенная загрузка полей

Это плохая практика, она предполагает, что при выборе загружаемых полей мы какое-то из них не загружаем, но при этом впоследствии к нему обращаемся. Вывод будет получен, но при этом для каждой записи будет выполнен отдельный SQL-запрос, чтобы «достать» нужные данные. (рисунок 46)

```
>>> news = News.objects.raw("SELECT title FROM news_news")
>>> for item in news:
...     print(item.title)
...
Traceback (most recent call last):
  File "<console>", line 1, in <module>
  File "D:\django-sites\testsite\venv\Lib\site-packages\django\db\models\query.py", line 2080, in __iter__
    self._fetch_all()
  File "D:\django-sites\testsite\venv\Lib\site-packages\django\db\models\query.py", line 2067, in _fetch_all
    self._result_cache = list(self.iterator())
                         ^^^^^^^^^^^^^^^^^^^^^
  File "D:\django-sites\testsite\venv\Lib\site-packages\django\db\models\query.py", line 2094, in iterator
    yield from RawModelIterable(self)
  File "D:\django-sites\testsite\venv\Lib\site-packages\django\db\models\query.py", line 168, in __iter__
    raise exceptions.FieldDoesNotExist(
django.core.exceptions.FieldDoesNotExist: Raw query must include the primary key
```

Рисунок 45

```
>>> news = News.objects.raw("SELECT id, title FROM news_news")
>>> for item in news:
...     print(item.title, item.is_published)
...
Новость 1 True
Новость 2 True
Новость 3 True
Новость 4 True
News 5 False
Новость из админки True
Новость из формы False
Спорт False
Новость из формы 2 True
Новость из Формы 3 True
```

Рисунок 46

Передача параметров в raw()

Обратите внимание на рисунок 47, на нем представлена плохая практика написания запросов. Если вы будете указывать условия таким образом, то сделаете ваше приложение уязвимым к атакам через SQL-инъекции.

```
>>> news = News.objects.raw("SELECT * FROM news_news WHERE title='News 5'")
>>> news
<RawQuerySet: SELECT * FROM news_news WHERE title='News 5'>
>>> for item in news:
...     print(item.title)
...
News 5
```

Рисунок 47

Если вам нужно выполнять параметризованные запросы, вы можете использовать аргумент **params.** (рисунок 48)

```
>>> news = News.objects.raw("SELECT * FROM news_news WHERE title = %s", ['News 5'])
>>> for item in news:
...     print(item.title)
...
News 5
```

Рисунок 48

Индивидуальные задания

БЛОК 1: БАЗОВЫЙ УРОВЕНЬ

Требования: каждый вариант включает 5 заданий. Все запросы выполняются в Django shell (python manage.py shell). Для каждого задания необходимо:

1. Написать код на Django ORM
2. Показать эквивалентный SQL-запрос
3. Продемонстрировать результат выполнения

Вариант 1

	Задание
	Получить все опубликованные новости (is_published=True)
	Получить новость с заголовком «Моя первая новость» (используя get)
	Отсортировать все новости по дате создания (от новых к старым)
	Получить первые 5 новостей с наибольшим количеством просмотров (views)
	Посчитать общее количество новостей в базе данных

Вариант 2

	Задание
	Получить все категории, у которых есть описание (поле description не пустое)
	Найти категорию с названием «Политика»
	Отсортировать категории по названию в алфавитном порядке
	Получить все новости, относящиеся к категории с id=3

	Задание
	Посчитать количество категорий в базе данных

Вариант 3

	Задание
	Получить все новости, созданные после 1 января 2024 года
	Найти новость с id=5, используя get
	Отсортировать новости по количеству просмотров (от большего к меньшему)
	Получить новости, у которых заголовок содержит слово «важно»
	Посчитать количество опубликованных новостей (is_published=True)

Вариант 4

	Задание
	Получить все комментарии к новости с id=2
	Найти комментарий автора «Иван Петров»
	Отсортировать комментарии по дате создания (от старых к новым)
	Получить последние 10 комментариев ко всем новостям
	Посчитать общее количество комментариев в базе данных

Вариант 5

	Задание
	Получить все новости, у которых есть фото (поле photo не пустое)
	Найти категорию с slug «sport»
	Отсортировать новости по заголовку в алфавитном порядке
	Получить новости, у которых количество просмотров больше 100
	Посчитать среднее количество просмотров всех новостей

Вариант 6

	Задание
	Получить все комментарии с рейтингом 5
	Найти новость, относящуюся к категории «Культура»
	Отсортировать новости по дате обновления (updated_at)
	Получить категории, у которых есть хотя бы одна новость
	Посчитать общее количество просмотров всех новостей (сумму)

Вариант 7

	Задание
	Получить все новости, созданные в декабре 2024 года
	Найти комментарий с текстом «Отличная статья!»
	Отсортировать категории по дате создания (от новых к старым)
	Получить новости, у которых количество просмотров между 50 и 200

	Задание
	Посчитать количество комментариев с рейтингом ниже 3

Вариант 8

	Задание
	Получить все непроверенные комментарии (is_moderated=False)
	Найти категорию, у которой больше всего новостей
	Отсортировать новости по заголовку в обратном алфавитном порядке
	Получить первые 3 новости из категории «Технологии»
	Посчитать средний рейтинг комментариев для новости с id=1

Вариант 9

	Задание
	Получить все новости, у которых заголовок начинается с буквы «П»
	Найти комментарий, оставленный пользователем с email «user@example.com»
	Отсортировать комментарии по рейтингу (от большего к меньшему)
	Получить новости, у которых содержание содержит слово «Django»
	Посчитать количество новостей в каждой категории

Вариант 10

	Задание
	Получить все новости, которые были обновлены за последние 7 дней
	Найти категорию с названием «Спорт» и вывести все её новости
	Отсортировать новости по количеству комментариев (от большего к меньшему)
	Получить новости, у которых есть хотя бы один комментарий
	Посчитать количество категорий, в которых есть опубликованные новости

Вариант 11

	Задание
	Получить все комментарии, оставленные за последний месяц
	Найти новость с наибольшим количеством просмотров
	Отсортировать категории по количеству новостей в них
	Получить все новости, у которых дата создания и дата обновления совпадают
	Посчитать количество уникальных авторов комментариев

Вариант 12

	Задание
	Получить все новости, у которых заголовок длиннее 50 символов

	Задание
	Найти категорию, которая не содержит ни одной новости
	Отсортировать комментарии по длине текста (от коротких к длинным)
	Получить новости, у которых количество просмотров меньше средней величины
	Посчитать максимальный рейтинг комментариев для каждой новости

Вариант 13

	Задание
	Получить все новости, отсортированные по категории, а внутри категории по дате
	Найти комментарий с самым высоким рейтингом
	Отсортировать категории по названию в обратном порядке
	Получить новости, у которых содержание не пустое
	Посчитать количество комментариев, оставленных каждым автором

Вариант 14

	Задание
	Получить все новости, у которых заголовок заканчивается на «!»
	Найти категорию, у которой slug = «health»
	Отсортировать новости по дате создания, а затем по заголовку

	Задание
	Получить новости, у которых количество просмотров равно 0
	Посчитать минимальный рейтинг комментариев для каждой новости

Вариант 15

	Задание
	Получить все новости, у которых дата создания — сегодня
	Найти комментарий автора «Admin»
	Отсортировать категории по дате создания (от старых к новым)
	Получить новости, у которых заголовок не содержит слово «секретно»
	Посчитать среднюю длину текста комментария для каждой новости

БЛОК 2: СРЕДНИЙ УРОВЕНЬ

Требования: использовать связи между моделями (ForeignKey), применять filter, exclude, Q, annotate, aggregate, select_related.

Вариант 1

	Задание
	Получить все новости категории «Политика» с количеством комментариев под каждой
	Найти категорию, в которой больше всего новостей (используя annotate и Count)

	Задание
	Получить топ-5 новостей по количеству комментариев
	Найти новости, у которых заголовок содержит «выборы» ИЛИ количество просмотров > 500
	Подсчитать, сколько новостей в каждой категории, и вывести только те категории, где новостей больше 2

Вариант 2

	Задание
	Получить все комментарии к новостям категории «Спорт» с указанием автора новости
	Найти автора (email), который оставил больше всего комментариев
	Получить новости, у которых средний рейтинг комментариев выше 4
	Найти категории, в которых есть хотя бы одна новость без комментариев
	Подсчитать общее количество комментариев, оставленных за последнюю неделю

Вариант 3

	Задание
	Получить все новости с их категориями, используя select_related
	Найти пользователя (author), который оставил комментарий с самым высоким рейтингом
	Получить новости, у которых количество просмотров больше среднего

	Задание
	по всем новостям
	Найти категории, где суммарное количество просмотров новостей превышает 1000
	Подсчитать среднее количество символов в комментариях для каждой новости

Вариант 4

	Задание
	Получить все новости, у которых есть хотя бы один комментарий с рейтингом 1
	Найти категорию с максимальным суммарным количеством просмотров
	Получить комментарии к новостям, опубликованным в декабре 2024 года
	Используя Q, найти новости, у которых заголовок содержит «важно» ИЛИ количество просмотров меньше 50
	Подсчитать для каждого автора комментариев: сколько комментариев он оставил и их средний рейтинг

Вариант 5

	Задание
	Получить все новости, у которых количество комментариев больше количества просмотров
	Найти новость с самой большой разницей между количеством

	Задание
	комментариев и просмотров
	Получить список категорий с количеством новостей и общим количеством просмотров
	Найти комментарии, где рейтинг ниже среднего по данной новости
	Подсчитать количество новостей, у которых есть фото и которые опубликованы

Вариант 6

	Задание
	Получить все новости, у которых нет ни одного комментария
	Найти автора комментария, который оставил самый длинный текст
	Получить топ-3 категории по суммарному рейтингу комментариев
	Используя F, увеличить количество просмотров всех новостей категории «Популярное» на 10
	Подсчитать для каждой новости: количество комментариев с рейтингом 5

Вариант 7

	Задание
	Получить все новости, у которых дата создания позже, чем дата создания категории
	Найти категорию, в которой средний рейтинг комментариев максимален

	Задание
	Получить комментарии, оставленные авторами, у которых в email есть «gmail.com»
	Найти новости, у которых количество просмотров увеличилось более чем на 50% за последнюю неделю
	Подсчитать количество уникальных авторов комментариев для каждой категории

Вариант 8

	Задание
	Получить все новости, у которых заголовок и содержание содержат одно и то же слово
	Найти комментарий, который был оставлен первым к своей новости
	Получить список новостей, отсортированных по дате последнего комментария
	Используя Q и ~ (НЕ), найти новости, у которых заголовок не содержит «секретно» и которые опубликованы
	Подсчитать для каждой категории: общее количество просмотров и средний рейтинг комментариев

Вариант 9

	Задание
	Получить все новости, у которых есть хотя бы один комментарий от автора, совпадающего с автором новости (если бы поле author было)

	Задание
	Найти категорию, в которой отношение комментариев к просмотрам максимально
	Получить новости, у которых количество просмотров в 2 раза больше среднего
	Найти авторов комментариев, которые оставляли комментарии к новостям из разных категорий
	Подсчитать медианный рейтинг комментариев для каждой новости (используя сложный запрос)

Вариант 10

	Задание
	Получить все новости, у которых есть комментарии с рейтингом выше, чем средний рейтинг всех комментариев
	Найти новость, у которой самый высокий процент комментариев с рейтингом 5
	Получить список категорий, в которых все новости имеют фото
	Используя F, уменьшить количество просмотров старых новостей (старше 1 года) на 50%
	Подсчитать коэффициент активности категории: (количество комментариев / количество просмотров)

Вариант 11

	Задание
	Получить все новости, у которых количество комментариев растёт (каждый следующий комментарий имеет более высокий рейтинг)
	Найти автора, который оставил комментарии к наибольшему количеству различных новостей
	Получить новости, у которых все комментарии имеют рейтинг не ниже 3
	Найти категории, где есть новости без фото, но с количеством просмотров > 100
	Подсчитать для каждой новости: разницу между максимальным и минимальным рейтингом комментариев

Вариант 12

	Задание
	Получить все новости, у которых последний комментарий был оставлен более 7 дней назад
	Найти автора комментария, чьи комментарии в среднем имеют самый высокий рейтинг
	Получить список категорий, отсортированных по дате последней новости в категории
	Используя Q, найти новости, у которых (заголовок содержит «срочно» И просмотры > 100) ИЛИ (категория = «Политика»)
	Подсчитать кумулятивное количество просмотров по дням для каждой новости

Вариант 13

	Задание
	Получить все новости, у которых количество комментариев превышает количество комментариев в среднем по категории
	Найти категорию, в которой новости набирают просмотры быстрее всего (по дате создания и просмотрам)
	Получить комментарии, которые являются ответами на другие комментарии (если добавить поле parent в Comment)
	Найти авторов, которые оставили комментарии к новостям из всех категорий
	Подсчитать для каждого часа суток: сколько комментариев было оставлено

Вариант 14

	Задание
	Получить все новости, у которых количество просмотров в выходные дни выше, чем в будни
	Найти новость с наибольшим количеством комментариев, оставленных в первый час после публикации
	Получить список авторов, которые никогда не оставляли комментарии с рейтингом ниже 3
	Используя F, добавить бонусные просмотры новостям, у которых отношение комментариев к просмотрам > 0.1
	Подсчитать для каждой категории: тренд просмотров (возрастает/убывает) за последние 30 дней

Вариант 15

	Задание
	Получить все новости, у которых количество комментариев больше медианы по всем новостям
	Найти пары новостей, которые часто комментируют одни и те же авторы
	Получить категории, в которых дисперсия количества просмотров минимальна
	Найти авторов, чьи комментарии получают статистически значимо более высокий рейтинг
	Подсчитать для каждой новости: прогнозируемое количество просмотров через 7 дней (на основе текущего тренда)

БЛОК 3: ПРОДВИНУТЫЙ УРОВЕНЬ

Требования: использовать сложные запросы с Q, F, annotate, subquery, Window; оптимизировать запросы через select_related, prefetch_related; работать с trunc для группировки по времени.

Вариант 1

	Задание
	Рассчитать скользящее среднее просмотров новостей по дням для каждой категории (используя Window)
	Найти новости, у которых просмотры выросли более чем на 50% по сравнению с предыдущей новостью в той же категории
	Выполнить ABC-анализ новостей по просмотрам (A — 80% просмотров, B — 15%, C — 5%)
	Оптимизировать запрос получения списка новостей с комментариями,

	Задание
	используя select_related и prefetch_related
	Найти категории, где коэффициент вариации просмотров новостей превышает 1.5

Вариант 2

	Задание
	С помощью subquery получить для каждой категории топ-3 новости по просмотрам
	Используя Count и Q, найти комментарии, которые получили лайки от авторов других комментариев
	Реализовать рекомендацию похожих новостей на основе общих тегов (если добавить модель Tag)
	Создать запрос для поиска новостей, чья динамика просмотров коррелирует с динамикой другой новости
	Оптимизировать ленту новостей с использованием prefetch_related для комментариев

Вариант 3

	Задание
	Найти аномалии: новости, у которых просмотры превышают среднее на 3 стандартных отклонения
	Используя Window, рассчитать кумулятивное количество просмотров по дням для каждой категории

	Задание
	Обнаружить ботов: IP-адреса (если добавить), с которых пришло более 10 просмотров одной новости за минуту
	Сгруппировать просмотры по дням и часам, найти часы пик для каждой категории
	Используя F(), создать обновление, которое понижает рейтинг новости при падении просмотров

Вариант 4

	Задание
	Рассчитать конверсию: из просмотра в комментарий (комментарии / просмотры) по каждой категории
	Найти среднее время между публикацией новости и первым комментарием (используя Lag)
	Определить LTV (Lifetime Value) новости: общее количество просмотров за первые 7 дней
	Реализовать прогноз: найти новости, которые не получат новых комментариев в ближайшие 3 дня
	Используя CASE и annotate, классифицировать новости по уровню вовлечённости

Вариант 5

	Задание
	Используя F(), увеличить рейтинг комментария на 1 за каждый

	Задание
	положительный ответ
	Найти цепочки комментариев (диалоги между двумя авторами) под одной новостью
	Подсчитать количество комментариев в час после публикации новости
	Реализовать поиск мультиаккаунтинга: авторов с одинаковыми email или IP
	С помощью annotate и Case классифицировать комментарии по тональности (позитивные/нейтральные/негативные)

Вариант 6

	Задание
	Используя subquery, найти для каждой новости 5 самых похожих (по категории и количеству просмотров)
	Предсказать количество просмотров новости на основе просмотров похожих новостей
	Создать персональную ленту: топ-10 новостей, которые могут понравиться пользователю (на основе его истории просмотров)
	Найти новости, чья популярность значительно изменилась за последнюю неделю
	Рассчитать для каждого пользователя коэффициент экстремальности: среднее отклонение его оценок комментариев от общего среднего

Вариант 7

	Задание
	Используя Window, рассчитать скользящие средние просмотров (MA7, MA30) для каждой новости
	Обнаружить «вирусные» новости: резкий рост просмотров в короткий промежуток времени
	Найти корреляцию между временем публикации и количеством просмотров (по часам)
	Реализовать поиск дивергенций: просмотры растут, а комментарии падают
	Используя F, рассчитать ожидаемый охват новости на основе исторических данных

Вариант 8

	Задание
	Рассчитать скорость набора просмотров (Velocity) для каждой новости за первые 24 часа
	Построить диаграмму выгорания: количество оставшихся новостей без комментариев по дням
	Найти новости, которые находятся в топе по просмотрам более 5 дней подряд
	Используя F, рассчитать эффективность категории: просмотры на одну новость
	Подсчитать cumulative flow: количество новостей в каждом статусе (is_published) на каждый день

Вариант 9

	Задание
	Найти все возможные связки новостей через общих авторов комментариев (используя JOIN таблицы Comment с самой собой)
	Рассчитать загрузку комментаторами по дням недели и категориям
	Найти оптимальное время публикации для получения максимального количества комментариев
	Используя дату и время, найти ближайшие по тематике новости для рекомендации
	Реализовать алгоритм поиска кратчайшего пути между новостями через общих читателей

Вариант 10

	Задание
	Рассчитать среднюю продолжительность «жизни» новости (время до последнего комментария)
	Используя Window, найти моменты, когда новость резко меняет темп набора просмотров
	Построить переходную матрицу: из какой категории читатели чаще переходят в какую
	Найти новости, в которых рейтинг комментариев ухудшается (снижается средний рейтинг по времени)
	Оптимизировать запрос для получения полной истории комментариев к новости

Вариант 11

	Задание
	Рассчитать коэффициент соответствия комментария новости на основе длины и рейтинга
	Используя annotate, найти авторов, у которых суммарный рейтинг комментариев превышает порог
	Реализовать рекомендацию новостей для комментатора на основе его истории комментариев
	Найти наиболее часто используемые слова в комментариях (анализ текста)
	С помощью F создать запрос, который обновляет теги новости на основе содержания комментариев

Вариант 12

	Задание
	Найти флаки-комментарии: комментарии, чей рейтинг сильно колеблется (по статистическому анализу)
	Рассчитать среднее время написания комментария для каждой новости и найти выбросы
	Используя Lag и Lead, определить комментарии, после которых рейтинг обсуждения резко меняется
	Построить матрицу корреляции между новостями: если выросла популярность новости A, с какой вероятностью вырастет у новости B
	Реализовать запрос для обнаружения комментариев, нарушающих правила (на основе стоп-слов)

Вариант 13

	Задание
	Найти новости, у которых сезонность просмотров очевидна (пик в определённые месяцы)
	Используя Extract, выделить день недели с максимальным количеством комментариев для каждой категории
	Построить прогноз просмотров на основе модели линейной регрессии (в рамках ORM)
	Найти аномальные всплески просмотров и определить их причину (по времени суток, категории)
	Реализовать поиск новостей-аутлаеров: просмотры которых не соответствуют тренду категории

Вариант 14

	Задание
	Найти новости, у которых количество просмотров в праздничные дни значимо выше обычных
	Используя TruncDay, сгруппировать просмотры и найти день с максимальным приростом
	Реализовать поиск «эффекта первой новости»: влияет ли первая новость в категории на последующие
	Найти авторов комментариев, которые переключаются между категориями с определённой периодичностью

	Задание
	Используя Case и When, присвоить новостям теги на основе их динамики (растущие, стабильные, падающие)

Вариант 15

	Задание
	Рассчитать индекс цитируемости новости: количество комментариев, ссылающихся на другие комментарии
	Найти пары новостей, чьи временные ряды просмотров имеют максимальную корреляцию
	Используя Subquery, добавить к каждой новости прогноз просмотров на основе среднего по категории
	Реализовать поиск «мёртвых душ»: авторов, чьи комментарии никогда не получают ответов
	Построить многомерную кластеризацию новостей по просмотрам, комментариям, времени жизни (в рамках ORM)

Контрольные вопросы

Базовый уровень

1. Что такое ORM и какие задачи он решает? Приведите пример из лабораторной работы, где вы использовали ORM вместо прямого SQL-запроса.

2. Как получить все записи из таблицы News? Какой SQL-запрос при этом генерируется?
3. Чем отличается метод all() от метода filter()? В каких случаях нужно использовать каждый из них?
4. Как выполнить сортировку записей по убыванию даты создания? Приведите пример кода.
5. Что произойдёт, если вызвать News.objects.get(pk=999), когда записи с таким ID не существует? Как обработать эту ситуацию?
6. Что такое атрибут _set и для чего он используется? Приведите пример получения всех новостей, относящихся к конкретной категории.
7. Чем отличается метод first() от метода earliest('created_at')? Какой сценарий использования каждого метода?
8. Что делает метод exists() и почему его использование эффективнее, чем преобразование QuerySet в список?
9. Как выполнить фильтрацию с условием «содержит подстроку» (независимо от регистра)? Приведите пример.
10. Что означают фильтры __gt, __gte, __lt, __lte? Приведите примеры использования.

Средний уровень

1. Как объединить несколько условий с логическим ИЛИ, используя класс Q? Приведите пример выборки новостей, у которых заголовок содержит «Python» ИЛИ количество просмотров больше 100.
2. Что такое «ленивые» запросы (lazy evaluation) в Django ORM? Когда реально выполняется запрос к базе данных?
3. Как получить первые 5 записей, начиная с 10-й? Какой синтаксис срезов поддерживается в QuerySet?

4. Какие агрегатные функции доступны в Django ORM? Приведите пример вычисления средней длины заголовка новостей.
5. Чем отличается метод aggregate() от метода annotate()? В каких ситуациях используется каждый из них?
6. Что делает метод values() и в чём его преимущество перед получением полных объектов модели?
7. Что такое объект F() и какие проблемы он решает? Приведите пример безопасного увеличения счётчика просмотров.
8. Как выполнить необработанный SQL-запрос через метод raw()? Какие меры предосторожности нужно соблюдать при передаче параметров?
9. Что такое «отложенная загрузка полей» (deferred loading)? Почему это может привести к проблемам с производительностью (N+1 запрос)?
10. Как настроить обратную связь между моделями с использованием параметра related_name? Приведите пример.

Продвинутый уровень

1. Как реализовать полнотекстовый поиск с использованием PostgreSQL в Django ORM? Какие инструменты для этого существуют?
2. Как работает механизм select_related и prefetch_related? В чём разница между ними и когда какой использовать?
3. Как реализовать пагинацию (постраничный вывод) с использованием Django ORM? Какие классы из django.core.paginator для этого используются?
4. Как выполнить сложный запрос с вложенными подзапросами (subquery) с использованием Django ORM? Приведите пример.
5. Что такое RawSQL и в каких случаях его использование оправдано, несмотря на рекомендацию использовать ORM?

6. Как в Django ORM выполнить обновление нескольких записей с использованием разных значений (CASE WHEN)?
7. Как работает механизм транзакций в Django ORM? Как обеспечить атомарность выполнения нескольких операций?
8. Что такое «миграции» и как они связаны с изменениями в моделях? Как создать и применить миграцию?
9. Как оптимизировать запрос, который вызывает проблему N+1 при получении связанных данных? Приведите пример исправления.
10. Как настроить кэширование результатов QuerySet для повышения производительности?

Лабораторная работа № 14.

Установка Django Debug Toolbar. Оптимизация SQL-запросов. Кастомизация админки. Миксины. Постраничная навигация

Цель работы: Изучить и освоить инструменты оптимизации SQL-запросов в Django (Django Debug Toolbar, select_related, prefetch_related), научиться применять миксины (классы-примеси) для вынесения общей логики и повторного использования кода, реализовать постраничную навигацию (пагинацию) для списков объектов, а также освоить базовую кастомизацию административной панели Django.

Методика выполнения лабораторной работы

1. **Установка Django Debug Toolbar. Оптимизация SQL-запросов**

В данной лабораторной работе мы установим один полезный пакет, позволяющий на этапе разработки отслеживать корректность и оптимальность работы приложения.

Инструкция по установке:

https://django-debug-toolbar.readthedocs.io/en/latest/

1. Введите в терминал команду **pip install django-debug-toolbar** для скачивания пакета
2. Добавьте **debug_toolbar** в список установленных приложений (файл настроек) и **debug_toolbar.middleware.DebugToolbarMiddleware** в список связующего ПО (Middleware). (рисунок 1)

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'debug_toolbar',
    'news.apps.NewsConfig',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    'debug_toolbar.middleware.DebugToolbarMiddleware',
]
```

Рисунок 1

3. Добавьте ссылки django-debug-toolbar в файле urls.py проекта. (рисунок 2)
4. Укажите IP-адреса, для которых должна показываться панель. Так как мы работаем локально, то добавим в файле настроек только 127.0.0.1. (рисунок 3)
5. Если вы нигде не ошиблись, то обновив страницу браузера вы увидите отладочную панель. (рисунок 4)

```python
if settings.DEBUG:
    import debug_toolbar

    urlpatterns = [
                      path("__debug__/", include("debug_toolbar.urls")),
                  ] + urlpatterns
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Рисунок 2

```python
INTERNAL_IPS = ["127.0.0.1"]
```

Рисунок 3

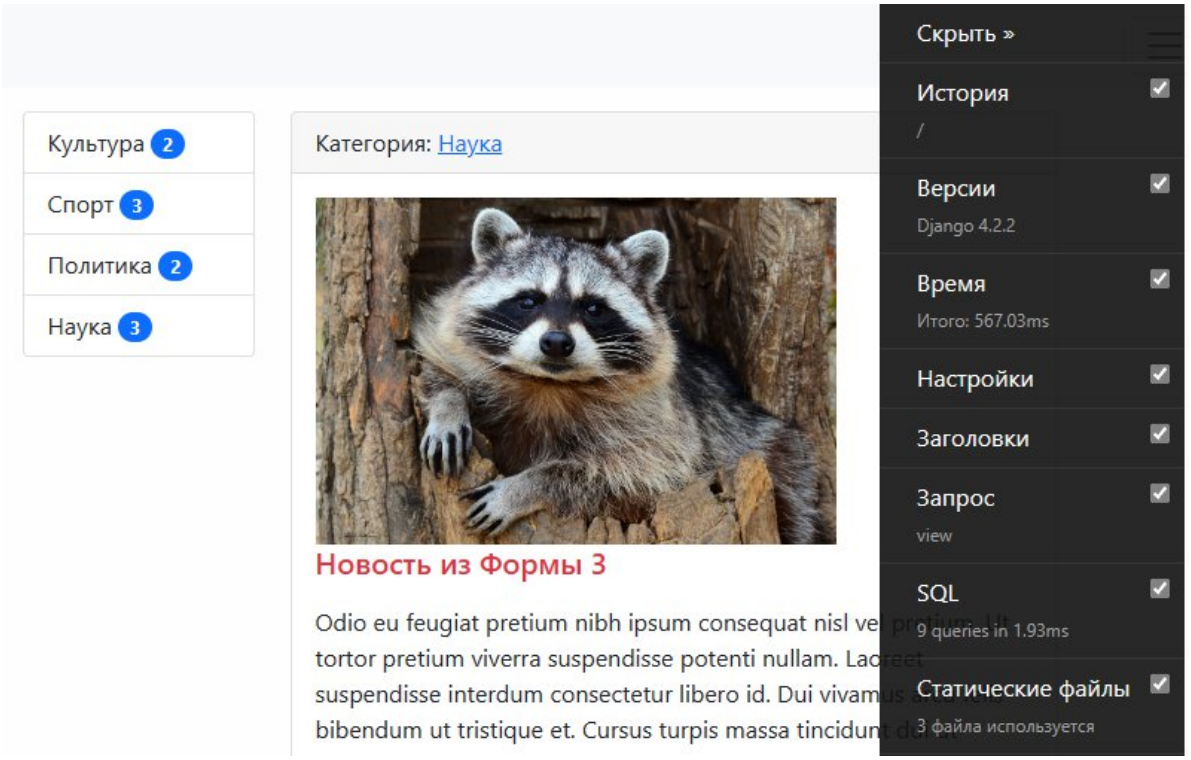


Рисунок 4

Оптимизация SQL-запросов

Раскройте вкладку SQL. На открывшейся странице вы увидите подробную информацию обо всех запросах, участвующих в формировании текущей страницы сайта. Обратите внимание, что под многими из них есть маркеры о дублировании и использовании схожих запросов. (рисунок 5)

Сейчас в формировании домашней страницы участвуют 9 запросов.

Наша задача – оптимизировать запросы. На самом деле в Django достаточно просто это реализовать. Для этого достаточно вызвать метод select_related и указать название поля внешнего ключа. (рисунок 5)

Обратите внимание, метод может вызывать и без параметров.

Документация:

https://docs.djangoproject.com/en/4.2/ref/models/querysets/#select-related

```
class HomeNews(ListView):
    model = News
    template_name = 'news/home_news_list.html'
    context_object_name = 'news'
    # extra_context = {'title': 'Главная'}

    AnnRedCrow
    def get_context_data(self, *, object_list=None, **kwargs):...

    AnnRedCrow *
    def get_queryset(self):
        return News.objects.filter(is_published=True).select_related('category')
```

Рисунок 5

В результате Django объединит простые запросы в более сложные, уберет отложенную загрузку полей, если она есть. Так вместо 9 запросов у нас получилось 2. (рисунок 6)

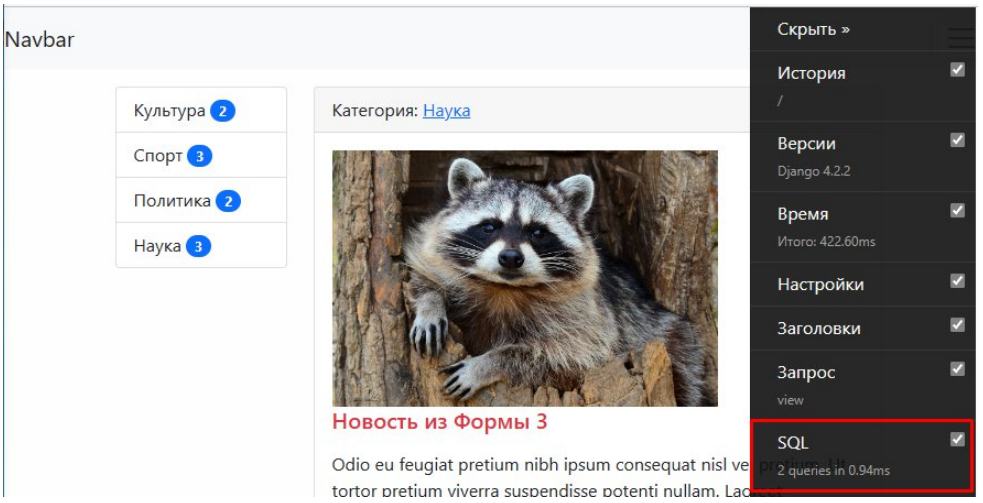


Рисунок 6

Аналогичным образом оптимизируем запросы для формирования страницы с выбранной категорией. (рисунок 7, рисунок 8, рисунок 9)

```python
class NewsByCategory(ListView):
    model = News
    template_name = 'news/home_news_list.html'
    context_object_name = 'news'
    allow_empty = False

    AnnRedCrow
    def get_context_data(self, *, object_list=None, **kwargs):...

    AnnRedCrow *
    def get_queryset(self):
        return News.objects.filter(category_id=self.kwargs['category_id'], is_published=True).select_related('category')
```

Рисунок 7

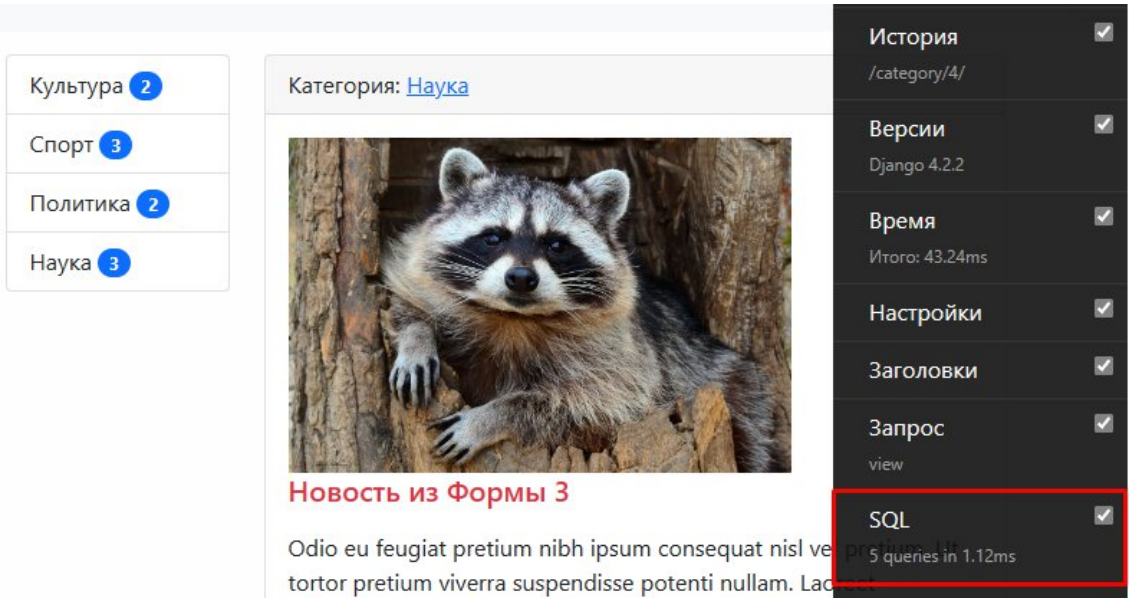


Рисунок 8

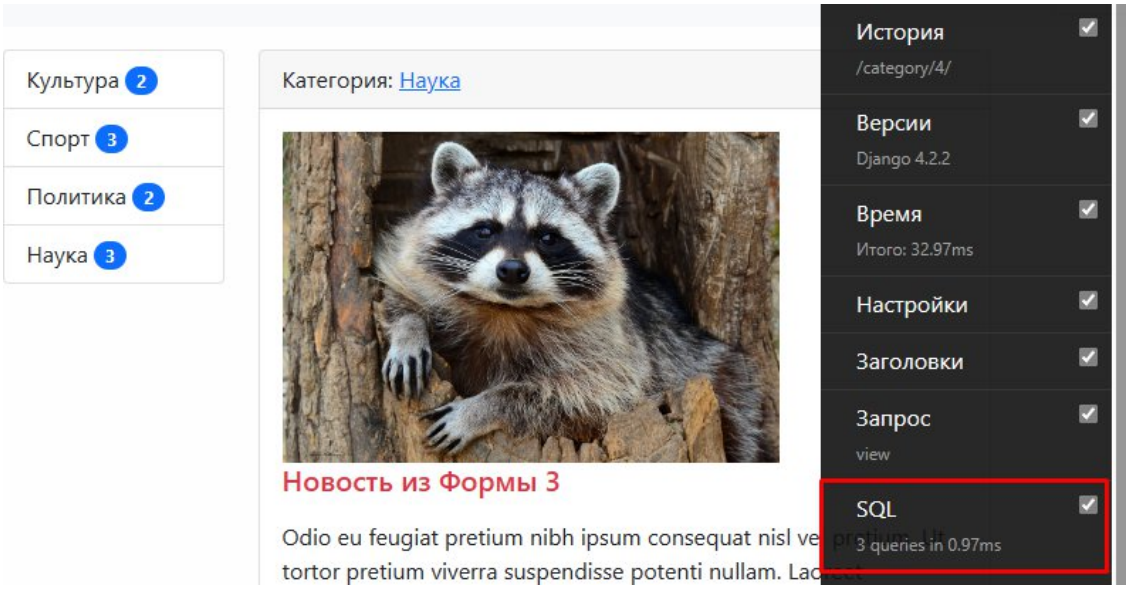


Рисунок 9

2. Миксины

В языке Python поддерживается множественное наследование. Однако не стоит забывать, что более значимые классы должны наследоваться раньше, так как поиск метода/атрибута прекращается при нахождении первого совпадения.

Миксины(классы-примеси) служат для вынесения общей логики и ,как следствие, сокращения количества кода.

Для начала рассмотрим довольно абстрактные примеры использования миксин. В Django следует добавлять классы-примеси в отдельном файле **utils.py**, создайте его в папке приложения.

Создание собственного миксина

Создадим класс MyMixin, добавим метод get_prop, возвращающий содержимое атрибута **mixin_prop** в верхнем регистре. (рисунок 10) Чтобы иметь возможность вызывать функции и атрибуты миксина, его необходимо указать в качестве родителя.

Давайте выведем строку «hello world» в шаблоне **home_news_list.html** в начале блока контента. (рисунок 12, рисунок 13) В результате мы увидим ее в верхнем регистре в начале страницы. (рисунок 14)

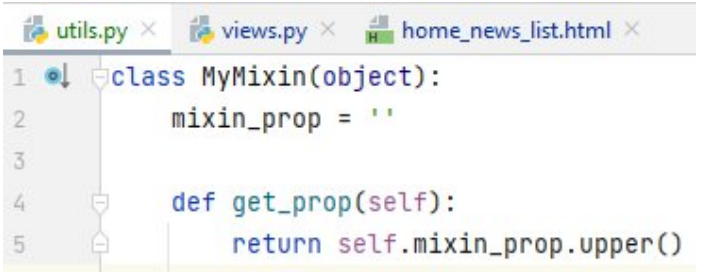


Рисунок 10

utils.py | views.py | home_news_list.html

```
4
5   from .models import News, Category
6   from .forms import NewsForm
7   from .utils import MyMixin
8
9
    AnnRedCrow *
10  class HomeNews(MyMixin, ListView):
11      model = News
12      template_name = 'news/home_news_list.html'
13      context_object_name = 'news'
14      mixin_prop = 'hello world'
15
16
```

Рисунок 11

```
    AnnRedCrow *
17  def get_context_data(self, *, object_list=None, **kwargs):
18      context = super().get_context_data(**kwargs)
19      context['title'] = 'Главная страница'
20      context['mixin_prop'] = self.get_prop()
21      return context
```

Рисунок 12

utils.py | views.py | home_news_list.html

```
10
11  {% block content %}
12  {{mixin_prop}}
13  {% for item in news %}
```

Рисунок 13

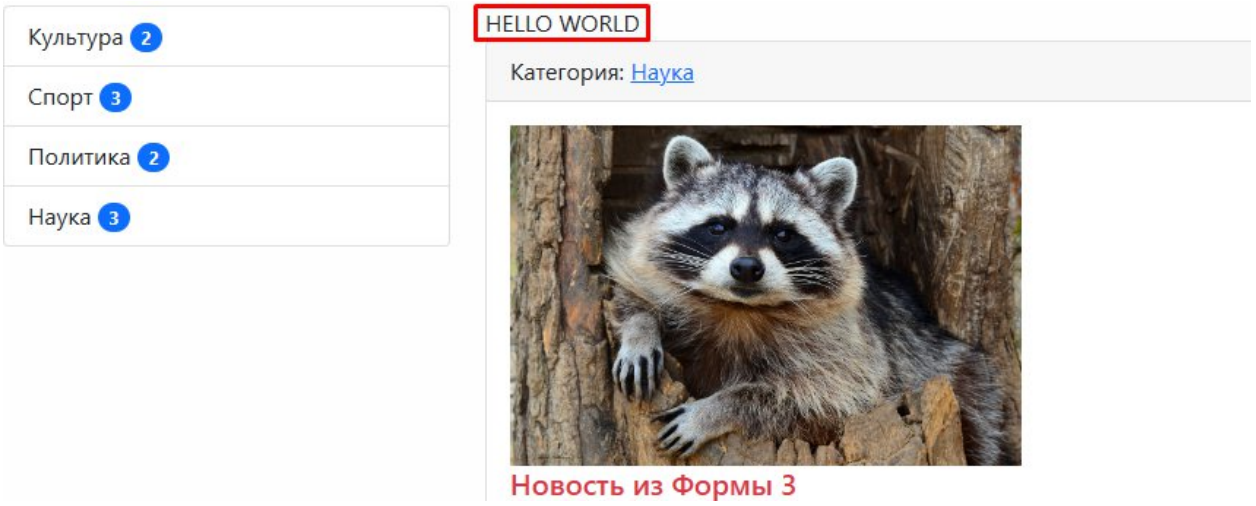


Рисунок 14

Далее давайте добавим еще один метод – **get_upper**, он будет принимать в качестве аргумента строку **s** и возвращать ее в верхнем регистре. (рисунок 15). Используем его для отображения тайтла страницы заглавными буквами. (рисунок 16, рисунок 17)

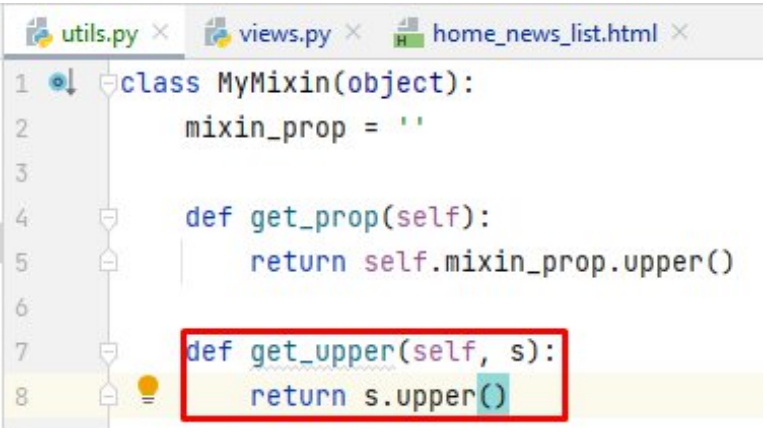


Рисунок 15

```
class HomeNews(MyMixin, ListView):
    model = News
    template_name = 'news/home_news_list.html'
    context_object_name = 'news'
    mixin_prop = 'hello world'


    AnnRedCrow *
    def get_context_data(self, *, object_list=None, **kwargs):
        context = super().get_context_data(**kwargs)
        context['title'] = self.get_upper('Главная страница')
        context['mixin_prop'] = self.get_prop()
        return context
```

Рисунок 16

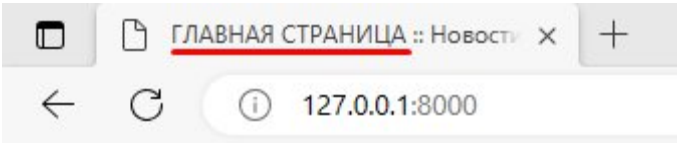


Рисунок 17

Если мы захотим, чтобы при выборе категории она отображалась аналогичным способом, то столкнемся с проблемой. Выражение Category.objects.get(pk=self.kwargs['category_id']) возвращает объект со всеми соответствующими атрибутами, поэтому мы не можем передать его в метод.

Для решения этого вопроса расширим метод get_upper(self, s), добавив проверку типа параметра s. Если это строка, все работает как прежде, а если объект – вызываем его атрибут title и возводим его в верхний регистр. (рисунок 18). Вызываем метод в классе NewsByCategory (рисунок 19) и получаем требуемый результат(рисунок 20).

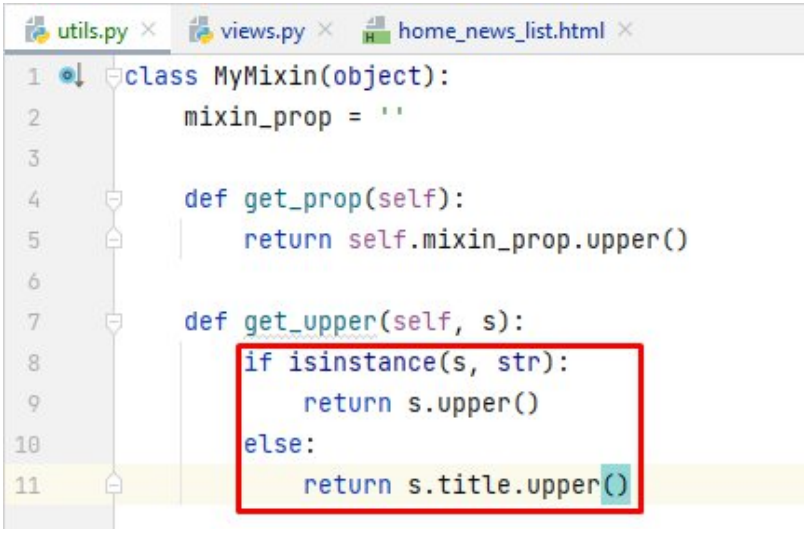


Рисунок 18

```
class NewsByCategory(MyMixin, ListView):
    model = News
    template_name = 'news/home_news_list.html'
    context_object_name = 'news'
    allow_empty = False

    AnnRedCrow *
    def get_context_data(self, *, object_list=None, **kwargs):
        context = super().get_context_data(**kwargs)
        context['title'] = self.get_upper(Category.objects.get(pk=self.kwargs['category_id']))
        return context
```

Рисунок 19

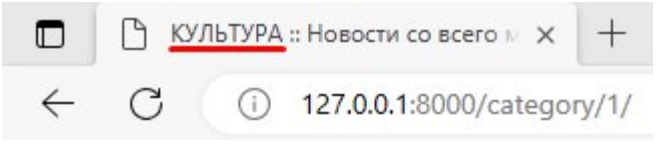


Рисунок 20

Ограничение доступа для неавторизованных пользователей. LoginRequiredMixin.

Ранее мы рассматривали довольно абстрактные примеры, теперь же изучим более полезный.

Наша задача – ограничить доступ к добавлению новостей для неавторизованных пользователей.

Обратите внимание, миксины могут быть использованы только для представлении-классов. В случае функций для решения задачи необходимо обратиться к декоратору login_required. В лабораторной работе мы не будем рассматривать этот вариант.

Вы можете использовать атрибут request.user для разделения предоставляемой информации для зарегистрированных и анонимных пользователей. Обратимся к нему в шаблоне _nav.html, чтобы скрыть ссылку на добавление новости в панели навигации. (рисунок 21)

Документация:
https://docs.djangoproject.com/en/4.2/topics/auth/default/#authentication-in-web-requests

Однако, если пользователю известна ссылка, то он может перейти по ней и получить доступ к добавлению новости(рисунок 22). Чтобы устранить эту уязвимость воспользуемся миксином LoginRequiredMixin. По возможности его следует указывать в самом начале списка наследуемых классов.

Добавим миксин в список наследуемых классов класса CreateNews. Далее у вас есть 2 варианта работы с пользователем:

1. Перенаправить пользователя на страницу регистрации, используя атрибут **login_url** (рисунок 23, рисунок 25)
2. Отобразить ошибку 403, используя атрибут **raise_exception** (рисунок 24, рисунок 26)

utils.py views.py _nav.html home_news_list.html

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <a class="navbar-brand" href="{% url 'home' %}">Navbar</a>
    <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupportedContent"
            aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
    </button>

    <div class="collapse navbar-collapse" id="navbarSupportedContent">
        <ul class="navbar-nav mr-auto">
            <li class="nav-item"><a class="nav-link" href="{% url 'home' %}">Главная</a></li>
            {% if request.user.is_authenticated %}
                <li class="nav-item"><a class="nav-link" href="{% url 'add_news' %}">Добавить новость</a></li>
            {% endif %}
        </ul>
    </div>
</nav>
```

Рисунок 21

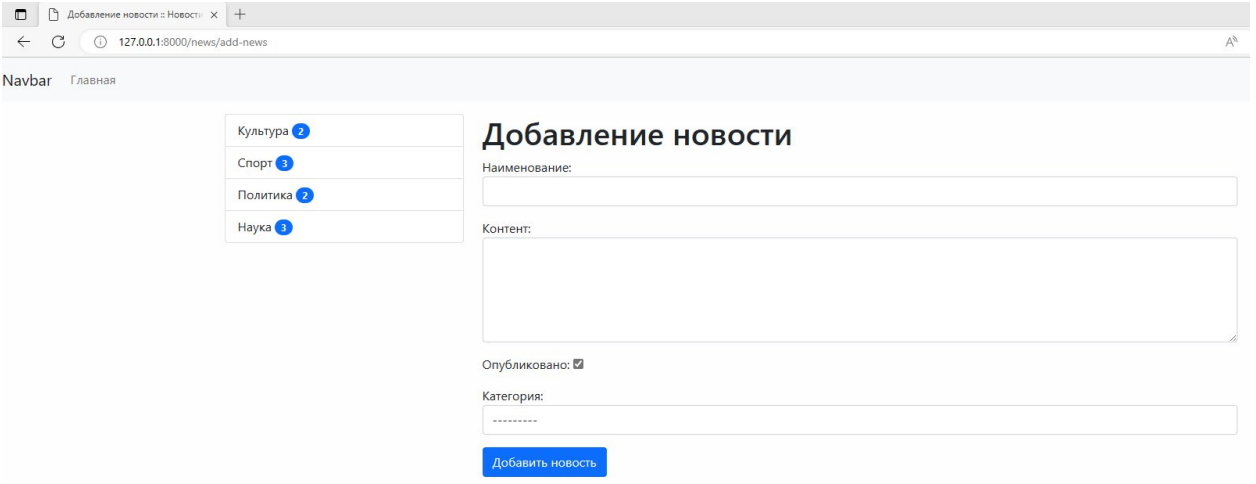


Рисунок 22

```
class CreateNews(LoginRequiredMixin, CreateView):
    form_class = NewsForm
    template_name = 'news/add_news.html'
    # success_url = reverse_lazy('home')
    login_url = '/admin/'
```

Рисунок 23

```
class CreateNews(LoginRequiredMixin, CreateView):
    form_class = NewsForm
    template_name = 'news/add_news.html'
    # success_url = reverse_lazy('home')
    # login_url = '/admin/'
    raise_exception = True
```

Рисунок 24

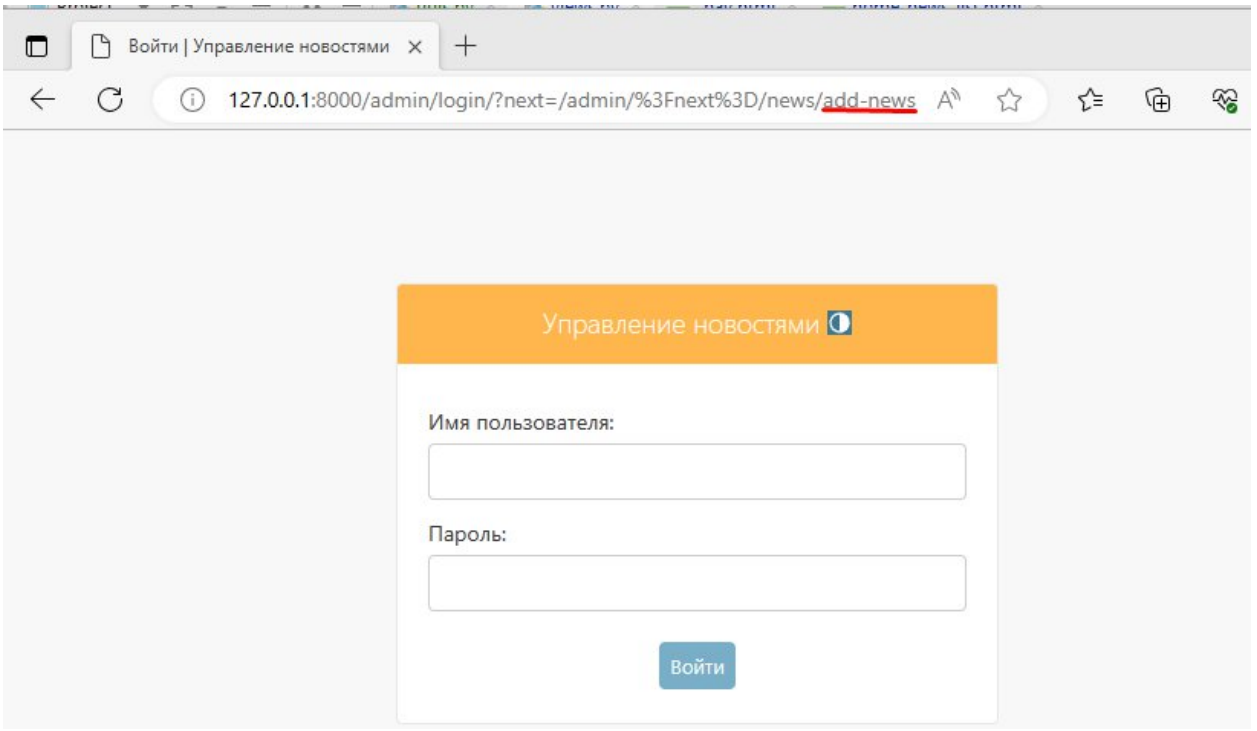


Рисунок 25

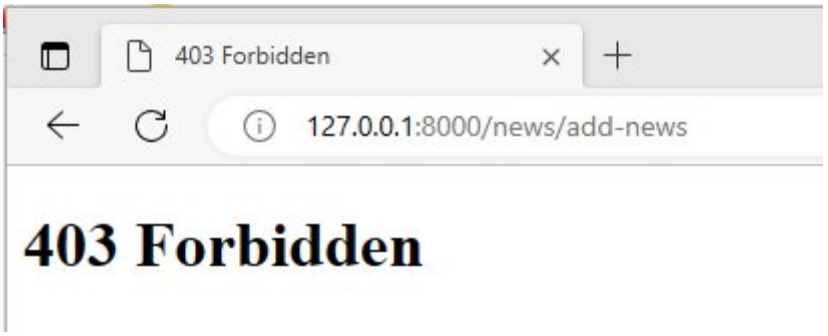


Рисунок 26

Подробнее о LoginRequiredMixin:

https://docs.djangoproject.com/en/4.2/topics/auth/default/#the-loginrequiredmixin-mixin

В процессе выполнения лабораторной работы мы вами рассмотрели создание собственных миксин(при желании вы можете выносить в них общий код, используемый несколькими классами, например, получение данных контекста) и работу с существующими.

3. Постраничная навигация (Пагинация, Pagination)

Документация:

https://docs.djangoproject.com/en/4.2/topics/pagination/

https://docs.djangoproject.com/en/4.2/ref/paginator/

Пагинация может быть реализована как для представлений-функций, так и для представлений-классов. Мы рассмотрим оба варианта.

Для начала необходимо импортировать класс Paginator:

from django.core.paginator import Paginator

Постраничная *навигация для функций*

Создадим тестовую функцию test (рисунок 27):

- В качестве списка для работы возьмем измененный вариант из документации: objects = ["john1", "paul2", "george3", "ringo4", "john5", "paul6", "george7"].
- Создадим объект класса Paginator paginator, передадим ему список objects и количество записей для вывода на одной странице.
- Получим номер текущей страницы из метода массива GET объекта request. Этого параметра может не быть, поэтому установим дефолтное значение равное единице.
- Далее получим объекты, соответствующие текущей странице. Тут возможно использовать 2 метода: page() и get_page(). Первый вызывает ошибку в случае отсутствия объектов для вывода, второй более гибкий и работает по умолчанию без исключений.
- Вернем отрендеренную страницу, в качестве контекста передадим “page_obj”:page_objects.

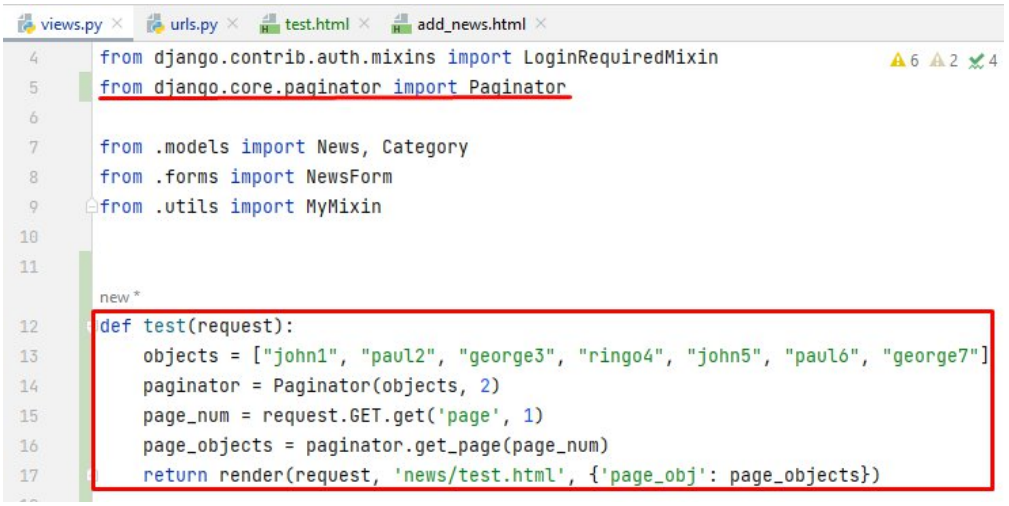


Рисунок 27

Мы используем **page_obj** для единой пагинации у функции и класса.

Далее добавим новый путь в файле urls.py. (рисунок 28)

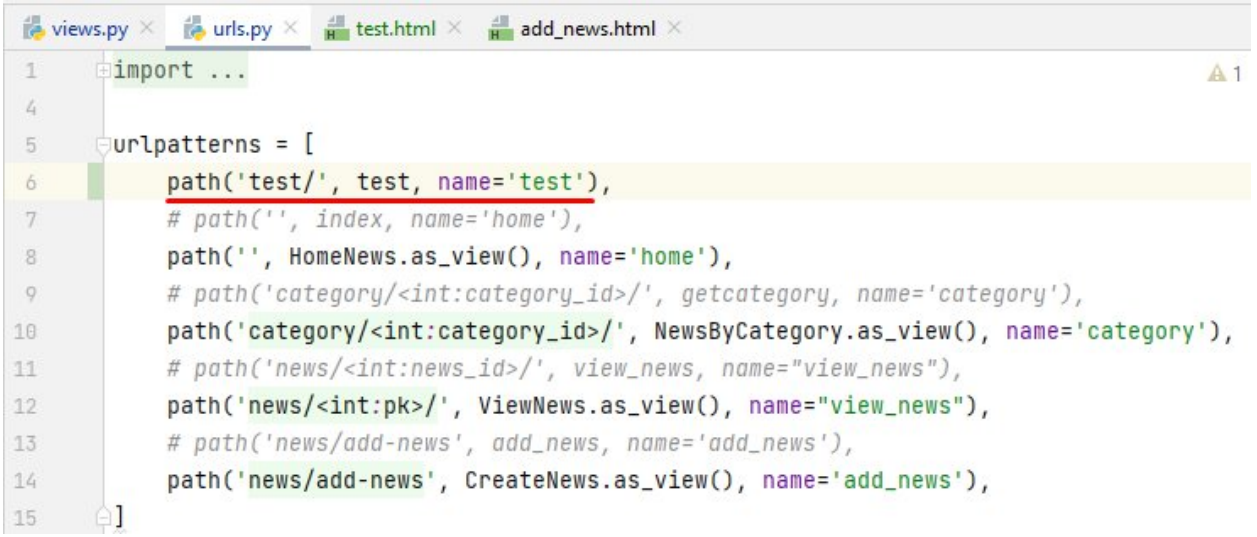


Рисунок 28

Последним шагом создадим файл шаблона test.html, скопируем код из add_news, удалим форму и заголовок h1, поменяем title. Далее для отображения объектов списка организуем цикл, перебирающий элементы списка, полученного из контекста. (рисунок 29)

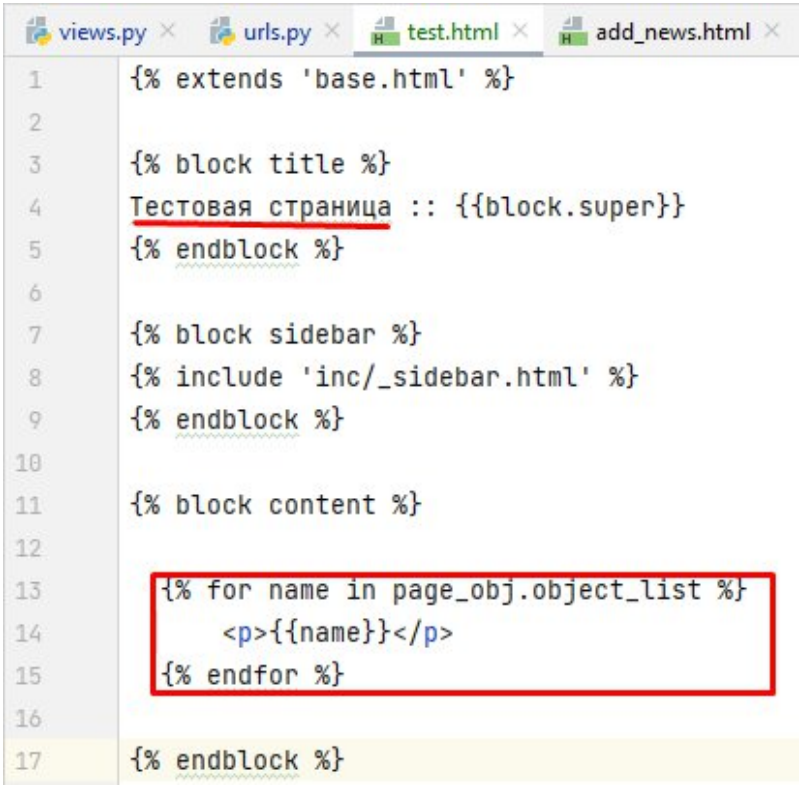


Рисунок 29

В результате все будет отображаться корректно. (рисунок 30)

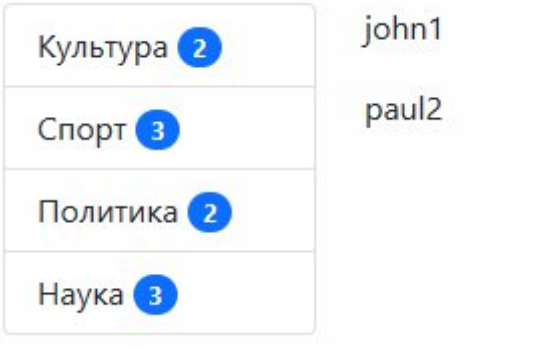


Рисунок 31

Однако у нас отсутствует возможность удобного перехода по страницам. На данном этапе единственный вариант – изменение ссылки, но мы как пользователи не знаем точно, сколько страниц существует. Давайте организуем список ссылок с номерами страниц внизу выведенных объектов.

Обратимся к Bootstrap и возьмем первый листинг по ссылке:

https://bootstrap-4.ru/docs/5.0/components/pagination/, поместим его после блока контента в базовом шаблоне base.html.

Оставим одну ссылку и поместим ее в цикл. Его диапазон определим с помощью специального атрибута **page_range**, он содержит (1, количество страниц + 1). (рисунок 32)

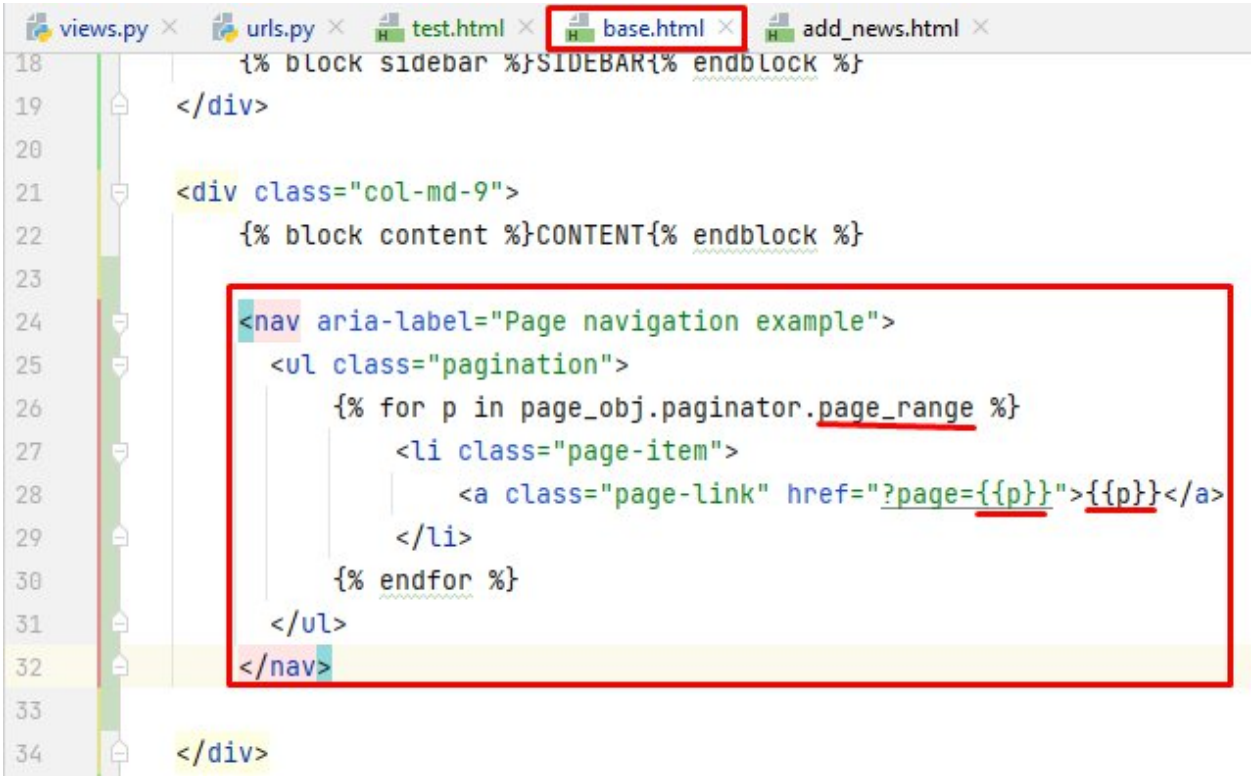


Рисунок 32

Получим требуемый результат (рисунок 33).

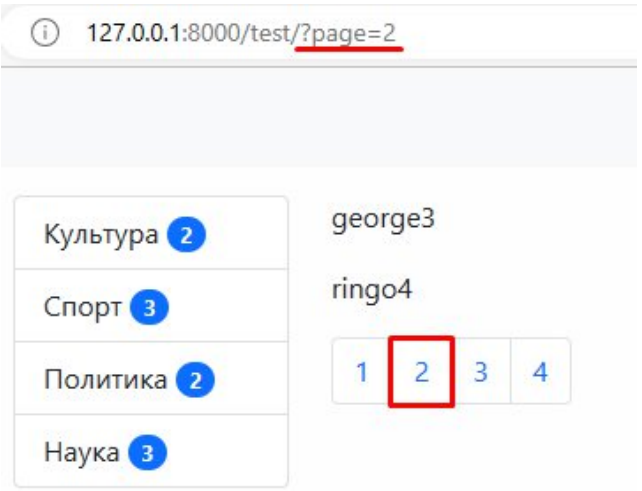


Рисунок 33

Постраничная навигация для классов

Реализовать пагинацию для классов, наследующих ListView, очень легко, для этого достаточно использовать атрибут **paginate_by**, чье значение обозначает количество объектов на одной странице. (рисунок 34)

```python
class HomeNews(MyMixin, ListView):
    model = News
    template_name = 'news/home_news_list.html'
    context_object_name = 'news'
    mixin_prop = 'hello world'
    paginate_by = 2
```

Рисунок 34

Далее поработаем над оформлением.

Сделаем так, чтобы ссылка на текущую страницу выделялась цветом. Для этого в шаблоне необходимо указать условие, проверяющее, совпадает ли номер текущей страницы со значением переменной p. Если да – добавляем оформление, указав класс **active** для пункта списка **li**. (рисунок 35, рисунок 36)

```
<div class="col-md-9">
    {% block content %}CONTENT{% endblock %}

    <nav aria-label="Page navigation example">
      <ul class="pagination">
          {% for p in page_obj.paginator.page_range %}
              {% if page_obj.number == p %}
                  <li class="page-item active" area-current="page">
                    <a class="page-link" href="?page={{p}}">{{p}}</a>
                  </li>
              {% else %}
                  <li class="page-item">
                    <a class="page-link" href="?page={{p}}">{{p}}</a>
                  </li>
              {% endif %}
          {% endfor %}
      </ul>
    </nav>

</div>
```

Рисунок 35

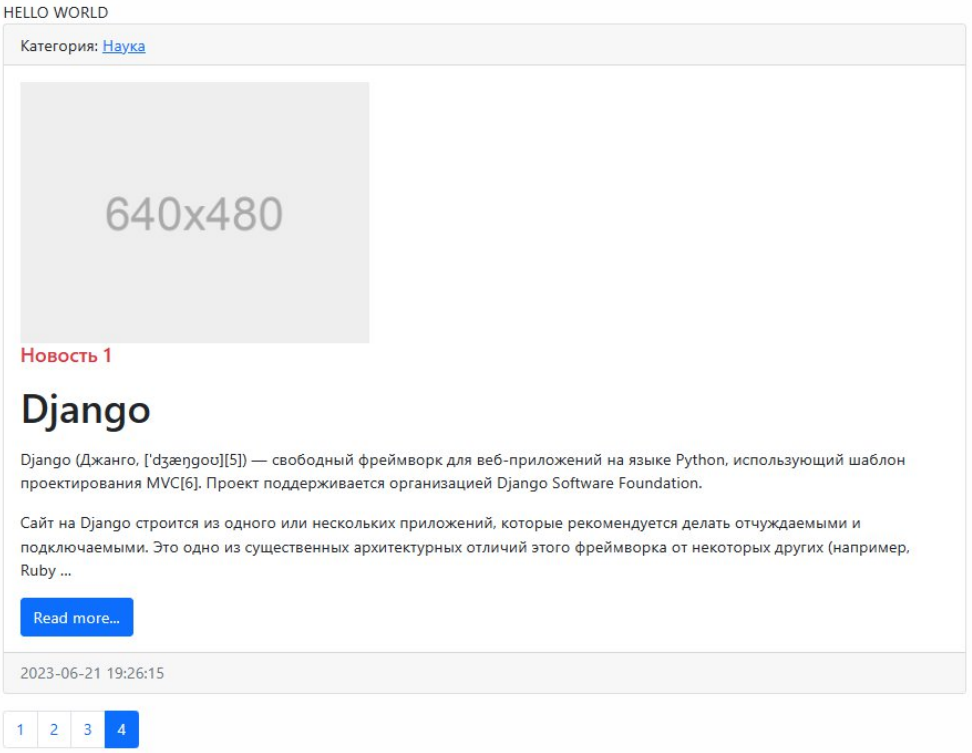


Рисунок 36

Для сайта с небольшим количеством хранимых данных записи **paginate_by = <количество записей на странице>** было бы достаточно для организации пагинации.

Однако в реальности базы данных сайтов хранят множество объектов, поэтому не стоит выводить все ссылки. Обычно оставляют 2–3 до и после текущей страницы. Давайте так и сделаем.

Для этого нам нужно заменить else на elif и добавить условие:

p > page_obj.number|add:-3 and p < page_obj.number|add:3. Таким образом номера страниц будут сравниваться с текущей и отображаться только при попадании в диапазон (p-3, p+3). (рисунок 37, рисунок 38)

```
{% for p in page_obj.paginator.page_range %}
    {% if page_obj.number == p %}
        <li class="page-item active" area-current="page">
          <a class="page-link" href="?page={{ p }}">{{p}}</a>
        </li>
    {% elif p > page_obj.number|add:-3 and p < page_obj.number|add:3 %}
        <li class="page-item">
          <a class="page-link" href="?page={{p}}">{{p}}</a>
        </li>
    {% endif %}
{% endfor %}
```

Рисунок 37

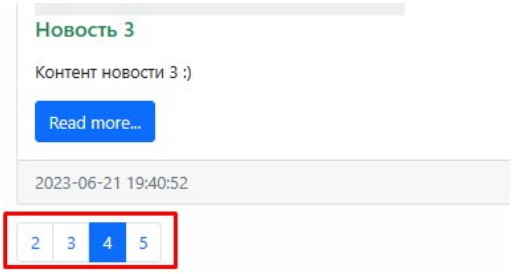


Рисунок 38

Далее добавим кнопки перехода на предыдущую и следующую страницу. В случае, если таких страниц нет, ссылки отображаться не должны.

Для этого нам понадобятся методы **has_previous** и **has_next** для проверки существования страниц и **previous_page_number** и **next_page_number** для получения номеров страниц и построения ссылок. (рисунок 39, рисунок 40)

```
<ul class="pagination">

    {% if page_obj.has_previous %}
        <li class="page-item">
            <a class="page-link" href="?page={{ page_obj.previous_page_number }}">Previous</a>
        </li>
    {% endif %}

    {% for p in page_obj.paginator.page_range %}
        {% if page_obj.number == p %}
            <li class="page-item active" area-current="page">
              <a class="page-link" href="?page={{ p }}">{{p}}</a>
            </li>
        {% elif p > page_obj.number|add:-3 and p < page_obj.number|add:3 %}
            <li class="page-item">
              <a class="page-link" href="?page={{p}}">{{p}}</a>
            </li>
        {% endif %}
    {% endfor %}

    {% if page_obj.has_next %}
        <li class="page-item">
            <a class="page-link" href="?page={{ page_obj.next_page_number }}">Next</a>
        </li>
    {% endif %}
```

Рисунок 39

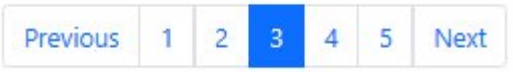


Рисунок 40

Осталось учесть последний момент, касающийся оформления. Заменим значение paginate_by с 2 на 20, теперь все новости у нас будут помещаться на одной странице. Однако внизу страницы также будет отображаться пагинация с одной ссылкой. Так как навигация в этом случае не нужна, уберем ее. Заключим весь ранее написанный код, касающийся пагинации в условие, проверяющее через метод **has_other_pages**, существуют ли другие страницы. (рисунок 41)

```
<div class="col-md-9">
    {% block content %}CONTENT{% endblock %}

    {% if page_obj.has_other_pages %}
    <nav aria-label="Page navigation example"...>
    {% endif %}

</div>
```

Рисунок 41

Мы закончили с оформлением. В завершение самостоятельно организуйте пагинацию для категорий. (Подсказка: вам будет достаточно использовать атрибут paginate_by для класса NewsByCategory)

Индивидуальные варианты выполнения

БЛОК 1: БАЗОВЫЙ УРОВЕНЬ

Требования: Установить Django Debug Toolbar, оптимизировать запросы на главной странице и странице категорий, добавить пагинацию на главную страницу.

Вариант 1

№	Задание
1	Установить Django Debug Toolbar, настроить в соответствии с инструкцией
2	Используя Debug Toolbar, определить количество SQL-запросов на главной странице (список новостей с категориями)
3	Оптимизировать главную страницу, добавив select_related('category') в контроллере HomeNews

№	Задание
4	Сравнить количество запросов до и после оптимизации, зафиксировать результат
5	Добавить пагинацию на главную страницу с выводом 5 новостей на страницу

Вариант 2

№	Задание
1	Определить количество SQL-запросов на странице категории NewsByCategory
2	Оптимизировать страницу категории, используя select_related('category')
3	С помощью Debug Toolbar убедиться, что количество запросов сократилось
4	Добавить пагинацию на страницу категории с выводом 3 новостей на страницу
5	Создать навигацию по страницам (ссылки «Предыдущая», «Следующая», номера страниц)

Вариант 3

№	Задание
1	Выявить дублирующиеся SQL-запросы на главной странице через Debug Toolbar
2	Применить select_related() без параметров для оптимизации
3	Сравнить сгенерированные SQL-запросы до и после оптимизации
4	Добавить пагинацию на главную страницу с выводом 10 новостей
5	Реализовать отображение текущей страницы (активный класс в Bootstrap)

Вариант 4

№	Задание
1	Проанализировать страницу отдельной новости (view_news) через Debug Toolbar
2	Оптимизировать запрос на получение новости с категорией через select_related
3	Убедиться, что лишние запросы исчезли
4	Добавить пагинацию на страницу категории с выводом 6 новостей
5	Скрыть пагинацию, если новостей меньше, чем заданное

№	Задание
	количество на странице

Вариант 5

№	Задание
1	Установить Django Debug Toolbar и убедиться в его работоспособности
2	Найти на странице категории запрос к базе данных, выполняющийся в цикле
3	Исправить проблему с помощью select_related
4	Добавить пагинацию на главную страницу с выводом 4 новостей
5	Реализовать ссылку на первую и последнюю страницу в пагинации

Вариант 6

№	Задание
1	Определить общее количество SQL-запросов на главной странице до оптимизации
2	Применить select_related('category') и повторно измерить количество запросов

№	Задание
3	Зафиксировать разницу в отчёте
4	Добавить пагинацию на страницу категории с выводом 8 новостей
5	Добавить кнопку «Следующая страница» с проверкой наличия следующей страницы

Вариант 7

№	Задание
1	С помощью Debug Toolbar найти запросы с пометкой «Similar Queries»
2	Объяснить причину появления похожих запросов
3	Оптимизировать запросы с использованием select_related
4	Добавить пагинацию на главную страницу с выводом 12 новостей
5	Реализовать ограничение количества отображаемых номеров страниц (показывать не более 5 ссылок)

Вариант 8

№	Задание
1	Проанализировать страницу со списком всех новостей

№	Задание
	(без фильтрации)
2	Оптимизировать запрос с помощью select_related('category')
3	Убедиться, что после оптимизации исчезли дополнительные запросы к таблице Category
4	Добавить пагинацию на страницу категории с выводом 7 новостей
5	Добавить отображение количества всех страниц в пагинации

Вариант 9

№	Задание
1	Определить, сколько запросов выполняется при отображении главной страницы с 3 новостями
2	Найти «проблемный» запрос, повторяющийся для каждой новости
3	Исправить с помощью select_related
4	Добавить пагинацию на главную страницу с выводом 9 новостей
5	Реализовать ссылку «Предыдущая страница» с проверкой

№	Задание
	наличия предыдущей страницы

Вариант 10

№	Задание
1	Установить Debug Toolbar и проверить работу на странице категории
2	Оптимизировать страницу категории, добавив select_related('category')
3	Сравнить время выполнения запросов до и после оптимизации
4	Добавить пагинацию на главную страницу с выводом 15 новостей
5	Добавить стилизацию для активной страницы в пагинации (Bootstrap)

Вариант 11

№	Задание
1	Проанализировать страницу с новостями, где используется prefetch_related (если есть)
2	Объяснить разницу между select_related и prefetch_related на примере

№	Задание
3	Добавить пагинацию на страницу категории с выводом 5 новостей
4	Реализовать отображение номеров страниц с использованием page_obj.paginator.page_range
5	Убрать пагинацию, если объектов меньше, чем paginate_by

Вариант 12

№	Задание
1	Установить Debug Toolbar, убедиться в его наличии в браузере
2	Оптимизировать запрос на главной странице с помощью select_related
3	Записать SQL-запрос, который генерируется после оптимизации
4	Добавить пагинацию на главную страницу с выводом 4 новостей
5	Реализовать кнопку «В конец» для перехода на последнюю страницу

Вариант 13

№	Задание
1	Определить количество запросов на странице категории с несколькими новостями
2	Найти причину дублирования запросов к категории
3	Исправить с помощью select_related('category')
4	Добавить пагинацию на страницу категории с выводом 6 новостей
5	Реализовать отображение текущей страницы и общего количества страниц

Вариант 14

№	Задание
1	Проанализировать страницу отдельной новости с комментариями (если есть)
2	Оптимизировать запрос комментариев через select_related или prefetch_related
3	Добавить пагинацию на главную страницу с выводом 8 новостей
4	Реализовать проверку: если есть предыдущая страница — показывать ссылку «Назад»
5	Стилизовать пагинацию с помощью Bootstrap Pagination

Вариант 15

№	Задание
1	Установить Django Debug Toolbar, настроить INTERNAL_IPS
2	Оптимизировать все представления, использующие News.objects.all() с select_related('category')
3	Проверить через Debug Toolbar, что оптимизация работает
4	Добавить пагинацию на страницу категории с выводом 10 новостей
5	Реализовать полную навигацию: «Первая», «Предыдущая», номера страниц, «Следующая», «Последняя»

БЛОК 2: СРЕДНИЙ УРОВЕНЬ

Требования: Реализовать пагинацию для всех списков (главная, категории, поиск), создать собственный миксин для вывода заголовка в верхнем регистре, применить LoginRequiredMixin для защиты добавления новости.

Вариант 1

№	Задание
1	Создать миксин UpperCaseMixin с методом get_upper, преобразующим строку в верхний регистр

№	Задание
2	Применить миксин к классу HomeNews для вывода заголовка страницы в верхнем регистре
3	Реализовать пагинацию для главной страницы с paginate_by = 5
4	Защитить CreateNews с помощью LoginRequiredMixin, перенаправлять на /admin
5	В шаблоне _nav.html скрыть ссылку на добавление новости для неавторизованных пользователей

Вариант 2

№	Задание
1	Создать миксин TitleMixin с атрибутом title_page и методом get_context_data для добавления заголовка
2	Применить миксин к HomeNews и NewsByCategory для установки заголовков страниц
3	Реализовать пагинацию для страницы категории с paginate_by = 4
4	Защитить CreateNews с помощью LoginRequiredMixin, перенаправлять на страницу входа
5	Создать собственный миксин MyMixin с

№	Задание
	методом get_prop, возвращающим строку в верхнем регистре

Вариант 3

№	Задание
1	Создать миксин ContextMixin для добавления в контекст переменной current_year (текущий год)
2	Применить миксин к HomeNews и вывести год в подвале сайта
3	Реализовать пагинацию для главной страницы с paginate_by = 6
4	Защитить CreateNews с помощью LoginRequiredMixin, использовать raise_exception = True
5	В шаблоне add_news.html вывести сообщение «Только для авторизованных» для неавторизованных

Вариант 4

№	Задание
1	Создать миксин CategoryMixin для получения списка категорий с количеством новостей
2	Применить миксин

№	Задание
	к HomeNews и NewsByCategory (убрать дублирование кода)
3	Реализовать пагинацию для страницы категории с paginate_by = 5
4	Защитить CreateNews с помощью LoginRequiredMixin, задать login_url = '/admin/'
5	В шаблоне пагинации скрыть блок, если страница только одна (has_other_pages)

Вариант 5

№	Задание
1	Создать миксин FilterPublishedMixin для фильтрации только опубликованных новостей
2	Применить миксин к HomeNews и NewsByCategory
3	Реализовать пагинацию для главной страницы с paginate_by = 8
4	Защитить UpdateNews (если есть) с помощью LoginRequiredMixin и UserPassesTestMixin
5	Добавить в пагинацию проверку: если есть следующая страница - показывать ссылку «Вперёд»

Вариант 6

№	Задание
1	Создать миксин OrderByDateMixin для сортировки новостей по дате (сначала новые)
2	Применить миксин к HomeNews и NewsByCategory
3	Реализовать пагинацию для страницы категории с paginate_by = 7
4	Защитить DeleteNews с помощью LoginRequiredMixin и проверить, что пользователь — админ
5	Добавить в пагинацию ссылки на первую и последнюю страницу

Вариант 7

№	Задание
1	Создать миксин BreadcrumbsMixin для формирования хлебных крошек (навигационной цепочки)
2	Применить миксин к NewsByCategory для отображения пути «Главная → Категория»
3	Реализовать пагинацию для главной страницы с paginate_by = 9
4	Защитить CreateNews с помощью LoginRequiredMixin, при отсутствии авторизации - 403 ошибка

№	Задание
5	В шаблоне пагинации выделить текущую страницу классом active

Вариант 8

№	Задание
1	Создать миксин CacheMixin для кэширования списка категорий (использовать cache.get и cache.set)
2	Применить миксин к HomeNews и NewsByCategory для оптимизации
3	Реализовать пагинацию для страницы категории с paginate_by = 6
4	Защитить CreateNews с помощью LoginRequiredMixin, использовать @method_decorator
5	Добавить в пагинацию отображение номеров страниц в диапазоне page_obj.number|add:-2 до +2

Вариант 9

№	Задание
1	Создать миксин MetaTagsMixin для добавления мета-тегов (description, keywords)
2	Применить миксин к HomeNews и NewsByCategory с

№	Задание
	разными значениями
3	Реализовать пагинацию для главной страницы с paginate_by = 10
4	Защитить CreateNews с помощью LoginRequiredMixin, перенаправлять на /login/
5	В пагинации реализовать отображение ссылок «Предыдущая» и «Следующая»

Вариант 10

№	Задание
1	Создать миксин ViewsCountMixin для увеличения счётчика просмотров новости
2	Применить миксин к ViewNews (страница отдельной новости)
3	Реализовать пагинацию для страницы категории с paginate_by = 5
4	Защитить CreateNews с помощью LoginRequiredMixin, задать сообщение о необходимости авторизации
5	В пагинации добавить условие: если page_obj.has_previous — показывать ссылку «Назад»

Вариант 11

№	Задание
1	Создать миксин CommentCountMixin для добавления в контекст количества комментариев к новости
2	Применить миксин к HomeNews для отображения количества комментариев на главной
3	Реализовать пагинацию для главной страницы с paginate_by = 12
4	Защитить CreateNews с помощью LoginRequiredMixin и проверить, что пользователь активен
5	В пагинации ограничить количество отображаемых страниц до 5

Вариант 12

№	Задание
1	Создать миксин RelatedNewsMixin для получения 3 похожих новостей (по категории)
2	Применить миксин к ViewNews для отображения похожих новостей
3	Реализовать пагинацию для страницы категории с paginate_by = 8

№	Задание
4	Защитить UpdateNews с помощью LoginRequiredMixin и UserPassesTestMixin (только автор)
5	Добавить в пагинацию ссылки с номерами страниц для всего диапазона

Вариант 13

№	Задание
1	Создать миксин PopularNewsMixin для получения топ-5 популярных новостей (по просмотрам)
2	Применить миксин к HomeNews для отображения в сайдбаре
3	Реализовать пагинацию для страницы категории с paginate_by = 6
4	Защитить CreateNews с помощью LoginRequiredMixin и PermissionRequiredMixin
5	В пагинации реализовать отображение «Первая» и «Последняя» ссылки

Вариант 14

№	Задание
1	Создать миксин ArchiveMixin для получения архивных

№	Задание
	новостей (по месяцам)
2	Применить миксин к отдельной странице архива
3	Реализовать пагинацию для страницы архива с paginate_by = 10
4	Защитить DeleteNews с помощью LoginRequiredMixin (только администратор)
5	В пагинации добавить стилизацию активной страницы и hover-эффекты

Вариант 15

№	Задание
1	Создать миксин SearchMixin для поиска новостей по заголовку и содержанию
2	Применить миксин к странице поиска
3	Реализовать пагинацию для результатов поиска с paginate_by = 5
4	Защитить CreateNews с помощью LoginRequiredMixin, добавить сообщение в шаблон
5	В пагинации скрыть ссылки на страницы, если их нет (page_obj.paginator.num_pages > 1)

БЛОК 3: ПРОДВИНУТЫЙ УРОВЕНЬ

Требования: Кастомизация админки Django, создание собственных миксинов для сложной логики, оптимизация запросов с prefetch_related, реализация пагинации с помощью функций и классов.

Вариант 1

№	Задание
1	В админ-панели зарегистрировать модель News с кастомным ModelAdmin
2	Добавить в список отображения поля title, category, created_at, is_published
3	Добавить фильтр по полю category и is_published
4	Добавить поиск по полю title
5	Реализовать пагинацию для главной страницы с помощью функций (используя Paginator)

Вариант 2

№	Задание
1	В админ-панели создать CategoryAdmin с полями title, slug, description
2	Добавить поиск по title и slug
3	Добавить фильтр по дате создания

№	Задание
4	Реализовать пагинацию для страницы категории с помощью функций (используя Paginator)
5	Создать собственный миксин AdminPermissionMixin для ограничения доступа к админке

Вариант 3

№	Задание
1	В админ-панели добавить поле category в list_editable для модели News
2	Добавить действие (action) для массовой публикации выбранных новостей
3	Настроить prepopulated_fields для поля slug в категориях
4	Реализовать пагинацию для результатов поиска с помощью функций
5	Создать миксин OrderMixin для сортировки по любому полю, передаваемому через GET-параметр

Вариант 4

№	Задание
1	В админ-панели настроить иерархию для категорий

№	Задание
	(родительская категория)
2	Добавить list_select_related = ('parent',) для оптимизации
3	Реализовать пагинацию для главной страницы с помощью классов (атрибут paginate_by)
4	Создать миксин DateRangeMixin для фильтрации новостей по диапазону дат
5	Настроить отображение пагинации на странице категории через page_range

Вариант 5

№	Задание
1	В админ-панели добавить поле views в readonly_fields для модели News
2	Настроить fieldsets для группировки полей в админке
3	Добавить в ModelAdmin метод get_queryset с оптимизацией select_related
4	Реализовать пагинацию для страницы категории с помощью классов
5	Создать миксин PaginateMixin с настраиваемым paginate_by для переиспользования

Вариант 6

№	Задание
1	В админ-панели создать CommentAdmin с полями news, author, rating, is_moderated
2	Добавить действие для массовой модерации комментариев (одобрение)
3	Добавить фильтр по rating и is_moderated
4	Реализовать пагинацию для страницы категории с кастомным диапазоном (показывать 3 страницы рядом)
5	Создать миксин CachedQuerysetMixin для кэширования результатов get_queryset

Вариант 7

№	Задание
1	В админ-панели добавить инлайн-модель комментариев для новости
2	Настроить list_display_links для модели News
3	Добавить save_on_top = True для удобства редактирования
4	Реализовать пагинацию с использованием page_obj.has_previous, has_next, previous_page_number, next_page_number
5	Создать миксин BreadcrumbMixin для динамического

№	Задание
	построения хлебных крошек в админке

Вариант 8

№	Задание
1	В админ-панели добавить кастомную вкладку (inline) для связанных тегов
2	Настроить ordering = ['-created_at'] для новостей в админке
3	Добавить show_full_result_count = False для ускорения подсчёта объектов
4	Реализовать пагинацию с ограничением отображаемых номеров (не более 5)
5	Создать миксин ActionPermissionMixin для проверки прав на массовые действия

Вариант 9

№	Задание
1	В админ-панели добавить date_hierarchy = 'created_at' для навигации по датам
2	Настроить empty_value_display = '— нет данных —' для пустых полей
3	Добавить кастомную get_queryset с аннотацией

№	Задание
	количества комментариев
4	Реализовать пагинацию с ссылками «Первая», «Последняя», «Предыдущая», «Следующая»
5	Создать миксин UserFilterMixin для фильтрации объектов по текущему пользователю

Вариант 10

№	Задание
1	В админ-панели добавить кастомную search_fields с поиском по связанной модели (category__title)
2	Настроить raw_id_fields для ForeignKey с большим количеством записей
3	Добавить list_per_page = 25 для пагинации в админке
4	Реализовать пагинацию для страницы с комментариями к новости
5	Создать миксин ExcelExportMixin для экспорта списка новостей в Excel через админку

Вариант 11

№	Задание
1	В админ-панели создать кастомную форму для добавления новости с валидацией заголовка

№	Задание
2	Настроить get_readonly_fields для динамического определения полей только для чтения
3	Добавить inlines для комментариев к новости
4	Реализовать пагинацию для главной страницы с использованием Paginator и Page
5	Создать миксин RelatedObjectsCountMixin для отображения количества связанных объектов в списке

Вариант 12

№	Задание
1	В админ-панели добавить кастомный formfield_for_foreignkey для фильтрации категорий
2	Настроить get_search_results для расширенного поиска
3	Добавить в NewsAdmin метод save_model для логирования изменений
4	Реализовать пагинацию для страницы категории с ограничением количества ссылок 2 до и после текущей
5	Создать миксин SlugMixin для автоматической генерации slug из заголовка

Вариант 13

№	Задание
1	В админ-панели добавить кастомный list_filter с диапазоном дат (DateRangeFilter)
2	Настроить autocomplete_fields для ForeignKey полей
3	Добавить в ModelAdmin метод get_list_display для динамического набора полей
4	Реализовать пагинацию для страницы поиска с сохранением параметров запроса
5	Создать миксин ActiveLinkMixin для добавления активной ссылки в навигации

Вариант 14

№	Задание
1	В админ-панели добавить кастомный AdminSite с изменённым заголовком и шаблоном
2	Настроить get_actions для скрытия некоторых действий для определённых пользователей
3	Добавить view_on_site = True для отображения кнопки просмотра на сайте
4	Реализовать пагинацию для страницы категории с отображением текущей страницы в URL
5	Создать миксин LastModifiedMixin для автоматического

№	Задание
	обновления updated_at при сохранении

Вариант 15

№	Задание
1	В админ-панели добавить кастомный ModelAdmin для модели Tag с slug
2	Настроить list_editable для поля is_published в новостях
3	Добавить radio_fields для выбора категории вместо выпадающего списка
4	Реализовать пагинацию для страницы категории с использованием get_page() вместо page()
5	Создать миксин PublishedManagerMixin с кастомным менеджером для опубликованных новостей

Контрольные вопросы

Базовый уровень

4. Для чего нужен Django Debug Toolbar? Какие основные вкладки (панели) он предоставляет?
5. Как установить и настроить Django Debug Toolbar? Какие изменения нужно внести в settings.py и urls.py?
6. Что такое select_related и для чего он используется? В каких случаях его применение эффективно?
7. Чем отличается select_related от prefetch_related? Приведите примеры использования каждого.

8. Что такое «проблема N+1 запроса»? Как её выявить с помощью Debug Toolbar и как исправить?
9. Что такое миксины (Mixins) в Python/Django? Для чего они используются?
10. Как создать собственный миксин? Приведите пример структуры миксина.
11. Что делает LoginRequiredMixin? Как его применить к классовому представлению?
12. Что такое пагинация? Для чего она нужна в веб-приложениях?
13. Как реализовать пагинацию для функционального представления? Какой класс для этого используется?

Средний уровень

1. Как в ListView добавить пагинацию? Какой атрибут для этого нужно указать?
2. Как в шаблоне получить доступ к объекту текущей страницы (page_obj)? Какие методы и атрибуты у него есть (has_previous, next_page_number и т.д.)?
3. Как в пагинации ограничить количество отображаемых номеров страниц? Приведите пример кода на Django Template Language.
4. Как сделать ссылку «Первая» и «Последняя» страница в пагинации? Как проверить, что текущая страница не первая/не последняя?
5. Как скрыть блок пагинации, если объектов меньше, чем paginate_by? Какой метод для этого используется?
6. Что такое page_range и как его использовать? Для чего нужен page_obj.paginator?
7. Чем отличается page() от get_page() при работе с Paginator? Какие ошибки они могут генерировать?

8. Как передать дополнительные параметры в пагинацию (например, фильтры)? Как сохранить GET-параметры в ссылках?
9. Как создать миксин, который добавляет данные в контекст (get_context_data)? Что важно помнить при переопределении этого метода?
10. Как защитить представление от доступа неавторизованных пользователей без использования миксинов? Какой декоратор используется для функций?

Продвинутый уровень

1. Как в админ-панели настроить list_editable? Какие ограничения есть у этого атрибута?
2. Как добавить кастомное действие (action) в админ-панель? Приведите пример массовой публикации новостей.
3. Что такое list_select_related в ModelAdmin? Зачем он нужен и как его использовать?
4. Как настроить prepopulated_fields для автоматического заполнения slug в админке?
5. Как в админ-панели добавить инлайн-модели (InlineModelAdmin)? В чём разница между TabularInline и StackedInline?
6. Что такое fieldsets в админ-панели? Как сгруппировать поля в разные секции?
7. Как в админ-панели добавить кастомный фильтр по диапазону дат (DateRangeFilter)?
8. Как в админ-панели переопределить метод get_queryset для оптимизации запросов с select_related?
9. Как создать кастомный AdminSite с собственной темой и названием?

10. Как реализовать пагинацию с сохранением состояния фильтрации и поиска (через GET-параметры)? Как правильно построить ссылки в шаблоне?

ИТОГОВЫЕ КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	**Базовый уровень (max 35)**	**Средний уровень (max 50)**	**Продвинутый уровень (max 60)**
Установка и настройка Debug Toolbar	5	—	—
Оптимизация SQL-запросов (select_related)	10	10	10
Создание собственных миксинов	—	15	15
Использование LoginRequiredMixin	—	10	10
Пагинация для функций/классов	15	10	10
Кастомизация админ-панели	—	—	15
Навигация в пагинации (ссылки, диапазоны)	5	5	10
Итого	**35**	**50**	**70**

Шкала перевода баллов в оценку

Оценка	Базовый уровень (35)	Средний уровень (50)	Продвинутый уровень (70)
5 (отлично)	32-35	46-50	64-70
4 (хорошо)	28-31	40-45	56-63
3 (удовлетворительно)	22-27	32-39	45-55
2 (неудовлетворительно)	0-21	0-31	0-44

Лабораторная работа №14 (часть2)

Визуальный редактор CKEditor. Капча. Кэширование

Цель работы: Изучить и освоить инструменты для улучшения пользовательского интерфейса административной панели и пользовательской части веб-приложения: установку и настройку визуального редактора CKEditor для форматирования контента, интеграцию капчи для защиты форм от автоматических отправок, а также освоить различные стратегии кэширования (полностраничное кэширование, кэширование фрагментов шаблона, низкоуровневое API кэширования) для оптимизации производительности и снижения нагрузки на базу данных.

Методика выполнения лабораторной работы

1. **Визуальный редактор CKEditor**

В этой лабораторной работе мы сделаем админку удобнее для редактирования контента на сайте рядовому пользователю, добавим возможности форматирования текста, прикрепления изображения, создания списка. Для этого установим графический редактор.

На самом деле, всю необходимую информацию и инструкции вы можете найти по этой ссылке: https://pypi.org/project/django-ckeditor/. Здесь мы продублируем некоторые важные моменты, касающиеся установки.

- Введите команду **pip install django-ckeditor** в терминале, чтобы установить пакет
- В настройках добавьте ckeditor в список установленных приложений (рисунок 1)

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'debug_toolbar',
    'ckeditor',
    'news.apps.NewsConfig',
]
```

Рисунок 1

- Соберите все статику: python manage.py collectstatic
- Укажите в настройках константу CKEDITOR_UPLOAD_PATH = "uploads/". По этому пути будут загружаться изображения.
- Так как за счет загрузки изображений CLEditor уже не просто графический редактор, но и файловый менеджер, то добавьте в список установленных приложений ckeditor_uploader.
- Добавьте CKEditor URL в файл urls.py проекта(папка mysite). (рисунок 2)
- Далее перейдите в файл admin.py, продолжим настройку там. Импортируйте виджет CKEditorUploadingWidget (если вам не нужна возможность загрузки файлов внутри формы, используйте CKEditorWidget). Перенесите код формы из документации, подкорректируйте его немного.

Замените CKEditorWidget на CKEditorUploadingWidget, укажите модель News. (рисунок 3)

```
settings.py | urls.py | admin.py | forms.py | views.py | test
1   """URL configuration for mysite project...."""
17  import ...
21
22
23  urlpatterns = [
24      path('admin/', admin.site.urls),
25      path('ckeditor/', include('ckeditor_uploader.urls')),
26      path('', include('news.urls')),
27  ]
```

Рисунок 2

```
settings.py | urls.py | admin.py | forms.py | views.py | test.html
1   from django.contrib import admin
2   from django.utils.safestring import mark_safe
3   from django import forms
4   from ckeditor_uploader.widgets import CKEditorUploadingWidget
5
6   from .models import News, Category
7
8
    AnnRedCrow
9   class NewsAdminForm(forms.ModelForm):
10      content = forms.CharField(widget=CKEditorUploadingWidget())
11
        AnnRedCrow
12      class Meta:
13          model = News
14          fields = '__all__'
```

Рисунок 3

Теперь, когда вы откроете режим создания/редактирования новости в административной части сайта, то увидите, что поле для ввода контента изменилось: появилась возможность форматирования текста, загрузки изображений на сервер и их настройки. (рисунок 4)

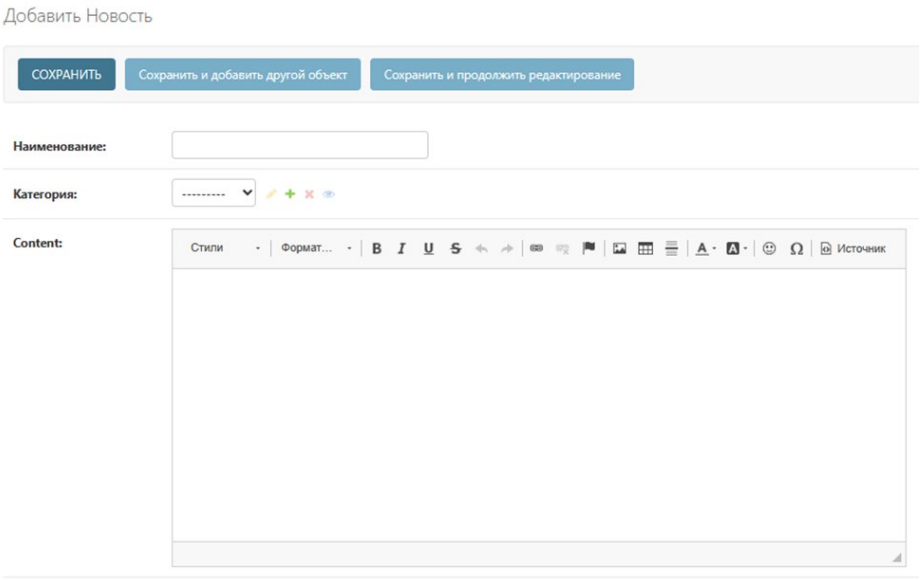


Рисунок 4

Давайте добавим новую новость: загрузим и настроим картинку (рисунок 5), добавим текст. (рисунок 6)

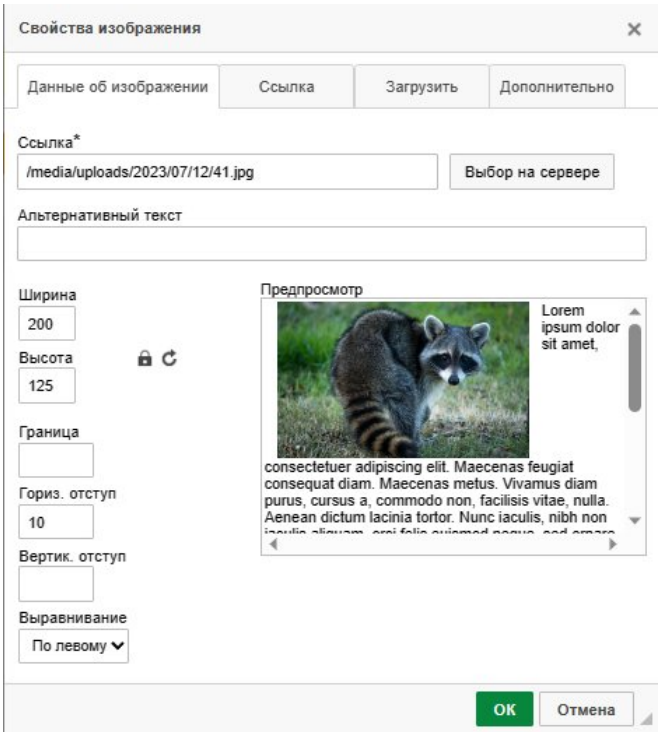


Рисунок 5



Рисунок 6

Обратите внимание, загружаемые на сервер изображения сохраняются в папке uploads внутри media и структурируются согласно указанным нами настройкам. (рисунок 7)

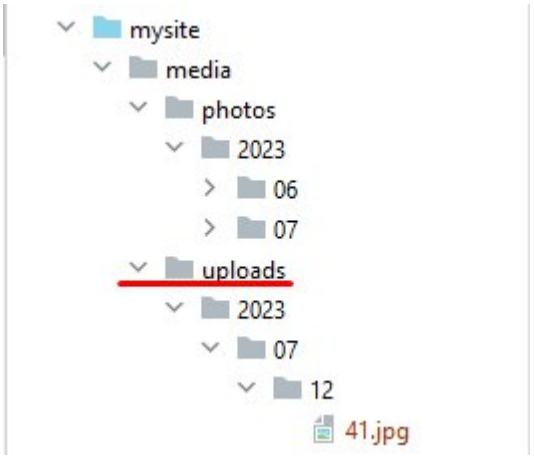


Рисунок 7

Вы можете расширить панель редактирования, добавив новые инструменты. Для этого перенесите код из документации в настройки. (рисунок 8)

Example ckeditor configuration

```
CKEDITOR_CONFIGS = {
    'default': {
        'skin': 'moono',
```

Рисунок 8

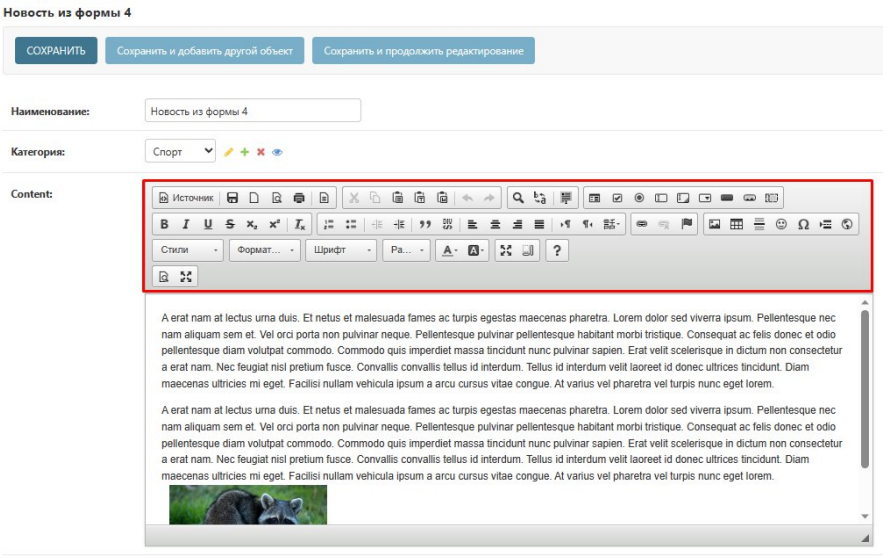


Рисунок 9

Теперь, открыв панель редактирования новости вы увидите, что возможности редактирования содержимого значительно расширились. (рисунок 9, рисунок 10)

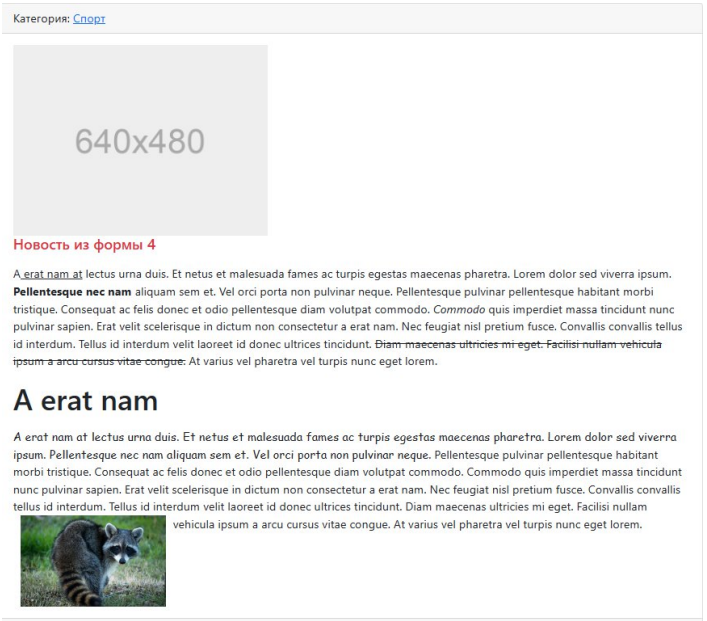


Рисунок 10

2. Капча

В ходе выполнения этой лабораторной работы мы усовершенствуем форму обратной связи, добавив к ней капчу.

Безусловно вы можете написать капчу самостоятельно, однако быстрее и удобнее, по крайней мере на начальных этапах, воспользоваться готовым пакетом.

Небольшое примечание

Ранее мы использовали название test для функции контроллера и ссылки, отвечающей за форму обратной связи. По возможности измените его на contact, чтобы оно отражало логическую структуру (рисунок 11). Также добавьте ссылку на шаблон навигационной панели (рисунок 12).

```
urlpatterns = [
    path('register/', register, name='register'),
    path('login/', user_login, name='login'),
    path('logout/', user_logout, name='logout'),
    path('contact/', contact, name='contact'),
```

Рисунок 11

```
<div class="collapse navbar-collapse" id="navbarSupportedContent">
    <ul class="navbar-nav mr-auto">
        <li class="nav-item"><a class="nav-link" href="{% url 'home' %}">Главная</a></li>
        {% if request.user.is_authenticated %}
            <li class="nav-item">
                <a class="nav-link" href="{% url 'add_news' %}">Добавить новость</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="{% url 'contact' %}">Обратная связь</a>
            </li>
        {% endif %}
    </ul>
```

Рисунок 12

Установка капчи

Мы установим самую простую капчу. Подробную документацию вы сможете найти по ссылке: https://django-simple-captcha.readthedocs.io/en/latest/.

Инструкция по установке:

- Установите пакет, введя следующую команду в терминале: pip install django-simple-captcha
- В настройках добавьте капчу в список установленных приложений (рисунок 3)
- Проведите миграцию: python manage.py migrate
- Добавьте ссылку в файл urls.py проекта (рисунок 14)

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'debug_toolbar',
    'ckeditor_uploader',
    'ckeditor',
    'captcha',
    'news.apps.NewsConfig',
]
```

Рисунок 13

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('ckeditor/', include('ckeditor_uploader.urls')),
    path('captcha/', include('captcha.urls')),
    path('', include('news.urls')),
]
```

Рисунок 14

Использование капчи в форме

Здесь все максимально просто: необходимо импортировать тип поля CaptchaField и затем добавить такое поле в саму форму. (рисунок 15)



Рисунок 15

В результате по умолчанию мы увидим капчу с 4 латинскими буквами в конце нашей формы. (рисунок 16)

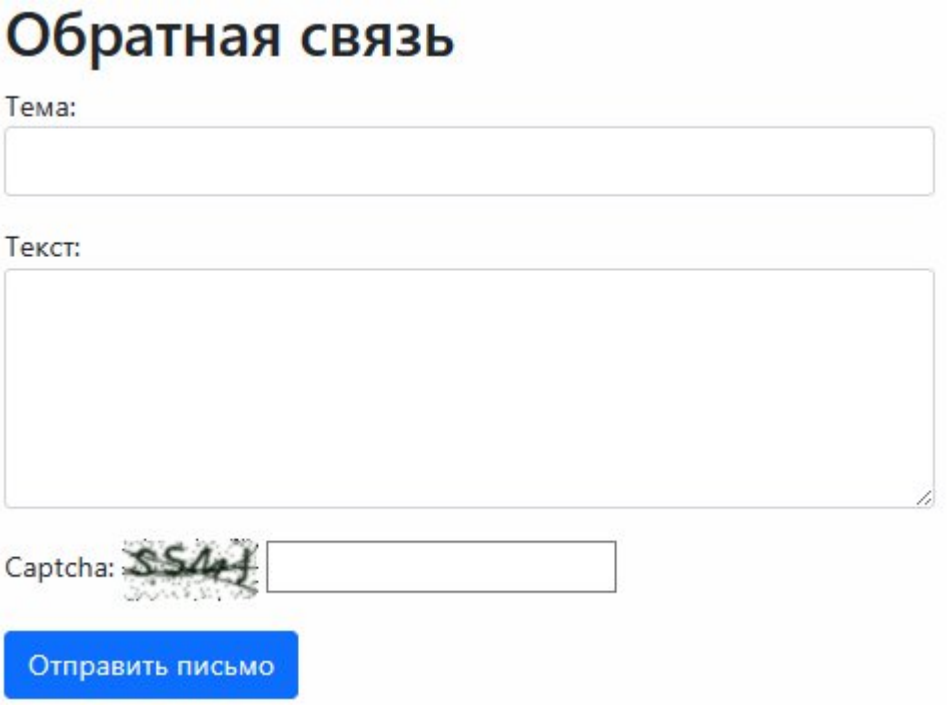


Рисунок 16

Логика работы капчи уже прописана в самом пакете, попробуйте ввести верное и неверное (рисунок 17) значение, чтобы убедиться. В случае совпадения письмо будет отправлено.

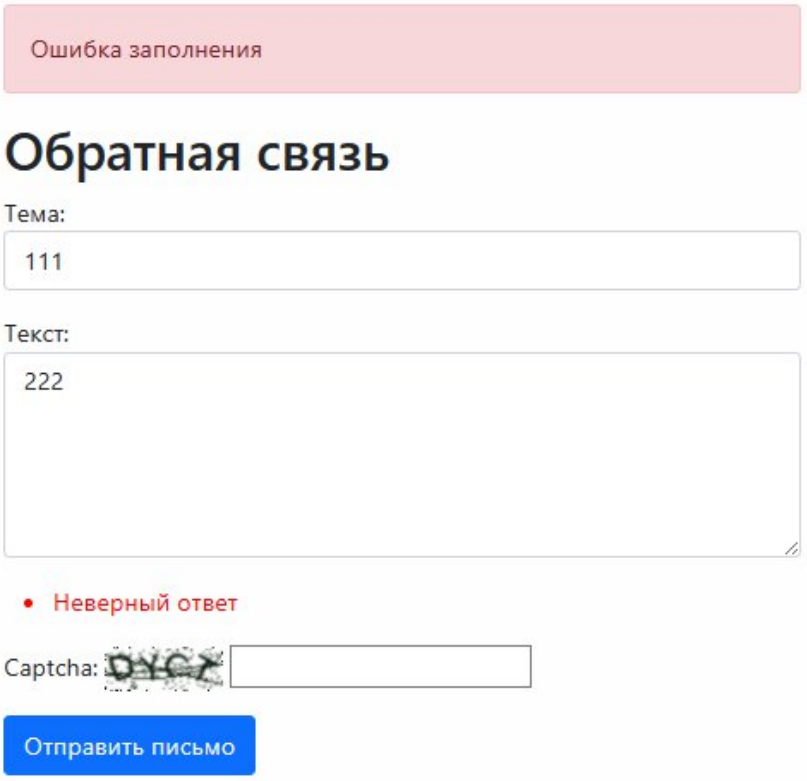


Рисунок 17

Настройка капчи

Список доступных настроек (они должны указываться в файле settings.py):

https://django-simple-captcha.readthedocs.io/en/latest/advanced.html#configuration-toggles

https://django-simple-captcha.readthedocs.io/en/latest/advanced.html#generators-and-modifiers

Рассмотрим некоторые из них. Об остальных читайте в документации.

CAPTCHA_LETTER_ROTATION

Настройка ротации букв. Каждая буква будет повернута со случайным значением, входящим в заданный диапазоном, по умолчанию это (-35,35) Чтобы отключить ротацию установите None (рисунок 18).



Рисунок 18

CAPTCHA_CHALLENGE_FUNCT

Настройка типа генерируемой капчи. Например, чтобы установить простые математические примеры в качестве проверки, используйте значение 'captcha.helpers.math_challenge' (рисунок 19).



Рисунок 19

CAPTCHA_NOISE_FUNCTIONS

Настройка шума на изображении. Чтобы убрать его, установите значение None. (рисунок 10)

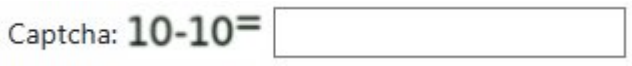


Рисунок 20

3. Кэширование

Кэширование помогает значительно снизить оказываемую на сайт нагрузку.

Документация: https://django.fun/ru/docs/django/4.1/topics/cache/

В Django представлено несколько видов кэширования, в данной лабораторной работе мы остановимся на одном из них, основывающемся на использовании файловой системы.

В этом случае серверная часть на основе файлов сериализует и сохраняет каждое значение кэша как отдельный файл.

Настройка (рисунок 1):

Чтобы использовать этот бэкэнд необходимо установите **BACKEND** на django.core.cache.backends.filebased.FileBasedCache и **LOCATION** в подходящий каталог.

Создайте папку **django_cache** в корне проекта. Тогда ваш путь будет выглядеть следующим образом: **'LOCATION': os.path.join(BASE_DIR, 'django_cache')**. Вы также можете указывать строкой абсолютный путь к

этой папке. На реальном проекте нужно будет проверять права пользователя, чтобы каталог был доступен для чтения и записи.

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.filebased.FileBasedCache',
        'LOCATION': os.path.join(BASE_DIR, 'django_cache'),
    }
}
```

Рисунок 21

Использование кэширования

Кэшировать весь сайт будет иметь смысл только в том случае, если он редко обновляется. В остальных лучше обратиться к другим способам.

Так как в нашей работе в большей степени представлены представления на основе классов, то рассмотрим следующий пример.

Указание кеша для каждого представления в URLconf

Здесь рассмотрим кэширование домашней страницы сайта. По умолчанию для ее формирования используются 5 SQL-запросов.

1. Перейдите в файл urls.py приложения
2. В нем импортируйте **cache_page** из модуля **django.views.decorators.cache**
3. Заключите представление в cache_page, укажите время кэширования в секундах (рисунок 22)

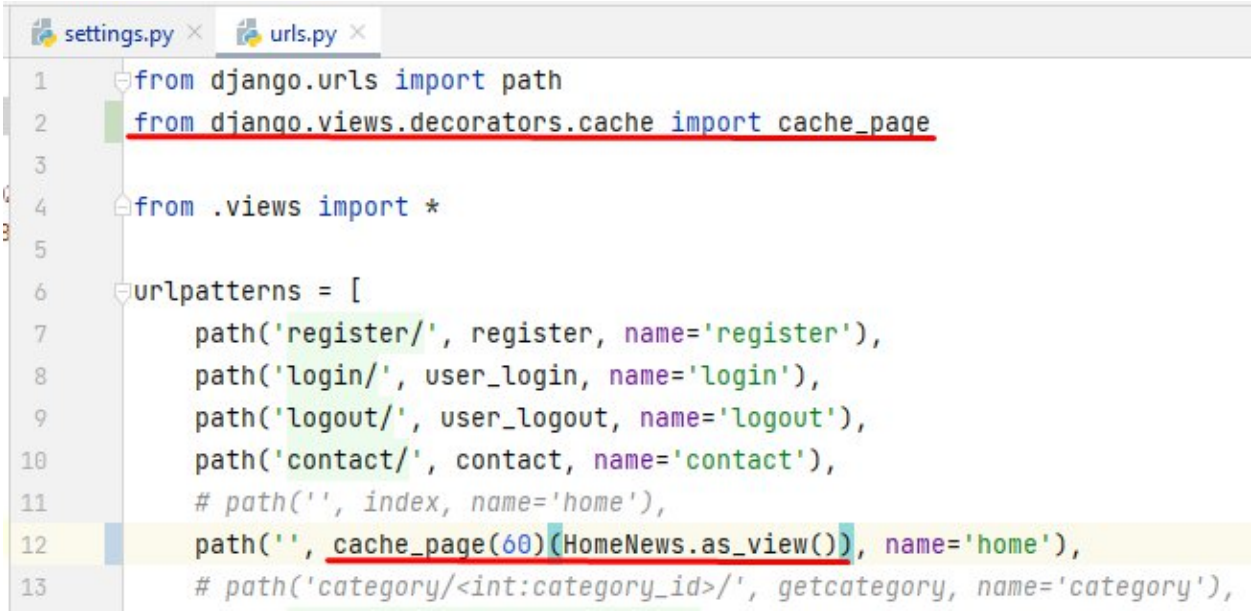


Рисунок 22

С первым обновлением страницы она будет добавлена в папку с кэшем. При повторном мы обнаружим, что количество запросов стало равным нулю. (рисунок 23) Какие бы изменения в страницу мы не внесли, они применятся только по истечении указанного времени.

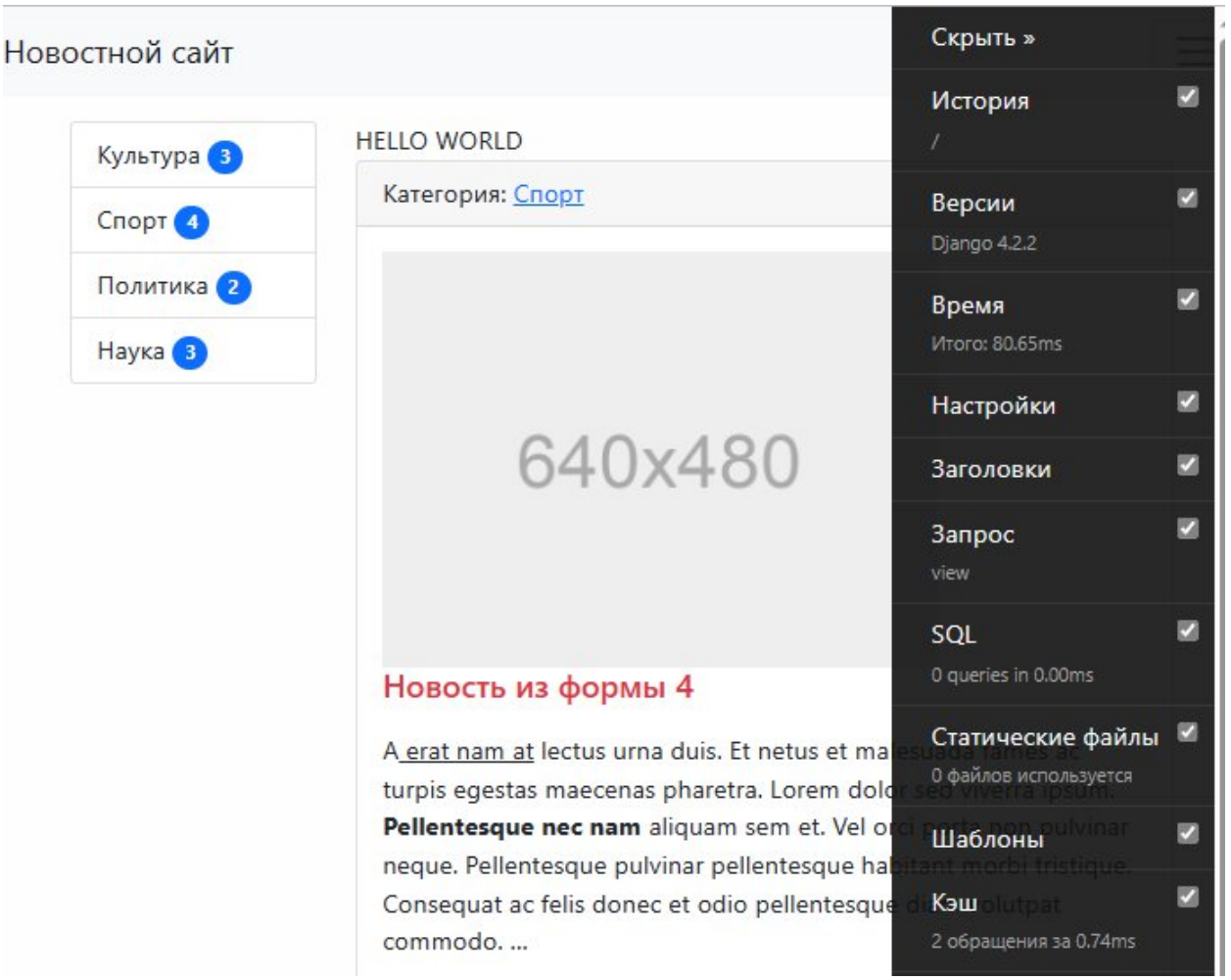


Рисунок 23

Закомментируйте кэширование, вернув код к первоначальному виду.

Кеширование фрагментов шаблона

Также существует возможность кэширования фрагментов шаблона. Рассмотрим ее на примере боковой панели, в ее формировании участвует 1 SQL-запрос.

Откройте базовый шаблон base.html

1. Поместите **{% load cache %}** в верхней части шаблона или непосредственно перед областью применения.
2. Заключите блок sidebar в тег шаблона {% cache %}.
3. Укажите для тега 2 аргумента: тайм-аут кеша в секундах и имя для присвоения фрагмента кеша. (рисунок 24)

```
<div class="container mt-3">
    <div class="row">

        <div class="col-md-3">
            {% load cache %}
            {% cache 40 sidebar %}
            {% block sidebar %}SIDEBAR{% endblock %}
            {% endcache %}
        </div>
```

Рисунок 24

Количество запросов уменьшится на 1, как и ожидалось. (рисунок 25)

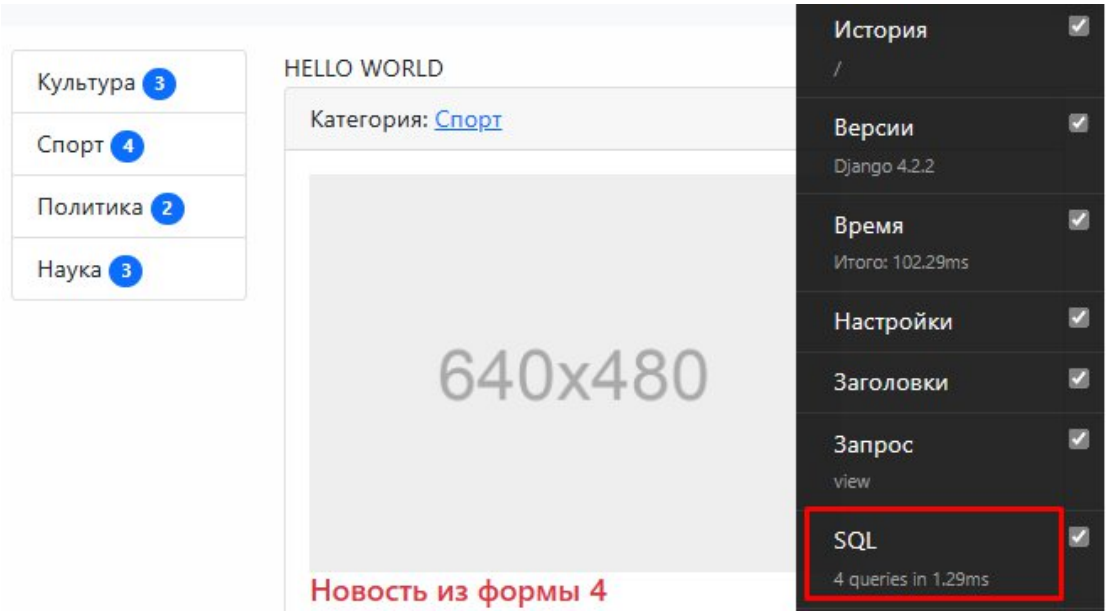


Рисунок 25

Закомментируйте кэширование, вернув код к первоначальному виду.

API низкоуровневого кеширования

В случае, если на вашем сайте есть представление, зависящее от нескольких дорогостоящих запросов, результаты которых меняются с разной периодичностью, предпочтительно не использовать полностраничное кэширование.

Для подобных случаев Django предоставляет простой низкоуровневый API кеширования. Вы можете использовать его для хранения объектов с любым уровнем детализации, который вам нравится.

Рассмотрим его на примере отображения категорий.

Перейдите в файл news/templatetags/news_tags.py

Импортируйте cache из модуля django.core.cache

Здесь мы будет работать с 2 основными методами: set для установки кэша и get для его получения. Приступим к редактированию функции show_categories.

Пробуем с помощью метода get с параметром ключа(указан в set) получить кэш в переменную categories.

Если кэш отсутствует, то получаем данные в переменную categories и устанавливаем кэш методом set, где параметрами передаем ключ(произвольный), объект кэширования и время в секундах. (рисунок 26)

```
@register.inclusion_tag('news/list_categories.html')
def show_categories(arg1='Hello', arg2='world'):
    categories = cache.get('categories')
    if not categories:
        categories = Category.objects.annotate(cnt=Count('news', filter=F('news__is_published'))).filter(cnt__gt=0)
        cache.set('categories', categories, 30)
```

Рисунок 26

В результате, как и в прошлом примере, получим 4 запроса.

Закомментируйте кэширование, вернув код к первоначальному виду.

Рекомендация

Не используйте кэширование на этапе разработки.

БЛОК 1: БАЗОВЫЙ УРОВЕНЬ

Требования: Установить и настроить CKEditor, добавить капчу на форму обратной связи, реализовать базовое кэширование.

Вариант 1

	Задание
	Установить django-ckeditor, добавить в INSTALLED_APPS и

	Задание
	настроить CKEDITOR_UPLOAD_PATH
	В админ-панели заменить стандартное поле content модели News на CKEditorWidget
	Собрать статику через collectstatic и убедиться в работе редактора
	Добавить капчу на форму обратной связи (страница /contact/)
	Настроить файловое кэширование для главной страницы на 60 секунд

Вариант 2

	Задание
	Настроить CKEditorUploadingWidget для возможности загрузки изображений
	Создать новость с изображением через админку, убедиться в сохранении
	Добавить капчу на форму добавления новости (пользовательская часть)
	Настроить кэширование фрагментов шаблона для боковой панели с категориями (время — 120 секунд)
	Проверить через Debug Toolbar, что количество запросов уменьшилось

Вариант 3

	Задание
	В настройках CKEditor расширить панель инструментов (добавить 'Format', 'Table', 'Image', 'Media')
	В админке проверить появление новых кнопок в редакторе
	Установить django-simple-captcha, добавить в INSTALLED_APPS и провести миграции
	Добавить капчу на форму регистрации пользователя
	Настроить кэширование страницы категории через cache_page на 30 секунд

Вариант 4

	Задание
	Настроить CKEditor с возможностью вставки кода (добавить 'CodeBlock' и 'CodeSnippet')
	Создать новость с примером кода Python
	Добавить капчу на страницу добавления комментария к новости
	Настроить низкоуровневое кэширование списка категорий (в кастомном теге show_categories)
	Установить время кэширования 300 секунд

Вариант 5

	Задание
	Настроить CKEditor с ограничением минимальной высоты окна (200px)
	В админке проверить корректность отображения
	Добавить капчу на форму обратной связи с математическим примером (CAPTCHA_CHALLENGE_FUNCT = 'captcha.helpers.math_challenge')
	Настроить кэширование главной страницы на 15 секунд
	Проверить через Debug Toolbar, что запросы кэшируются

Вариант 6

	Задание
	Настроить CKEditor с возможностью загрузки файлов (не только изображений)
	Создать новость с прикреплённым PDF-файлом
	Добавить капчу на форму подписки на новости
	Настроить низкоуровневое кэширование списка популярных новостей (в кастомном теге)
	Настроить отключение ротации букв в капче (CAPTCHA_LETTER_ROTATION = None)

Вариант 7

	Задание
	Установить django-ckeditor, настроить загрузку изображений в папку uploads/news/
	Создать новость с форматированным текстом (заголовки, списки, жирный шрифт)
	Добавить капчу на форму обратной связи с удалением шума (CAPTCHA_NOISE_FUNCTIONS = ())
	Настроить кэширование страницы категории на 45 секунд
	Убедиться в работе кэша через Debug Toolbar

Вариант 8

	Задание
	Настроить CKEditor с цветовой темой (например, 'skin': 'moono-lisa')
	Добавить капчу на форму добавления новости пользователем
	Настроить кэширование боковой панели с категориями на 90 секунд
	Настроить низкоуровневое кэширование для получения 5 последних новостей
	Проверить, что после изменения категории кэш не обновляется (проблема инвалидации)

Вариант 9

	Задание
	Настроить CKEditor с дополнительными стилями форматирования
	Создать новость с таблицей через редактор
	Добавить капчу на страницу восстановления пароля
	Настроить кэширование главной страницы с cache_page и параметром key_prefix
	Проверить, что разные пользователи видят одинаковый кэш

Вариант 10

	Задание
	Установить django-ckeditor, настроить CKEDITOR_CONFIGS для русского языка
	В админке проверить локализацию интерфейса редактора
	Добавить капчу на форму обратной связи, настроить длину капчи в 6 символов
	Настроить кэширование страницы категории с игнорированием GET-параметров
	С помощью Debug Toolbar убедиться в уменьшении нагрузки на БД

Вариант 11

	Задание
	Настроить CKEditor с возможностью вставки видео (YouTube/Vimeo)
	Создать новость с встроенным видео
	Добавить капчу на форму обратной связи с изображением серого цвета (CAPTCHA_BACKGROUND_COLOR = '#cccccc')
	Настроить кэширование списка категорий через cache.set и cache.get
	Установить время жизни кэша 600 секунд

Вариант 12

	Задание
	Настроить CKEditor с отключением автоматической замены смайликов
	Добавить капчу на форму регистрации с математическим выражением
	Настроить кэширование главной страницы на 20 секунд
	Настроить инвалидацию кэша при добавлении новой новости (вручную)
	Проверить, что кэш обновляется после добавления новости

Вариант 13

	Задание
	Настроить CKEditor с возможностью вставки специальных символов
	Добавить капчу на форму добавления комментария с размером 150x50
	Настроить низкоуровневое кэширование для топ-10 популярных новостей
	Настроить кэширование страницы категории через декоратор @cache_page
	Установить время кэширования 25 секунд

Вариант 14

	Задание
	Установить django-ckeditor, настроить CKEDITOR_UPLOAD_PATH = 'uploads/ckeditor/'
	Создать новость с форматированием и изображением
	Добавить капчу на форму обратной связи с увеличенным шрифтом
	Настроить кэширование боковой панели с категориями на 180 секунд
	Проверить, что при изменении категории в админке кэш не сбрасывается

Вариант 15

	Задание
	Настроить CKEditor с кастомной конфигурацией (убрать лишние кнопки, оставив только базовые)
	Добавить капчу на форму обратной связи с фоном (CAPTCHA_BACKGROUND_COLOR = '#eeeeee')
	Настроить кэширование главной страницы на 10 секунд
	Настроить низкоуровневое кэширование для получения количества новостей по категориям
	Собрать статику и убедиться в работе всех настроек

БЛОК 2: СРЕДНИЙ УРОВЕНЬ

Требования: Кастомизация CKEditor, настройка капчи с различными параметрами, реализация кэширования с учётом инвалидации, комбинирование различных видов кэширования.

Вариант 1

	Задание
	Настроить CKEditor с кастомным набором инструментов (добавить 'TextColor', 'BGColor', 'JustifyBlock')
	Создать форму обратной связи с капчей на основе математического выражения

	Задание
	Реализовать кэширование главной страницы с автоматической инвалидацией при добавлении новости (через сигналы)
	Настроить кэширование фрагментов шаблона для списка категорий с разными ключами для авторизованных и анонимов
	Оптимизировать запросы с помощью select_related в дополнение к кэшированию

Вариант 2

	Задание
	Настроить CKEditor с возможностью перетаскивания изображений (drag-and-drop)
	Добавить капчу на форму регистрации с временным ограничением (один запрос в минуту)
	Реализовать низкоуровневое кэширование для страницы категории с обновлением при изменении новостей
	Настроить комбинированное кэширование: главная страница — 60 сек, боковая панель — 300 сек
	Создать кастомную команду для очистки всего кэша

Вариант 3

	Задание
	Настроить CKEditor с добавлением плагина 'uploadimage' для

	Задание
	загрузки изображений без перезагрузки
	Создать кастомную капчу с изображением (своя генерация)
	Реализовать кэширование результатов поиска новостей с учётом поискового запроса
	Настроить инвалидацию кэша категорий при добавлении/удалении новости
	Использовать prefetch_related вместе с кэшированием для оптимизации комментариев

Вариант 4

	Задание
	Настроить CKEditor с интеграцией 'filebrowser' для управления файлами
	Добавить капчу на форму обратной связи с проверкой ввода по таймеру (не менее 3 секунд)
	Реализовать версионное кэширование с использованием select_related в классе HomeNews
	Настроить кэширование с разным временем для разных разделов сайта
	Создать middleware для автоматического кэширования всех GET-запросов

Вариант 5

	Задание
	Настроить CKEditor с подключением внешнего словаря для проверки орфографии
	Создать AJAX-проверку капчи без перезагрузки страницы
	Реализовать кэширование с использованием Redis (установить django-redis)
	Настроить инвалидацию кэша при редактировании новости через post_save сигнал
	Сравнить производительность файлового и Redis-кэширования

Вариант 6

	Задание
	Настроить CKEditor с кастомными стилями для заголовков (H1-H6)
	Добавить капчу на форму обратной связи с аудиовызовом (CAPTCHA_AUDIO_FORMAT = 'mp3')
	Реализовать кэширование с использованием cache_page для всех страниц категорий
	Настроить динамическое обновление кэша при добавлении новой новости
	Использовать django-cachalot для автоматического кэширования

	Задание
	ORM-запросов

Вариант 7

	Задание
	Настроить CKEditor с плагином 'wordcount' для подсчёта символов
	Создать капчу с выбором правильной картинки (из 4 вариантов)
	Реализовать кэширование страницы новости с учётом news_id
	Настроить инвалидацию кэша при изменении новости через post_save сигнал
	Комбинировать кэширование с select_related для страницы категории

Вариант 8

	Задание
	Настроить CKEditor с возможностью создания и редактирования таблиц
	Добавить капчу на форму обратной связи с изменённым количеством символов (6 букв)
	Реализовать кэширование с использованием декоратора @cache_page для всех страниц новостей

	Задание
	Настроить инвалидацию кэша при добавлении комментария
	Использовать prefetch_related для комментариев в кэшированной странице

Вариант 9

	Задание
	Настроить CKEditor с плагином 'autosave' для автоматического сохранения черновиков
	Создать капчу с размытым фоном (CAPTCHA_NOISE_FUNCTIONS = 'captcha.helpers.noise_arcs')
	Реализовать кэширование с использованием django-redis и настройкой TTL
	Настроить кэширование результатов фильтрации новостей по категории
	Создать декоратор для кэширования с автоматической инвалидацией

Вариант 10

	Задание
	Настроить CKEditor с плагином 'emoji' для вставки эмодзи
	Добавить капчу на форму обратной связи с использованием фонового изображения

	Задание
	Реализовать кэширование с использованием cache.get_or_set для списка категорий
	Настроить кэширование с разными префиксами для разных типов пользователей
	Оптимизировать запросы с select_related и prefetch_related вместе с кэшированием

Вариант 11

	Задание
	Настроить CKEditor с плагином 'exportpdf' для экспорта в PDF
	Создать капчу с изменённым размером шрифта (CAPTCHA_FONT_SIZE = 30)
	Реализовать низкоуровневое кэширование с использованием Redis для страницы категории
	Настроить автоматическую инвалидацию кэша при изменении категории
	Сравнить производительность с файловым кэшированием и без кэширования

Вариант 12

	Задание
	Настроить CKEditor с плагином 'mathjax' для вставки формул

	Задание
	Добавить капчу на форму обратной связи с двумя словами вместо одного
	Реализовать кэширование с использованием cache_page с параметром key_prefix='home'
	Настроить кэширование боковой панели с использованием {% cache %} и дополнительного ключа
	Использовать django-cachalot для кэширования сложных запросов с аннотациями

Вариант 13

	Задание
	Настроить CKEditor с плагином 'floating-tools' для плавающей панели инструментов
	Создать капчу с использованием только цифр (CAPTCHA_CHALLENGE_FUNCT = 'captcha.helpers.numeric_challenge')
	Реализовать кэширование с использованием cache_page с учётом GET-параметров пагинации
	Настроить кэширование результатов поиска с учётом страницы
	Комбинировать кэширование с values() для уменьшения объёма кэшируемых данных

Вариант 14

	Задание
	Настроить CKEditor с плагином 'codesnippetgeshi' для подсветки синтаксиса
	Добавить капчу на форму обратной связи с проверкой ввода без учёта регистра
	Реализовать кэширование с использованием cache.set_many и cache.get_many для списка категорий
	Настроить инвалидацию кэша при массовых операциях (например, удалении нескольких новостей)
	Использовать select_related в кэшированных запросах для уменьшения времени десериализации

Вариант 15

	Задание
	Настроить CKEditor с кастомным плагином для вставки собственных блоков
	Создать капчу с проверкой на стороне клиента и сервера
	Реализовать кэширование с использованием cache.incr для счётчика просмотров страниц
	Настроить комбинированное кэширование для страницы новости (фрагменты + полностраничное)
	Оптимизировать кэширование с учётом авторизации пользователя

	Задание
	(разные кэши для анонимов и авторизованных)

БЛОК 3: ПРОДВИНУТЫЙ УРОВЕНЬ

Требования: Сложная кастомизация CKEditor, создание собственной капчи, продвинутые стратегии кэширования (инвалидация, версионирование, Redis), интеграция с кэшированием ORM-запросов.

Вариант 1

	Задание
	Настроить CKEditor с поддержкой вставки QR-кодов через API
	Разработать собственную капчу (генерация случайного слова, сохранение в сессии)
	Реализовать кэширование с инвалидацией по тегам (используя django-cacheops)
	Настроить кэширование результатов работы Paginator для больших списков
	Оптимизировать кэширование с использованием connection.queries для анализа запросов

Вариант 2

	Задание
	Настроить CKEditor с интеграцией серверного файлового менеджера

	Задание
	Создать капчу с движущимися элементами (анимированная GIF)
	Реализовать кэширование с использованием Redis с настройкой кластеризации
	Настроить кэширование страниц с использованием django-cache-machine
	Реализовать асинхронную инвалидацию кэша через Celery

Вариант 3

	Задание
	Настроить CKEditor с поддержкой совместного редактирования (real-time collaboration)
	Создать капчу на основе распознавания объектов на изображении
	Реализовать кэширование с использованием django-cachalot и кастомных правил инвалидации
	Настроить кэширование с разной стратегией для разных типов запросов (GET/POST)
	Реализовать мониторинг попаданий в кэш через Prometheus

Вариант 4

	Задание
	Настроить CKEditor с возможностью вставки 3D-моделей (Three.js

	Задание
	интеграция)
	Создать капчу с использованием голосового ввода
	Реализовать гибридное кэширование (файловое + Redis) с автоматическим переключением
	Настроить кэширование с использованием django-cache-utils для сложных запросов
	Оптимизировать кэширование для сайтов с высокой нагрузкой (более 1000 запросов/сек)

Вариант 5

	Задание
	Настроить CKEditor с поддержкой drag-and-drop для сортировки изображений
	Создать капчу на основе биометрических данных (тайминг нажатия клавиш)
	Реализовать кэширование с использованием django-redis-cache и сжатием данных
	Настроить кэширование с использованием django-cacheops и автоматической инвалидацией
	Реализовать распределённое кэширование для микросервисной архитектуры

Вариант 6

	Задание
	Настроить CKEditor с поддержкой WebDAV для работы с облачными хранилищами
	Создать адаптивную капчу (изменяющуюся в зависимости от поведения пользователя)
	Реализовать кэширование с использованием django-cacheops и денормализацией данных
	Настроить кэширование с использованием django-cachalot и кастомных транзакций
	Реализовать кэширование для GraphQL API (если используется)

Вариант 7

	Задание
	Настроить CKEditor с поддержкой вставки SVG-анимаций
	Создать капчу на основе ReCaptcha v3 (интеграция с Google)
	Реализовать кэширование с использованием django-cache-machine с настраиваемым TTL
	Настроить кэширование с использованием django-cachalot для связанных запросов
	Реализовать предиктивную загрузку кэша на основе пользовательских сессий

Вариант 8

	Задание
	Настроить CKEditor с поддержкой вставки диаграмм Mermaid.js
	Создать капчу с использованием поведенческого анализа (mouse tracking)
	Реализовать кэширование с использованием django-cacheops с автоматической денормализацией
	Настроить кэширование с использованием django-cachalot с кастомными индексами
	Реализовать кэширование для мультиязычных сайтов (с учётом локали)

Вариант 9

	Задание
	Настроить CKEditor с поддержкой вставки интерактивных карт (OpenStreetMap)
	Создать капчу на основе распознавания аудиопоследовательностей
	Реализовать кэширование с использованием django-cacheops с партиционированием
	Настроить кэширование с использованием django-cachalot с временными таблицами

	Задание
	Реализовать адаптивное кэширование (изменение стратегии на основе нагрузки)

Вариант 10

	Задание
	Настроить CKEditor с поддержкой криптографической подписи вставляемых данных
	Создать капчу на основе блокчейн-вызовов (web3)
	Реализовать кэширование с использованием django-cacheops с репликацией
	Настроить кэширование с использованием django-cachalot с сжатием данных
	Реализовать предварительное заполнение кэша на основе аналитики

Вариант 11

	Задание
	Настроить CKEditor с поддержкой вставки математических формул (LaTeX)
	Создать капчу на основе решения капчи-пазла
	Реализовать кэширование с использованием django-cacheops с шардированием

	Задание
	Настроить кэширование с использованием django-cachalot с кастомным сериализатором
	Реализовать кэширование с учётом географического расположения пользователя

Вариант 12

	Задание
	Настроить CKEditor с поддержкой вставки графиков (Chart.js)
	Создать капчу на основе решения логической задачи
	Реализовать кэширование с использованием django-cacheops с кастомным разрешением конфликтов
	Настроить кэширование с использованием django-cachalot с кастомным кэш-бэкендом
	Реализовать кэширование для SPA с использованием Service Workers

Вариант 13

	Задание
	Настроить CKEditor с поддержкой вставки видео конференций (Jitsi)
	Создать капчу на основе распознавания эмоций по лицу

	Задание
	Реализовать кэширование с использованием django-cacheops с автоматической оптимизацией
	Настроить кэширование с использованием django-cachalot с кастомным анализом запросов
	Реализовать гибридное кэширование для гибридных приложений (SSR + SPA)

Вариант 14

	Задание
	Настроить CKEditor с поддержкой вставки дополненной реальности (AR)
	Создать капчу на основе распознавания жестов через веб-камеру
	Реализовать кэширование с использованием django-cacheops с автоматической балансировкой
	Настроить кэширование с использованием django-cachalot с кастомным кэш-менеджментом
	Реализовать кэширование для PWA с офлайн-режимом

Вариант 15

	Задание
	Настроить CKEditor с поддержкой вставки голосовых сообщений (WebRTC)

	Задание
	Создать капчу на основе распознавания намерений (NLP)
	Реализовать кэширование с использованием django-cacheops с ИИ-оптимизацией
	Настроить кэширование с использованием django-cachalot с кастомным предиктором
	Реализовать самообучающееся кэширование на основе ML-модели

Контрольные вопросы

Базовый уровень

1. Что такое CKEditor и для чего он используется в Django?
2. Как установить и настроить django-ckeditor?
3. Чем отличается CKEditorWidget от CKEditorUploadingWidget?
4. Как загружать изображения через CKEditor?
5. Что такое капча и для каких целей она используется?
6. Как установить django-simple-captcha?
7. Как добавить поле капчи в форму Django?
8. Что такое кэширование и какие задачи оно решает?
9. Какие виды кэширования поддерживает Django?
10. Как настроить файловое кэширование в Django?

Средний уровень

1. Как расширить панель инструментов CKEditor?
2. Как изменить тип капчи (например, на математический пример)?
3. Как убрать шум на изображении капчи?
4. В чем разница между cache_page, {% cache %} и низкоуровневым API?
5. Как кэшировать страницу на определённое время?
6. Как кэшировать только часть шаблона?
7. Как использовать низкоуровневое кэширование для списка категорий?
8. Как проверить эффективность кэширования через Debug Toolbar?
9. Как настроить ротацию букв в капче?
10. Как изменить цвет фона капчи?

Продвинутый уровень

1. Как создать собственный плагин для CKEditor?
2. Как реализовать кастомную капчу (своя логика генерации)?
3. Как реализовать инвалидацию кэша при изменении данных?
4. Как настроить кэширование с использованием Redis?
5. Как использовать django-cachalot для автоматического кэширования ORM-запросов?
6. Как реализовать версионное кэширование для инвалидации?
7. Как кэшировать результаты пагинации?
8. Как кэшировать с учётом авторизации пользователя?
9. Как настроить кэширование для API-эндпоинтов DRF?
10. Как реализовать гибридное кэширование (файловое + Redis)?

ИТОГОВЫЕ КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	**Базовый уровень (max 35)**	**Средний уровень (max 50)**	**Продвинутый уровень (max 70)**
Установка и настройка CKEditor	5	5	5
Кастомизация панели инструментов CKEditor	—	10	10
Интеграция CKEditor в админ-панель	5	5	5
Установка и настройка капчи	5	5	5

Критерий	Базовый уровень (max 35)	Средний уровень (max 50)	Продвинутый уровень (max 70)
Кастомизация параметров капчи	—	10	10
Создание собственной капчи	—	—	15
Настройка файлового кэширования	10	—	—
Кэширование фрагментов шаблона	5	10	10
Низкоуровневое кэширование	5	10	10
Инвалидация кэша	—	5	10
Redis и продвинутые стратегии кэширования	—	—	15
Итого	35	60	95

Шкала перевода баллов в оценку

Оценка	Базовый уровень (35)	Средний уровень (60)	Продвинутый уровень (95)
5 (отлично)	32-35	55-60	86-95
4 (хорошо)	28-31	48-54	76-85
3 (удовлетворительно)	22-27	42-47	66-75
2 (неудовлетворительно)	0-21	0-41	0-65

Лабораторная работа №15.

Создание React-проекта и настройка роутинга

Цель работы:

Освоить создание одностраничного приложения (SPA) на React с использованием маршрутизации. Научиться создавать React-проект через Vite, устанавливать и настраивать React Router, создавать компоненты-страницы и организовывать навигацию между ними.

Теоретическое обоснование

Что такое React и зачем он нужен?

React - это библиотека для создания пользовательских интерфейсов. Представьте, что ваш сайт - это дом, а React - это набор инструментов,

который помогает строить этот дом из кирпичиков-компонентов. Каждый компонент - это самостоятельная часть интерфейса: шапка, меню, карточка новости, кнопка.

В чём отличие React от обычного HTML? В обычном HTML страница перезагружается целиком при каждом переходе. В React страница НЕ перезагружается - меняется только та часть экрана, которая нужна. Это работает быстрее и плавнее.

Простая аналогия: Представьте, что обычный сайт - это книга. Вы переворачиваете страницу - и всё исчезает, появляется новая. А React-сайт - это лента в социальной сети. Вы листаете, а картинки и текст просто подгружаются, не заставляя ждать.

Что такое SPA (Single Page Application)?

SPA - это приложение, которое загружает одну HTML-страницу и дальше работает без перезагрузок. Это как игра, которая один раз загрузилась и дальше просто меняет уровни, не выходя из программы.

Плюсы SPA:

Быстрая работа - нет ожидания перезагрузки страницы

Плавные переходы - анимации не прерываются

Экономия трафика - загружается только новая информация, не вся страница

Минусы:

- Первая загрузка может быть чуть дольше
- Сложнее настраивать поисковые системы (SEO)
- Что такое роутинг (маршрутизация)?

Роутинг - это механизм, который понимает, какой компонент показывать при переходе по определённому адресу.

Аналогия: Роутинг работает как табличка в больнице. Вы заходите - видите список кабинетов. «Кардиология - 2 этаж, кабинет 5». Вы идёте по адресу и попадаете в нужное место. В React роутинг делает то же самое:

видит URL /news - показывает компонент новостей, видит /profile - показывает профиль.

Что такое Vite и почему мы используем его вместо Create React App?

Vite - это инструмент для быстрого создания и сборки проектов. Раньше все использовали Create React App (CRA). Но CRA работает медленно, особенно в больших проектах. Vite работает почти мгновенно.

Сравнение Vite и CRA:

Что сравниваем	**Vite**	**Create React App (CRA)**
Время запуска проекта	1 - 2 секунды	10-30 секунд
Обновление при изменении кода	Мгновенное	1-5 секунд
Размер итогового файла	Меньше	Больше
Сложность настройки	Простая	Простая

Вывод: Vite - современный стандарт, будем использовать его.

Что такое React Router?

React Router - это библиотека, которая добавляет роутинг в React-приложение. Без него React не умеет переключать страницы по изменению URL. Он нужен почти в каждом реальном приложении.

Основные компоненты React Router:

Компонент	Что делает	Пример
BrowserRouter	Оборачивает всё приложение, включает роутинг	<BrowserRouter><App /></BrowserRouter>
Routes	Контейнер для всех маршрутов	<Routes>...</Routes>
Route	Один маршрут - связывает путь и компонент	<Route path="/" element={<Home />} />
Link	Ссылка для перехода (без перезагрузки)	<Link to="/about">О нас</Link>
NavLink	То же, что Link, но активная ссылка подсвечивается	<NavLink to="/">Главная</NavLink>

Практический пример

Шаг 1. Создание проекта через Vite

Что мы делаем: Создаём папку с новым React-проектом и подготавливаем его к работе.

Команды, которые нужно ввести в терминал:

bash

```
# Шаг 1.1. Создаём проект с именем "news-frontend"
npm create vite@latest news-frontend -- --template react

# Шаг 1.2. Переходим в папку проекта
cd news-frontend
```

```
# Шаг 1.3. Устанавливаем зависимости
npm install

# Шаг 1.4. Устанавливаем React Router
npm install react-router-dom

# Шаг 1.5. Запускаем проект для проверки
npm run dev
```

Подробное объяснение каждой команды:

- npm create vite@latest news-frontend -- --template react
- npm create vite@latest - запускает Vite и загружает его последнюю версию
- news-frontend - имя папки, в которой создастся проект (назовите так, чтобы было понятно, что это фронтенд для новостного сайта)
- -- --template react - указываем, что нужен шаблон для React (без этого шаблона пришлось бы всё настраивать вручную)
- cd news-frontend - переходим во вновь созданную папку. Без этого мы будем выполнять команды не в том месте.
- npm install - устанавливает все необходимые пакеты, которые нужны для работы React (эта команда читает файл package.json и скачивает всё, что там написано).
- npm install react-router-dom - устанавливает библиотеку для маршрутизации. Пока без неё мы сможем создать только одну страницу.
- npm run dev - запускает сервер для разработки. После этого в терминале появится ссылка типа http://localhost:5173. Нажмите на неё (Ctrl + клик), и откроется браузер с вашим новым приложением.

Что должно появиться в браузере: Страница с вращающимся логотипом Vite и React, и текстом «Hello World». Это значит, что всё работает правильно.

Остановить сервер: Нажмите Ctrl + C в терминале. Потом можно снова запустить командой npm run dev.

Шаг 2. Изучаем структуру проекта

Что мы делаем: Смотрим, какие файлы создал Vite, и понимаем, за что отвечает каждый.

```
news-frontend/
├── index.html          # Главная HTML-страница (там есть <div id="root">)
├── package.json        # Список всех установленных пакетов
├── vite.config.js      # Настройки Vite (пока трогать не будем)
├── public/             # Статические файлы (иконки, шрифты)
└── src/                # Тут живёт весь наш код
    ├── main.jsx        # Точка входа - здесь React подключается к HTML
    ├── App.jsx         # Главный компонент - тут будет всё приложение
    ├── App.css         # Стили для App
    ├── index.css       # Глобальные стили
    └── assets/         # Картинки и другие файлы
```

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App.jsx'
import './index.css'

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
)
```

Пояснение кода:

1. ReactDOM.createRoot(document.getElementById('root')) - находит в index.html элемент с id="root" (это пустой контейнер)

2. .render(<App />) - говорит: «Вставь сюда компонент App»
3. <React.StrictMode> - специальный режим, который помогает находить ошибки

Файл src/App.jsx - главный компонент. Пока там что-то вроде вращающегося логотипа. Наша задача - переписать его так, чтобы там было навигационное меню и несколько страниц.

Шаг 3. Создаём папку для страниц (pages)

Что мы делаем: Организуем файлы, чтобы не было путаницы. Все компоненты-страницы будем хранить в папке pages.

Действия в PyCharm или другой среде:

- Внутри папки src создайте новую папку с именем pages
- Внутри pages создайте файлы (щелчок правой кнопкой → New → File):
- Home.jsx - главная страница
- NewsPage.jsx - страница со списком новостей
- Contacts.jsx - страница контактов
- NotFound.jsx - страница 404 (если адрес не найден)

В итоге структура станет такой:

```
src/
├── pages/
│   ├── Home.jsx
│   ├── NewsPage.jsx
│   ├── Contacts.jsx
│   └── NotFound.jsx
├── App.jsx
├── main.jsx
└── ...
```

Почему это важно: Когда проект вырастет, будет понятно, где искать компоненты страниц. Если держать всё в одном файле App.jsx, он превратится в кашу из тысяч строк.

Шаг 4. Создаём компоненты-страницы

Что мы делаем: Пишем простые компоненты, которые показывают текст. Позже мы заменим текст на реальные новости из API, но пока достаточно понять, как переключаются страницы.

```
const Home = () => {
  return (
    <div>
      <h1>Добро пожаловать!</h1>
      <p>Это главная страница нашего новостного сайта.</p>
      <p>Здесь позже будет список новостей.</p>
    </div>
  );
};

export default Home
```

Объяснение кода:

const Home = () => { ... } - мы создаём стрелочную функцию. В React компонент - это функция, которая возвращает HTML-подобный код (JSX).

... - то, что React покажет на экране. Обратите внимание, что у компонента должен быть один корневой элемент (обёртка).

export default Home; - делает компонент доступным для других файлов. Без этой строчки мы не сможем импортировать Home в App.jsx.

Файл src/pages/NewsPage.jsx:

```
const NewsPage = () => {
  return (
```

```
      <div>
        <h1>Новости</h1>
        <p>Здесь будет список всех новостей.</p>
      </div>
    );
};

export default NewsPage;
```

Файл src/pages/Contacts.jsx:

```
const Contacts = () => {
  return (
    <div>
      <h1>Контакты</h1>
      <p>Свяжитесь с нами по email: news@example.com</p>
      <p>Телефон: +7 (123) 456-78-90</p>
    </div>
  );
};

export default Contacts;
```

Файл src/pages/NotFound.jsx:

```
const NotFound = () => {
  return (
    <div>
      <h1>404</h1>
      <p>Страница не найдена. Проверьте правильность адреса.</p>
      <a href="/">Вернуться на главную</a>
    </div>
  );
};
```

```
export default NotFound;
```

Шаг 5. Создаём компонент навигации (Navbar)

Что мы делаем: Создаём меню, которое будет показываться на всех страницах. В нём будут ссылки для перехода.

Действия: Создайте папку src/components и внутри неё файл Navbar.jsx.

Файл src/components/Navbar.jsx:

```
import { Link, NavLink } from 'react-router-dom';

const Navbar = () => {
  return (
    <nav style={{
      padding: '10px 20px',
      backgroundColor: '#333',
      display: 'flex',
      gap: '20px',
      marginBottom: '20px'
    }}>
      <NavLink
        to="/"
        style={({ isActive }) => ({
          color: isActive ? '#ff6b6b' : 'white',
          textDecoration: 'none'
        })}
      >
        Главная
      </NavLink>
      <NavLink
        to="/news"
        style={({ isActive }) => ({
          color: isActive ? '#ff6b6b' : 'white',
          textDecoration: 'none'
```

```
          }}}
        >
          Новости
        </NavLink>
        <NavLink
          to="/contacts"
          style={({ isActive }) => ({
            color: isActive ? '#ff6b6b' : 'white',
            textDecoration: 'none'
          })}
        >
          Контакты
        </NavLink>
      </nav>
    );
};

export default Navbar;
```

Подробное объяснение:

- import { Link, NavLink } from 'react-router-dom' - импортируем компоненты для ссылок
- Link - обычная ссылка (как <a>, но без перезагрузки страницы)
- NavLink - улучшенный Link, который знает, активна ли сейчас ссылка
- style={({ isActive }) => ({...})} - если ссылка активна (пользователь на этой странице), то красим её в красный цвет (#ff6b6b), иначе - в белый
- Внутри nav мы разместили три ссылки: Главная (/), Новости (/news), Контакты (/contacts)

Шаг 6. Настройка маршрутов в App.jsx

Что мы делаем: Объединяем всё в главном компоненте. Говорим React Router: «Когда пользователь переходит по адресу / - покажи Home, по адресу /news - покажи NewsPage».

Файл src/App.jsx (полностью переписываем его):

```
import { BrowserRouter, Routes, Route } from 'react-router-dom';
import Navbar from './components/Navbar';
import Home from './pages/Home';
import NewsPage from './pages/NewsPage';
import Contacts from './pages/Contacts';
import NotFound from './pages/NotFound';

function App() {
  return (
    <BrowserRouter>
      <Navbar />
      <div style={{ padding: '20px' }}>
        <Routes>
          <Route path="/" element={<Home />} />
          <Route path="/news" element={<NewsPage />} />
          <Route path="/contacts" element={<Contacts />} />
          <Route path="*" element={<NotFound />} />
        </Routes>
      </div>
    </BrowserRouter>
  );
}

export default App;
```

Разбор строк:

Строка	Что делает
<BrowserRouter>	Оборачивает всё приложение и включает роутинг
<Navbar />	Вставляет наше меню на каждой странице
<Routes>	Место, где будут переключаться страницы
<Route path="/" element={<Home />} />	Когда URL = http://localhost:5173/ → показываем компонент Home
<Route path="/news" element={<NewsPage />} />	Когда URL заканчивается на /news → показываем NewsPage
<Route path="/contacts" element={<Contacts />} />	Когда URL заканчивается на /contacts → показываем Contacts
<Route path="*" element={<NotFound />} />	На любой другой адрес → показываем страницу 404

Шаг 7. Запуск и проверка

Запустите проект командой:

```
npm run dev
```

Что нужно проверить в браузере:

Открылась страница http://localhost:5173/

Вы видите меню: Главная, Новости, Контакты

При клике на «Новости» URL меняется на /news и появляется заголовок «Новости»

При клике на «Контакты» URL меняется на /contacts и появляется страница контактов

При вводе несуществующего адреса (например, /abc) появляется страница 404

Если всё работает - отлично! Вы создали своё первое React-приложение с маршрутизацией.

Индивидуальные задания (3 уровня)

Базовый уровень

Требования: Создать приложение с 3-4 статическими страницами и навигационным меню. Активная ссылка должна подсвечиваться.

Вариант	Страницы (путь → название)
1	/ → Главная, /news → Новости, /about → О сайте
2	/ → Главная, /catalog → Каталог, /delivery → Доставка
3	/ → Главная, /services → Услуги, /price → Цены
4	/ → Главная, /blog → Блог, /authors → Авторы
5	/ → Главная, /portfolio → Портфолио, /reviews → Отзывы
6	/ → Главная, /courses → Курсы, /teachers → Преподаватели

Вариант	Страницы (путь → название)
7	/ → Главная, /gallery → Галерея, /events → События
8	/ → Главная, /recipes → Рецепты, /ingredients → Ингредиенты
9	/ → Главная, /movies → Фильмы, /actors → Актеры
10	/ → Главная, /games → Игры, /downloads → Загрузки
11	/ → Главная, /health → Здоровье, /advice → Советы
12	/ → Главная, /sport → Спорт, /team → Команда
13	/ → Главная, /music → Музыка, /artists → Исполнители
14	/ → Главная, /tours → Туры, /countries → Страны
15	/ → Главная, /education → Образование, /career → Карьера

Средний уровень

Требования: Создать приложение с динамическими маршрутами (параметрами в URL). На одной из страниц должен быть список элементов

(массив данных внутри компонента). При клике на элемент должен открываться детальный просмотр с параметром :id.

Вариант	Тип элементов	Какие поля
1	Города	название, население, страна
2	Сотрудники	имя, должность, опыт
3	Товары	название, цена, описание
4	Статьи	заголовок, автор, дата
5	Фильмы	название, режиссёр, год
6	Книги	название, автор, год
7	Игры	название, жанр, рейтинг
8	Рестораны	название, кухня, рейтинг
9	Университеты	название, специальности, балл

Вариант	Тип элементов	Какие поля
10	Спортсмены	имя, вид спорта, достижения
11	Языки программирования	название, год создания, создатель
12	Автомобили	марка, модель, год, цена
13	Художники	имя, эпоха, известные работы
14	Монеты	страна, номинал, год выпуска
15	Парки	название, город, площадь

Пример для варианта 3 (товары):

text

/todos → список товаров

/todos/1 → страница товара с id=1

/todos/2 → страница товара с id=2

Продвинутый уровень (15 вариантов)

Требования: Создать приложение с вложенными маршрутами, хлебными крошками и защитой маршрута (имитация авторизации).

Вариант	Тип приложения	Структура URL
1	Интернет-магазин	/catalog, /catalog/:category, /catalog/:category/:id
2	Блог	/posts, /posts/:id, /posts/:id/comments
3	Доска объявлений	/ads, /ads/:category, /ads/:id
4	Портфолио	/projects, /projects/:id, /projects/:id/gallery
5	Образовательная платформа	/courses, /courses/:id, /courses/:id/lessons
6	Система тикетов	/tickets, /tickets/:id, /tickets/:id/comments
7	Расписание	/schedule, /schedule/:day, /schedule/:day/:time
8	Социальная сеть	/profile, /profile/:id, /profile/:id/friends
9	Новостной сайт	/news, /news/:id, /news/:id/comments
10	Форум	/categories, /categories/:id/

Вариант	Тип приложения	Структура URL
		topics, /topics/:id
11	CRM	/clients, /clients/:id, /clients/:id/orders
12	Админ-панель	/dashboard, /dashboard/users, /dashboard/settings
13	Дашборд	/stats, /stats/daily, /stats/monthly
14	Агрегатор новостей	/sources, /sources/:id/news, /news/:id
15	Чат-приложение	/chats, /chats/:id, /chats/:id/messages

Контрольные вопросы

1. Что такое SPA и чем оно отличается от обычного многостраничного сайта?
2. Для чего нужен React Router?
3. Какая разница между тегом <a> и компонентом <Link> из React Router?
4. Как сделать так, чтобы активная ссылка в меню подсвечивалась?
5. Как создать страницу 404 (для несуществующих адресов)?
6. Как получить параметр из URL (например, id товара) с помощью хука useParams?

7. Для чего нужен хук useNavigate и как с его помощью вернуться на предыдущую страницу?
8. Как сделать так, чтобы при клике на карточку товара открывалась страница с его детальным описанием?
9. Как передать данные между страницами через параметры URL?
10. Как создать вложенные маршруты с помощью Outlet?
11. Что такое хлебные крошки и как их реализовать в React Router v6?
12. Как защитить маршрут от неавторизованных пользователей (создать PrivateRoute)?
13. В чём разница между Navigate и useNavigate?

Требования к отчёту

Титульный лист (название работы, ФИО, группа, вариант)

Цель работы (формулировка из лабораторной работы)

Теоретическое обоснование (2-3 абзаца своими словами - что узнали)

Выполнение практического примера (код всех файлов с комментариями, скриншоты)

Выполнение индивидуального задания (код, скриншоты)

Ответы на контрольные вопросы (не менее 5 по своему уровню)

Вывод (что сделали, что нового узнали, с какими трудностями столкнулись)

Критерии оценивания

Критерий	Базовый уровень (max 35)	Средний уровень (max 50)	Продвинутый уровень (max 65)
Создание	5	5	5

Критерий	Базовый уровень (max 35)	Средний уровень (max 50)	Продвинутый уровень (max 65)
проекта через Vite			
Настройка базовых маршрутов	10	10	5
Активные ссылки (NavLink)	5	5	5
Страница 404	5	5	5
Динамические маршруты (useParams)	—	10	5
Вложенные маршруты (Outlet)	—	—	10
Защита маршрутов (имитация)	—	—	10
Хлебные крошки	—	—	5
Качество кода и комментарии	5	10	10

Критерий	Базовый уровень (max 35)	Средний уровень (max 50)	Продвинутый уровень (max 65)
Ответы на вопросы	5	5	5
Итого	35	50	65

Шкала перевода баллов в оценку

Оценка	Базовый (35)	Средний (50)	Продвинутый (65)
5 (отлично)	32-35	46-50	59-65
4 (хорошо)	28-31	40-45	51-58
3 (удовлетворительно)	22-27	32-39	42-50
2 (неудовлетворительно)	0-21	0-31	0-41

Лабораторная работа №16

Управление состоянием с Context API

1. Цель работы

Научиться управлять глобальным состоянием в React-приложениях с использованием Context API. Освоить создание контекста, провайдера,

использование хука useContext, а также создание кастомных хуков для удобной работы с контекстом.

2. Теоретическое обоснование

2.1. Что такое состояние (state) в React?

Состояние (state) - это данные, которые могут меняться в процессе работы приложения и влиять на то, что пользователь видит на экране.

Примеры состояния:

- Список новостей (загрузились → показались на экране)
- Авторизован пользователь или нет (показать кнопку «Выйти» или «Войти»)
- Тема оформления (светлая/тёмная)
- Открыта форма добавления комментария или скрыта

2.2. Локальное vs глобальное состояние

Тип состояния	Где используется	Пример
Локальное	Внутри одного компонента	Открыто ли меню на телефоне, значение поля ввода
Глобальное	Во многих компонентах по всему приложению	Данные пользователя, тема оформления

Простая аналогия:

Локальное состояние - ваша личная заметка в телефоне (только вы её видите)

Глобальное состояние - объявление на доске в офисе (видят все сотрудники)

2.3. Проблема «проброса пропсов» (prop drilling)

Проблема: Данные нужно передать из компонента A в компонент E, который находится глубоко внутри. Приходится передавать пропсы через каждый промежуточный компонент, хотя эти компоненты данные не используют.

Аналогия: Вы хотите передать книгу соседу на пятом этаже, но лифт не работает. Приходится просить жильцов с первого по четвёртый этаж передавать книгу дальше, хотя им эта книга не нужна.

Пример проблемы (без Context API):

```
// Глубоко вложенный компонент, которому нужна тема
const DeepComponent = ({ theme }) => {
  return <div style={{ color: theme === 'dark' ? 'white' : 'black' }}>Привет!</div>;
};

// Промежуточный компонент, который просто передаёт тему дальше
const MiddleComponent = ({ theme }) => {
  return <DeepComponent theme={theme} />;
};

// Родительский компонент, который знает тему
const ParentComponent = () => {
  const [theme, setTheme] = useState('light');
  return <MiddleComponent theme={theme} />;
};
```

MiddleComponent не использует тему, но вынужден её принимать и передавать дальше.

2.4. Что такое Context API?

Context API - это встроенный в React механизм для передачи данных через любую вложенность без проброса пропсов.

Как это работает:

Создаётся контекст (контейнер для данных)

Данные кладутся в специальный компонент-провайдер

Любой компонент (на любой глубине) может подписаться на контекст и получить данные

2.5. Компоненты Context API

Компонент/ Хук	Что делает	Как используется
createContext()	Создаёт новый контекст	const ThemeContext = createContext('light')
Context.Provider	Даёт значение контекста всем вложенным компонентам	<ThemeContext.Provider value="dark">
useContext()	Хук для чтения значения контекста	const theme = useContext(ThemeContext)

2.6. Когда использовать Context API

Ситуация	Решение	Почему
Данные нужны только родителю и одному ребёнку	Пропсы	Проще, код понятнее
Данные нужны нескольким компонентам на разной глубине	Context	Не нужно передавать через 5 уровней
Данные нужны почти всем компонентам	Context	Удобно и чисто

3. Практический пример

Что мы будем делать: Создадим приложение с переключением темы (светлая/тёмная). Сначала покажем проблему проброса пропсов, а затем решим её с помощью Context API.

Шаг 1. Создание проекта (если ещё не создали)

```
npm create vite@latest context-demo -- --template react
cd context-demo
npm install
npm run dev
```

Шаг 2. Создаём структуру папок

```
src/
├── components/
│   ├── Toolbar.jsx          # Панель инструментов (будет содержать кнопку)
│   └── ThemedButton.jsx # Кнопка, цвет которой зависит от темы
```

```
├── contexts/
│   └── ThemeContext.jsx # Контекст темы
├── App.jsx
└── main.jsx
```

Шаг 3. Сначала покажем проблему (без Context API)

Файл src/App.jsx (первая версия - с пробросом пропсов):

```jsx
import { useState } from 'react';

// Глубоко вложенный компонент, которому нужна тема
const ThemedButton = ({ theme }) => {
  return (
    <button style={{
      backgroundColor: theme === 'dark' ? '#333' : '#eee',
      color: theme === 'dark' ? '#fff' : '#000',
      padding: '10px 20px',
      border: 'none',
      borderRadius: '5px',
      cursor: 'pointer'
    }}>
      Кнопка в {theme} теме
    </button>
  );
};

// Промежуточный компонент (просто передаёт тему дальше)
const Toolbar = ({ theme }) => {
  return (
```

```
    <div style={{ padding: '20px' }}>
      <h3>Панель инструментов</h3>
      <ThemedButton theme={theme} />
    </div>
  );
};

function App() {
  const [theme, setTheme] = useState('light');

  const toggleTheme = () => {
    setTheme(theme === 'light' ? 'dark' : 'light');
  };

  return (
    <div>
      <h1>Проблема проброса пропсов</h1>
      <button onClick={toggleTheme}>
        Переключить тему (сейчас {theme})
      </button>
      <Toolbar theme={theme} />
    </div>
  );
}

export default App;
```

Что здесь плохо: Компонент Toolbar не использует тему, но вынужден её принимать и передавать в ThemedButton. Если у нас будет 10 уровней вложенности, придётся передавать тему через все 10.

Шаг 4. Создаём контекст

Файл src/contexts/ThemeContext.jsx:

```
import { createContext, useState, useContext } from 'react';

// 1. Создаём контекст
const ThemeContext = createContext();

// 2. Создаём провайдер (компонент, который даёт значение контекста)
export const ThemeProvider = ({ children }) => {
  const [theme, setTheme] = useState('light');

  const toggleTheme = () => {
    setTheme(theme === 'light' ? 'dark' : 'light');
  };

  // Всё, что нужно передать в контекст
  const value = {
    theme,
    toggleTheme
  };

  return (
    <ThemeContext.Provider value={value}>
      {children}
    </ThemeContext.Provider>
  );
};

// 3. Создаём кастомный хук для удобного использования контекста
```

```
export const useTheme = () => {
  const context = useContext(ThemeContext);
  if (!context) {
    throw new Error('useTheme must be used within ThemeProvider');
  }
  return context;
};
```

Разбор кода:

Строка/Блок	Что делает
createContext()	Создаёт новый контекст (контейнер для данных)
ThemeProvider	Компонент-провайдер, который даёт значение контекста всем вложенным компонентам
useState('light')	Состояние темы хранится внутри провайдера
value = { theme, toggleTheme }	Формируем объект со всеми данными, которые будут доступны
ThemeContext.Provider	Оборачивает детей и даёт им доступ к value
useTheme	Кастомный хук - упрощает чтение контекста

Шаг 5. Используем контекст в компонентах

Файл src/components/ThemedButton.jsx:

```
import { useTheme } from '../contexts/ThemeContext';

const ThemedButton = () => {
  // Получаем тему из контекста напрямую! Без пропсов!
  const { theme } = useTheme();

  return (
    <button style={{
      backgroundColor: theme === 'dark' ? '#333' : '#eee',
      color: theme === 'dark' ? '#fff' : '#000',
      padding: '10px 20px',
      border: 'none',
      borderRadius: '5px',
      cursor: 'pointer'
    }}>
      Кнопка в {theme} теме
    </button>
  );
};

export default ThemedButton;
```

Файл src/components/Toolbar.jsx:

```
import ThemedButton from './ThemedButton';

const Toolbar = () => {
  // Обратите внимание: пропсов нет! Компонент просто рендерит дочерние компоненты
```

```
  return (
    <div style={{ padding: '20px' }}>
      <h3>Панель инструментов</h3>
      <ThemedButton />
    </div>
  );
};

export default Toolbar;
```

Файл src/App.jsx (новая версия - с Context):

```
import { useTheme } from './contexts/ThemeContext';
import Toolbar from './components/Toolbar';

function App() {
  const { theme, toggleTheme } = useTheme();

  return (
    <div>
      <h1>Решение: Context API</h1>
      <button onClick={toggleTheme}>
        Переключить тему (сейчас {theme})
      </button>
      <Toolbar />
    </div>
  );
}

export default App;
```

Файл src/main.jsx (оборачиваем приложение в провайдер):

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';
import { ThemeProvider } from './contexts/ThemeContext';

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <ThemeProvider>
      <App />
    </ThemeProvider>
  </React.StrictMode>
);
```

Шаг 6. Запуск и проверка

```
npm run dev
```

Что должно работать:

При нажатии на кнопку тема переключается между светлой и тёмной

Кнопка внутри Toolbar меняет цвет без получения темы через пропсы

4. Более сложный пример: корзина товаров

Теперь создадим второй контекст - для управления корзиной товаров. Это покажет, как несколько контекстов работают вместе.

Шаг 4.1. Создаём контекст корзины

Файл src/contexts/CartContext.jsx:

```
import { createContext, useState, useContext } from 'react';
```

```
const CartContext = createContext();

export const CartProvider = ({ children }) => {
  const [cartItems, setCartItems] = useState([]);

  // Добавление товара в корзину
  const addToCart = (product) => {
    setCartItems(prevItems => {
      const existingItem = prevItems.find(item => item.id === product.id);

      if (existingItem) {
        return prevItems.map(item =>
          item.id === product.id
            ? { ...item, quantity: item.quantity + 1 }
            : item
        );
      } else {
        return [...prevItems, { ...product, quantity: 1 }];
      }
    });
  };

  // Удаление товара из корзины
  const removeFromCart = (productId) => {
    setCartItems(prevItems => prevItems.filter(item => item.id !== productId));
  };

  // Изменение количества товара
```

```
  const updateQuantity = (productId, newQuantity) => {
    if (newQuantity <= 0) {
      removeFromCart(productId);
      return;
    }

    setCartItems(prevItems =>
      prevItems.map(item =>
        item.id === productId
          ? { ...item, quantity: newQuantity }
          : item
      )
    );
  };

  // Очистка корзины
  const clearCart = () => {
    setCartItems([]);
  };

  // Подсчёт общей стоимости
  const getTotalPrice = () => {
    return cartItems.reduce((total, item) => total + item.price * item.quantity,
0);
  };

  // Подсчёт общего количества товаров
  const getTotalItems = () => {
    return cartItems.reduce((total, item) => total + item.quantity, 0);
```

```
  };

  const value = {
    cartItems,
    addToCart,
    removeFromCart,
    updateQuantity,
    clearCart
    getTotalPrice,
    getTotalItems
  };

  return (
    <CartContext.Provider value={value}>
      {children}
    </CartContext.Provider>
  );
};

export const useCart = () => {
  const context = useContext(CartContext);
  if (!context) {
    throw new Error('useCart must be used within CartProvider');
  }
  return context;
};
```

Шаг 4.2. Создаём компоненты

Файл src/components/ProductCard.jsx:

```
import { useCart } from '../contexts/CartContext';

const ProductCard = ({ product }) => {
  const { addToCart } = useCart();

  return (
    <div style={{
      border: '1px solid #ccc',
      borderRadius: '8px',
      padding: '15px',
      margin: '10px',
      display: 'inline-block',
      width: '200px'
    }}>
      <h3>{product name}</h3>
      <p>Цена: {product price} ₽</p>
      <button onClick={() => addToCart(product)}>
        Добавить в корзину
      </button>
    </div>
  );
};

export default ProductCard;
```

Файл src/components/Cart.jsx:

```
import { useCart } from '../contexts/CartContext';

const Cart = () => {
```

```
const { cartItems, removeFromCart, updateQuantity, clearCart, getTotalPrice, getTotalItems } = useCart();

if (cartItems.length === 0) {
  return (
    <div style={{ padding: '20px', textAlign: 'center' }}>
      <h3>Корзина пуста</h3>
      <p>Добавьте товары</p>
    </div>
  );
}

return (
  <div style={{ border: '1px solid #ccc', borderRadius: '8px', padding: '15px', marginTop: '20px' }}>
    <h3>Корзина ({getTotalItems()} товаров)</h3>

    {cartItems.map(item => (
      <div key={item.id} style={{ display: 'flex', justifyContent: 'space-between', marginBottom: '10px' }}>
        <div>
          {item.name} - {item.price} ₽ × {item.quantity} = {item.price * item.quantity} ₽
        </div>
        <div>
          <button onClick={() => updateQuantity(item.id, item.quantity - 1)}>-</button>
          <span style={{ margin: '0 10px' }}>{item.quantity}</span>
```

```
                <button onClick={() => updateQuantity(item.id, item.quantity +
1)}>+</button>
                <button onClick={() => removeFromCart(item.id)}
style={{ marginLeft: '10px' }}>Удалить</button>
              </div>
            </div>
          ))}

          <div style={{ marginTop: '15px', textAlign: 'right' }}>
            <strong>Итого: {getTotalPrice()} ₽</strong>
            <button onClick={clearCart} style={{ marginLeft:
'10px' }}>Очистить</button>
          </div>
        </div>
      );
    };

    export default Cart;
```

Шаг 4.3. Обновляем App.jsx

Файл src/App.jsx:

```
import { useTheme } from './contexts/ThemeContext';
import { useCart } from './contexts/CartContext';
import Toolbar from './components/Toolbar';
import Cart from './components/Cart';
import ProductCard from './components/ProductCard';

// Пример данных товаров
const products = [
```

```
    { id: 1, name: 'Ноутбук', price: 50000 },
    { id: 2, name: 'Мышь', price: 1500 },
    { id: 3, name: 'Клавиатура', price: 4000 },
  ];

  function App() {
    const { theme, toggleTheme } = useTheme();
    const { getTotalItems } = useCart();

    return (
      <div style={{
        padding: '20px',
        backgroundColor: theme === 'dark' ? '#222' : '#fff',
        color: theme === 'dark' ? '#fff' : '#000',
        minHeight: '100vh'
      }}>
        <div style={{ display: 'flex', justifyContent: 'space-between', alignItems:
'center' }}>
          <h1>Интернет-магазин</h1>
          <div>
            <button onClick={toggleTheme}>
              {theme === 'light' ? '🌙 Тёмная тема' : '☀ Светлая тема'}
            </button>
            <span style={{ marginLeft: '15px' }}>🛒 Корзина: {getTotalItems()}
товаров</span>
          </div>
        </div>

        <Toolbar />
```

```
      <h2>Товары</h2>
      <div>
        {products.map(product => (
          <ProductCard key={product.id} product={product} />
        ))}
      </div>

      <Cart />
    </div>
  );
}

export default App;
```

Файл src/main.jsx (оба провайдера):

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';
import { ThemeProvider } from './contexts/ThemeContext';
import { CartProvider } from './contexts/CartContext';

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <ThemeProvider>
      <CartProvider>
        <App />
      </CartProvider>
    </ThemeProvider>
```

```
  </React.StrictMode>
);
```

5. Индивидуальные задания

5.1. Базовый уровень

Требования: Создать приложение с переключением темы (светлая/тёмная) и минимум 3 компонентами, использующими тему.

Вариант	Дополнительные элементы
1	Кнопка меняет цвет текста на странице
2	Карточки товаров меняют фон в зависимости от темы
3	В шапке отображается иконка солнца/луны
4	Форма ввода меняет цвета полей в зависимости от темы
5	Таблица данных меняет стили строк при смене темы
6	Список задач (todo list) с адаптивными цветами
7	Календарь с цветами под тему
8	Чат-компонент с адаптивной цветовой схемой
9	Галерея изображений с подписями, меняющими цвет

Вариант	Дополнительные элементы
10	Аккордеон (раскрывающиеся панели) с темой
11	Вкладки (tabs) с цветами темы
12	Модальное окно, меняющее цвета при смене темы
13	Панель настроек с переключателями
14	График (простой) с цветами под тему
15	Прогресс-бар, меняющий цвет в зависимости от темы

5.2. Средний уровень (15 вариантов)

Требования: Создать приложение с корзиной товаров и дополнительным функционалом. Использовать минимум два контекста.

Вариант	Дополнительный функционал
1	Добавить поле «промокод» с вычислением скидки
2	Добавить избранное (wishlist) в отдельном контексте
3	Добавить фильтрацию товаров по цене
4	Добавить сортировку товаров (по цене, по имени)

Вариант	Дополнительный функционал
5	Добавить постраничный вывод товаров (пагинацию)
6	Добавить отзывы к товарам в отдельном контексте
7	Сохранять корзину в localStorage
8	Добавить кнопку «Купить в один клик»
9	Добавить сравнение товаров (до 3 товаров)
10	Добавить поиск товаров по названию
11	Добавить уведомления (toast) при добавлении в корзину
12	Добавить гистограмму цен товаров в корзине
13	Добавить возможность задавать адрес доставки
14	Добавить калькулятор доставки
15	Добавить историю заказов в отдельном контексте

5.3. Продвинутый уровень

Требования: Создать сложное приложение с несколькими взаимосвязанными контекстами.

Вариант	Тип приложения	Контексты
1	Планировщик задач	задачи, фильтры, пользователь
2	Менеджер расходов	транзакции, категории, бюджет
3	Приложение для заметок	заметки, теги, поиск
4	Трекер привычек	привычки, статистика, настройки
5	Голосовалка/опросы	вопросы, ответы, результаты
6	Бронирование столиков	рестораны, время, корзина
7	Плейлист музыки	треки, очередь, плеер
8	Игра «Угадай число»	игра, статистика, рекорды
9	Приложение-таймер	таймеры, история, настройки

Вариант	Тип приложения	Контексты
10	Канбан-доска	колонки, задачи, перемещение
11	Конвертер валют	курсы, избранные валюты, история
12	Погодное приложение	города, текущая погода, прогноз
13	Редактор Markdown	документ, тема, предпросмотр
14	Чат-приложение	сообщения, пользователи, комната
15	Календарь событий	события, напоминания, фильтры

6. Контрольные вопросы

1. Что такое состояние (state) в React и зачем оно нужно?
2. В чём разница между локальным и глобальным состоянием?
3. Что такое «проброс пропсов» (prop drilling) и в чём его проблема?
4. Для чего нужен Context API?
5. Какие компоненты и хуки используются при работе с Context API?
6. Как создать провайдер (Provider) и передать в него значение?
7. Как получить значение из контекста в любом компоненте?

8. Зачем нужен кастомный хук для контекста (например, useTheme)?
9. Как создать несколько независимых контекстов в одном приложении?
10. Как объединить несколько провайдеров в одном приложении?
11. Как оптимизировать контекст, чтобы избежать лишних перерисовок?
12. В чём разница между Context API и Redux? Когда что лучше использовать?
13. Можно ли использовать Context API без провайдера?
14. Что произойдёт, если использовать useContext вне провайдера?
15. Как передать функцию через контекст для изменения состояния?

7. Требования к отчёту

Титульный лист (название работы, ФИО, группа, вариант)

Цель работы

Теоретическое обоснование (своими словами)

Выполнение практического примера (код, скриншоты)

Выполнение индивидуального задания (код, скриншоты)

Ответы на контрольные вопросы (не менее 5 по своему уровню)

Вывод

8. Критерии оценивания

Критерий	Базовый (max 35)	Средний (max 50)	Продвинутый (max 60)
Понимание useState	10	5	5

Критерий	Базовый (max 35)	Средний (max 50)	Продвинутый (max 60)
Понимание проблемы проброса пропсов	10	5	5
Создание контекста и провайдера	10	10	10
Использование хука useContext	5	5	5
Создание кастомного хука	—	5	5
Несколько независимых контекстов	—	5	5
Сложная бизнес-логика	—	10	10
Дополнительные функции (фильтрация, поиск)	—	5	10
Качество кода	—	—	5
Итого	35	50	60

Шкала перевода баллов в оценку:

Оценка	Базовый (35)	Средний (50)	Продвинутый (60)
5 (отлично)	32-35	46-50	55-60
4 (хорошо)	28-31	40-45	48-54
3 (удовлетворительно)	22-27	32-39	38-47
2 (неудовлетворительно)	0-21	0-31	0-37

Лабораторная работа №17

Управление состоянием с Redux Toolkit (RTK)

1. Цель работы

Освоить управление глобальным состоянием в React-приложениях с использованием Redux Toolkit. Научиться устанавливать и настраивать Redux Toolkit, создавать слайсы (slices), настраивать store, использовать хуки useSelector и useDispatch, а также работать с асинхронными действиями через createAsyncThunk.

2. Теоретическое обоснование

2.1. Что такое Redux и зачем он нужен?

Redux - это библиотека для управления глобальным состоянием приложения. Если Context API подходит для небольших приложений (тема, язык, авторизация), то Redux создан для сложных приложений с большим количеством состояния и частыми изменениями.

Когда Context API становится недостаточно:

Проблема	Описание
Избыточные перерисовки	При изменении любого поля в контексте перерисовываются ВСЕ потребители
Сложность отладки	Непонятно, кто и когда изменил данные
Нет инструментов разработчика	Невозможно «отмотать» состояние назад
Сложная логика обновления	Бизнес-логика изменения данных становится запутанной

2.2. Три принципа Redux

Принцип	Что означает
Единый источник истины	Состояние всего приложения хранится в одном объекте-дереве
Состояние только для чтения	Нельзя изменить состояние напрямую - только через действия
Чистые функции-редюсеры	Как состояние меняется в ответ на действие - определяют чистые функции

2.3. Основные понятия Redux

Понятие	Что это	Аналогия
Store (хранилище)	Единый объект, содержащий всё состояние	Склад с товарами
Action (действие)	Объект, описывающий, что произошло	Заказ на складе
Reducer (редюсер)	Чистая функция, которая решает, как изменить состояние	Кладовщик, выполняющий заказ
Dispatch (отправка)	Функция для отправки действий в store	Подача заказа кладовщику
Selector (селектор)	Функция для извлечения данных из store	Форма заказа на получение товара

2.4. Почему Redux Toolkit, а не классический Redux?

Классический Redux (много кода):

```
// actions.js
const INCREMENT = 'INCREMENT';
const increment = () => ({ type: INCREMENT });

// reducer.js
const initialState = { count: 0 };
```

```
function counterReducer(state = initialState, action) {
  switch (action type) {
    case INCREMENT:
      return { count: state count + 1 };
    default:
      return state;
  }
}

// store.js
import { createStore } from 'redux';
const store = createStore(counterReducer);
```

Redux Toolkit (меньше кода):

```
import { createSlice, configureStore } from '@reduxjs/toolkit';

const counterSlice = createSlice({
  name: 'counter',
  initialState: { count: 0 },
  reducers: {
    increment: (state) => { state count += 1; }
  }
});

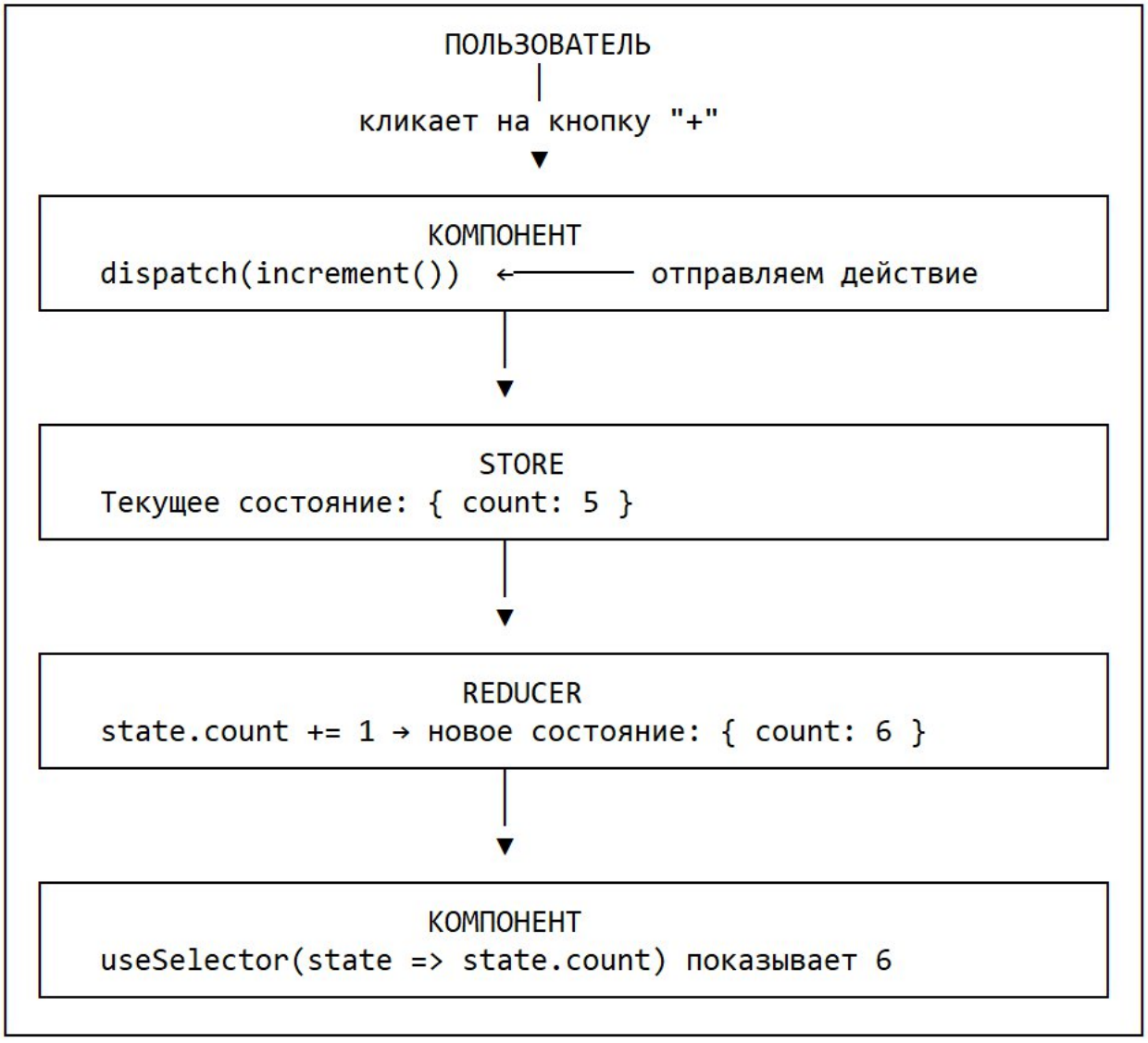
const store = configureStore({ reducer: counterSlice reducer });
```

Что даёт Redux Toolkit:

1. Гораздо меньше шаблонного кода

2. Встроенная поддержка асинхронности (createAsyncThunk)
3. Immer позволяет писать мутирующий код (безопасно)
4. Автоматическое подключение Redux DevTools

2.5. Схема работы Redux



3. Практический пример (пошагово, для новичков)

Что мы будем делать: Создадим корзину интернет-магазина с использованием Redux Toolkit. В приложении будет список товаров, возможность добавлять/удалять товары, менять количество и видеть общую стоимость.

Шаг 1. Создание проекта и установка зависимостей

```
# Создаём проект
```

```
npm create vite@latest rtk-cart -- --template react
cd rtk-cart

# Устанавливаем зависимости
npm install

# Устанавливаем Redux Toolkit и React-Redux
npm install @reduxjs/toolkit react-redux

# Запускаем проект
npm run dev
```

Шаг 2. Создаём структуру папок

```
src/
├── store/
│   ├── store.js          # Настройка глобального хранилища
│   └── cartSlice.js      # Слайс для корзины
├── components/
│   ├── ProductList.jsx   # Список товаров
│   ├── Cart.jsx          # Корзина
│   └── CartItem.jsx      # Один товар в корзине
├── App.jsx
└── main.jsx
```

Шаг 3. Создаём слайс для корзины (cartSlice.js)

Файл src/store/cartSlice.js:

```
import { createSlice } from '@reduxjs/toolkit';

// 1. Начальное состояние корзины
```

```
const initialState = {
  items: [],        // Массив товаров в корзине
  totalQuantity: 0, // Общее количество товаров
  totalPrice: 0,    // Общая стоимость
};

// 2. Создаём слайс
const cartSlice = createSlice({
  name: 'cart',
  initialState,
  reducers: {
    // Добавление товара в корзину
    addItem: (state, action) => {
      const newItem = action payload;
      const existingItem = state items find(item => item id === newItem id);

      if (existingItem) {
        // Если товар уже есть - увеличиваем количество
        existingItem quantity += newItem quantity;
        existingItem totalPrice += newItem price * newItem quantity;
      } else {
        // Если товара нет - добавляем новый
        state items push({
          id: newItem id,
          name: newItem name,
          price: newItem price,
          quantity: newItem quantity,
          totalPrice: newItem price * newItem quantity
        });
```

```
    }

    // Обновляем общие показатели
    state.totalQuantity += newItem.quantity;
    state.totalPrice += newItem.price * newItem.quantity;
  },

  // Удаление товара из корзины
  removeItem: (state, action) => {
    const id = action.payload;
    const existingItem = state.items.find(item => item.id === id);

    if (existingItem) {
      state.totalQuantity -= existingItem.quantity;
      state.totalPrice -= existingItem.totalPrice;
      state.items = state.items.filter(item => item.id !== id);
    }
  },

  // Увеличение количества товара
  increaseQuantity: (state, action) => {
    const id = action.payload;
    const existingItem = state.items.find(item => item.id === id);

    if (existingItem) {
      existingItem.quantity++;
      existingItem.totalPrice += existingItem.price;
      state.totalQuantity++;
      state.totalPrice += existingItem.price;
```

```
    }
  },

  // Уменьшение количества товара
  decreaseQuantity: (state, action) => {
    const id = action.payload;
    const existingItem = state.items.find(item => item.id === id);

    if (existingItem && existingItem.quantity > 1) {
      existingItem.quantity--;
      existingItem.totalPrice -= existingItem.price;
      state.totalQuantity--;
      state.totalPrice -= existingItem.price;
    } else if (existingItem && existingItem.quantity === 1) {
      // Если количество становится 0 - удаляем товар
      state.totalQuantity--;
      state.totalPrice -= existingItem.price;
      state.items = state.items.filter(item => item.id !== id);
    }
  },

  // Очистка корзины
  clearCart: (state) => {
    state.items = [];
    state.totalQuantity = 0;
    state.totalPrice = 0;
  },
},
});
```

```
// 3. Экспортируем действия (actions)
export const { addItem, removeItem, increaseQuantity, decreaseQuantity,
clearCart } = cartSlice.actions;

// 4. Экспортируем редюсер
export default cartSlice.reducer;
```

Шаг 4. Настройка store (store.js)

Файл src/store/store.js:

```
import { configureStore } from '@reduxjs/toolkit';
import cartReducer from './cartSlice';

export const store = configureStore({
  reducer: {
    cart: cartReducer, // ключ 'cart' будет в глобальном состоянии
  },
});
```

Шаг 5. Подключение Redux к React (main.jsx)

Файл src/main.jsx:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import { Provider } from 'react-redux';
import { store } from './store/store';
import App from './App';

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
```

```
    <Provider store={store}>
      <App />
    </Provider>
  </React.StrictMode>
);
```

Шаг 6. Создаём компонент списка товаров (ProductList.jsx)

Файл src/components/ProductList.jsx:

```
import { useDispatch } from 'react-redux';
import { addItem } from '../store/cartSlice';

// Данные товаров (обычно приходят с сервера)
const products = [
  { id: 1, name: 'Ноутбук', price: 50000 },
  { id: 2, name: 'Мышь', price: 1500 },
  { id: 3, name: 'Клавиатура', price: 4000 },
  { id: 4, name: 'Монитор', price: 25000 },
  { id: 5, name: 'Наушники', price: 3000 },
];

const ProductList = () => {
  const dispatch = useDispatch();

  const handleAddToCart = (product) => {
    dispatch(addItem({
      id: product id,
      name: product name,
      price: product price,
      quantity: 1
```

```
  }));
 };

 return (
  <div style={{ marginBottom: '30px' }}>
   <h2>Товары</h2>
   <div style={{ display: 'flex', flexWrap: 'wrap', gap: '15px' }}>
    {products.map(product => (
     <div key={product.id} style={{
      border: '1px solid #ccc',
      borderRadius: '8px',
      padding: '15px',
      width: '200px',
      textAlign: 'center'
     }}>
      <h3>{product.name}</h3>
      <p>{product.price} ₽</p>
      <button onClick={() => handleAddToCart(product)}>
       Добавить в корзину
      </button>
     </div>
    ))}
   </div>
  </div>
 );
};

export default ProductList;
```

Шаг 7. Создаём компонент корзины (Cart.jsx)

Файл src/components/Cart.jsx:

```
import { useSelector, useDispatch } from 'react-redux';
import { increaseQuantity, decreaseQuantity, removeItem, clearCart } from '../store/cartSlice';

const Cart = () => {
  const dispatch = useDispatch();
  // Получаем данные из store
  const { items, totalQuantity, totalPrice } = useSelector((state) => state.cart);

  if (items.length === 0) {
    return (
      <div style={{ border: '1px solid #ccc', borderRadius: '8px', padding: '20px', textAlign: 'center' }}>
        <h2>Корзина пуста</h2>
        <p>Добавьте товары, чтобы они появились здесь</p>
      </div>
    );
  }

  return (
    <div style={{ border: '1px solid #ccc', borderRadius: '8px', padding: '20px' }}>
      <h2>Корзина ({totalQuantity} товаров)</h2>

      {items.map(item => (
        <div key={item.id} style={{
          display: 'flex',
```

```
            justifyContent: 'space-between',
            alignItems: 'center',
            borderBottom: '1px solid #eee',
            padding: '10px 0'
          }}>
            <div style={{ flex: 2 }}>
              <strong>{item name}</strong>
              <div>{item price} ₽ × {item quantity} = {item totalPrice} ₽</div>
            </div>
            <div>
              <button onClick={() => dispatch(decreaseQuantity(item.id))} style={{ margin: '0 5px' }}>-</button>
              <span>{item quantity}</span>
              <button onClick={() => dispatch(increaseQuantity(item.id))} style={{ margin: '0 5px' }}>+</button>
              <button onClick={() => dispatch(removeItem(item.id))} style={{ marginLeft: '10px', background: '#ff4444' }}>
                Удалить
              </button>
            </div>
          </div>
        ))}

        <div style={{ marginTop: '20px', textAlign: 'right' }}>
          <strong>Итого: {totalPrice} ₽</strong>
          <button onClick={() => dispatch(clearCart())} style={{ marginLeft: '10px', background: '#ff4444' }}>
            Очистить корзину
          </button>
```

```
      </div>
    </div>
  );
};

export default Cart;
```

Шаг 8. Собираем всё в App.jsx

Файл src/App.jsx:

```
import ProductList from './components/ProductList';
import Cart from './components/Cart';

function App() {
  return (
    <div style={{ maxWidth: '800px', margin: '0 auto', padding: '20px' }}>
      <h1>Интернет-магазин</h1>
      <ProductList />
      <Cart />
    </div>
  );
}

export default App;
```

Шаг 9. Запуск и проверка

```
npm run dev
```

Что должно работать:

1. Товары отображаются в списке
2. При нажатии «Добавить в корзину» товар появляется в корзине

3. Можно увеличивать и уменьшать количество
4. Можно удалять отдельные товары
5. Можно очистить всю корзину
6. Общая стоимость и количество пересчитываются автоматически

4. Асинхронные действия с createAsyncThunk

В реальных приложениях товары обычно загружаются с сервера. Рассмотрим, как добавить асинхронную загрузку данных.

Шаг 4.1. Создаём слайс для товаров с асинхронной загрузкой

Файл src/store/productsSlice.js:

```
import { createSlice, createAsyncThunk } from '@reduxjs/toolkit';
import axios from 'axios';

// 1. Создаём асинхронное действие для загрузки товаров
export const fetchProducts = createAsyncThunk(
  'products/fetchProducts',
  async (_, thunkAPI) => {
    try {
      // Имитация запроса к API (можно заменить на реальный)
      const response = await new Promise((resolve) => {
        setTimeout(() => {
          resolve({
            data: [
              { id: 1, name: 'Ноутбук', price: 50000 },
              { id: 2, name: 'Мышь', price: 1500 },
              { id: 3, name: 'Клавиатура', price: 4000 },
              { id: 4, name: 'Монитор', price: 25000 },
            ]
```

```
        });
      }, 1000);
    });
    return response.data;
  } catch (error) {
    return thunkAPI.rejectWithValue(error.message);
  }
}
);

// 2. Начальное состояние
const initialState = {
  items: [],
  status: 'idle',   // 'idle' | 'loading' | 'succeeded' | 'failed'
  error: null
};

// 3. Создаём слайс
const productsSlice = createSlice({
  name: 'products',
  initialState,
  reducers: {},
  // 4. Обрабатываем асинхронные действия
  extraReducers: (builder) => {
    builder
      .addCase(fetchProducts.pending, (state) => {
        state.status = 'loading';
        state.error = null;
      })
```

```
      .addCase(fetchProducts.fulfilled, (state, action) => {
        state.status = 'succeeded';
        state.items = action.payload;
      })
      .addCase(fetchProducts.rejected, (state, action) => {
        state.status = 'failed';
        state.error = action.payload;
      });
  }
});

export default productsSlice.reducer;
```

Шаг 4.2. Подключаем новый слайс в store

Файл src/store/store.js (обновлённый):

```
import { configureStore } from '@reduxjs/toolkit';
import cartReducer from './cartSlice';
import productsReducer from './productsSlice';

export const store = configureStore({
  reducer: {
    cart: cartReducer,
    products: productsReducer,
  },
});
```

Шаг 4.3. Создаём компонент с асинхронной загрузкой

Файл src/components/ProductListAsync.jsx:

```
import { useEffect } from 'react';
```

```
import { useDispatch, useSelector } from 'react-redux';
import { fetchProducts } from '../store/productsSlice';
import { addItem } from '../store/cartSlice';

const ProductListAsync = () => {
  const dispatch = useDispatch();
  const { items: products, status, error } = useSelector((state) =>
state products);

  // Загружаем товары при монтировании компонента
  useEffect(() => {
    if (status === 'idle') {
      dispatch(fetchProducts());
    }
  }, [status, dispatch]);

  if (status === 'loading') {
    return <div>Загрузка товаров...</div>;
  }

  if (status === 'failed') {
    return <div>Ошибка: {error}</div>;
  }

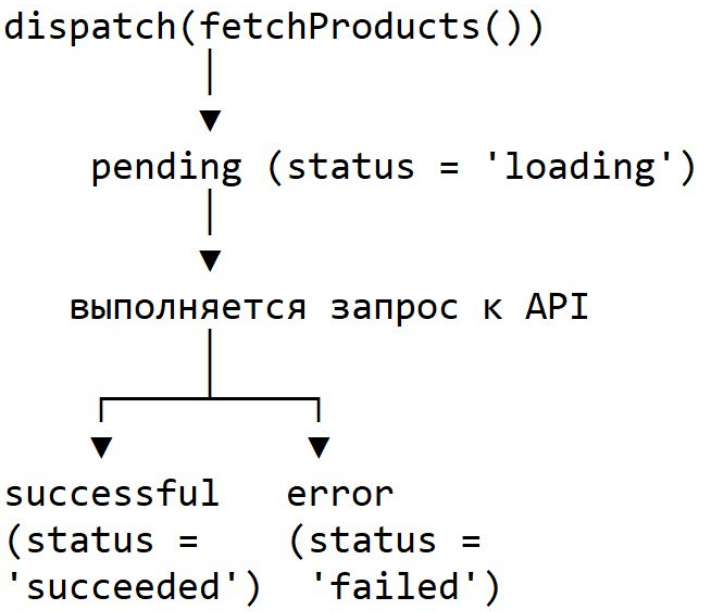
  const handleAddToCart = (product) => {
    dispatch(addItem({
      id: product id,
      name: product name,
      price: product price,
```

```
    quantity: 1
  }));
};

return (
  <div style={{ marginBottom: '30px' }}>
    <h2>Товары</h2>
    <div style={{ display: 'flex', flexWrap: 'wrap', gap: '15px' }}>
      {products.map(product => (
        <div key={product.id} style={{
          border: '1px solid #ccc',
          borderRadius: '8px',
          padding: '15px',
          width: '200px',
          textAlign: 'center'
        }}>
          <h3>{product.name}</h3>
          <p>{product.price} ₽</p>
          <button onClick={() => handleAddToCart(product)}>
            Добавить в корзину
          </button>
        </div>
      ))}
    </div>
  </div>
);
};

export default ProductListAsync;
```

Жизненный цикл асинхронного действия:



5. Индивидуальные задания

5.1. Базовый уровень

Требования: Создать корзину покупок с использованием Redux Toolkit. Реализовать добавление, удаление, изменение количества товаров.

Вариант	Тип товаров	Дополнительное поле
1	Электроника	вес товара
2	Одежда	размер
3	Книги	автор
4	Продукты	срок годности
5	Спорттовары	бренд

Вариант	Тип товаров	Дополнительное поле
6	Косметика	объём
7	Игрушки	возрастное ограничение
8	Мебель	материал
9	Автозапчасти	совместимость
10	Лекарства	дозировка
11	Стройматериалы	количество в упаковке
12	Подарки	упаковка
13	Цветы	сорт
14	Бытовая химия	тип упаковки
15	Зоотовары	тип животного

5.2. Средний уровень (15 вариантов)

Требования: Добавить асинхронную загрузку товаров с сервера (createAsyncThunk), индикатор загрузки и обработку ошибок.

Вариант	Дополнительный функционал
1	Добавить поиск по названию товара
2	Добавить фильтрацию по цене
3	Добавить сортировку по цене
4	Добавить пагинацию (постраничный вывод)
5	Добавить категории товаров
6	Добавить избранное (wishlist)
7	Сохранять корзину в localStorage
8	Добавить промокод со скидкой
9	Добавить историю заказов
10	Добавить отзывы к товарам
11	Добавить рейтинг товаров
12	Добавить кнопку «Купить в один клик»
13	Добавить расчёт доставки
14	Добавить валюту (рубли/доллары)
15	Добавить сравнение товаров

5.3. Продвинутый уровень (15 вариантов)

Требования: Реализовать полноценное приложение с несколькими слайсами, асинхронными запросами и сложной бизнес-логикой.

Вариант	Тип приложения	Слайсы
1	Доска задач (Trello)	задачи, колонки, пользователь
2	Планировщик задач	задачи, категории, фильтры
3	Менеджер расходов	транзакции, категории, бюджет
4	Приложение для заметок	заметки, теги, поиск
5	Трекер привычек	привычки, статистика, настройки
6	Голосовалка/опросы	вопросы, ответы, результаты
7	Бронирование столиков	рестораны, время, брони
8	Плейлист музыки	треки, очередь, плеер
9	Игра «Угадай число»	игра, статистика, рекорды

Вариант	Тип приложения	Слайсы
10	Приложение-таймер	таймеры, история, настройки
11	Канбан-доска	колонки, задачи, перемещение
12	Конвертер валют	курсы, избранное, история
13	Погодное приложение	города, погода, прогноз
14	Чат-приложение	сообщения, пользователи, комната
15	Календарь событий	события, напоминания, фильтры

6. Контрольные вопросы

1. Что такое Redux и какие проблемы он решает?
2. Какие три принципа лежат в основе Redux?
3. Что такое store в Redux?
4. Что такое action и как он выглядит?
5. Что такое reducer и каковы его свойства?
6. Чем отличается классический Redux от Redux Toolkit?
7. Что такое createSlice и для чего он нужен?

8. Как настроить store с помощью configureStore?
9. Для чего нужен хук useSelector?
10. Для чего нужен хук useDispatch?
11. Что такое createAsyncThunk и для чего он используется?
12. Какие три состояния бывают у асинхронного действия?
13. Как добавить новый слайс в существующий store?

14. В чём разница между useSelector и store.getState()?
15. Как оптимизировать useSelector для предотвращения лишних перерисовок?
16. Что такое extraReducers и когда он используется?
17. Как обрабатывать ошибки в createAsyncThunk?
18. Как организовать несколько слайсов, взаимодействующих друг с другом?

7. Требования к отчёту

Титульный лист (название работы, ФИО, группа, вариант)

Цель работы

Теоретическое обоснование (своими словами)

Выполнение практического примера (код, скриншоты)

Выполнение индивидуального задания (код, скриншоты)

Ответы на контрольные вопросы (не менее 5 по своему уровню)

Вывод

8. Критерии оценивания

Критерий	Базовый (max 40)	Средний (max 55)	Продвинутый (max 70)
Установка и настройка RTK	5	5	5
Создание слайса (корзина)	15	10	10
Настройка store	5	5	5
Использование useSelector	5	5	5
Использование useDispatch	5	5	5
createAsyncThunk	—	15	10
Несколько слайсов	—	10	10
Сложная бизнес-логика	—	—	10
Дополнительные функции	5	—	5
Качество кода	—	—	5
Итого	40	55	70

Шкала перевода баллов в оценку:

Оценка	Базовый (40)	Средний (55)	Продвинутый (70)
5 (отлично)	36-40	50-55	63-70
4 (хорошо)	32-35	44-49	56-62
3 (удовлетворительно)	26-31	38-43	49-55
2 (неудовлетворительно)	0-25	0-37	0-48

Лабораторная работа №18

Работа с API с использованием React Query (TanStack Query)

1. Цель работы

Научиться работать с серверными данными в React-приложениях с использованием библиотеки React Query (TanStack Query). Освоить кэширование данных, автоматическую синхронизацию с бэкендом, использование хуков useQuery для получения данных и useMutation для их изменения, а также механизм инвалидации кэша.

2. Теоретическое обоснование

2.1. Какие проблемы решает React Query?

При работе с API в React без специальных инструментов разработчики сталкиваются с рядом проблем:

Проблема	Описание	Пример
Много повторяющегося кода	Для каждого запроса нужно писать useState для данных, загрузки и ошибки	10 запросов = 30 useState
Нет кэширования	При перемонтировании компонента данные загружаются заново	Пользователь перешёл на страницу и обратно - повторный запрос
Гонка запросов	Быстрое переключение страниц может вызвать старые ответы, перезаписывающие новые	Пользователь кликнул страницу 5, затем 3 - сначала загрузится страница 3
Нет повторных попыток	При ошибке соединения данные никогда не загрузятся	Упал Wi-Fi на секунду - всё сломалось
Нет синхронизации	При изменении данных в одном месте, другое не обновляется	Добавили пост, список не обновился

2.2. Что делает React Query (TanStack Query)?

React Query - это библиотека для управления серверным состоянием. Она берёт на себя:

1. Кэширование данных
2. Фоновую синхронизацию
3. Повторные попытки при ошибках
4. Дедупликацию запросов (один запрос на несколько компонентов)
5. Инвалидацию и обновление кэша

2.3. Сравнение: обычный fetch vs React Query

Обычный fetch (без библиотек):

```
function NewsList() {
  const [news, setNews] = useState([]);
  const [isLoading, setIsLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    fetch('/api/news/')
      .then(res => res.json())
      .then(data => {
        setNews(data);
        setIsLoading(false);
      })
      .catch(err => {
        setError(err);
        setIsLoading(false);
      });
  }, []);

  if (isLoading) return <div>Загрузка...</div>;
  if (error) return <div>Ошибка: {error.message}</div>;
```

```
  return <div>{news.map(...)}</div>;
}
```

С React Query:

```
import { useQuery } from '@tanstack/react-query';

function NewsList() {
  const { data, isLoading, error } = useQuery({
    queryKey: ['news'],
    queryFn: () => fetch('/api/news/').then(res => res.json())
  });

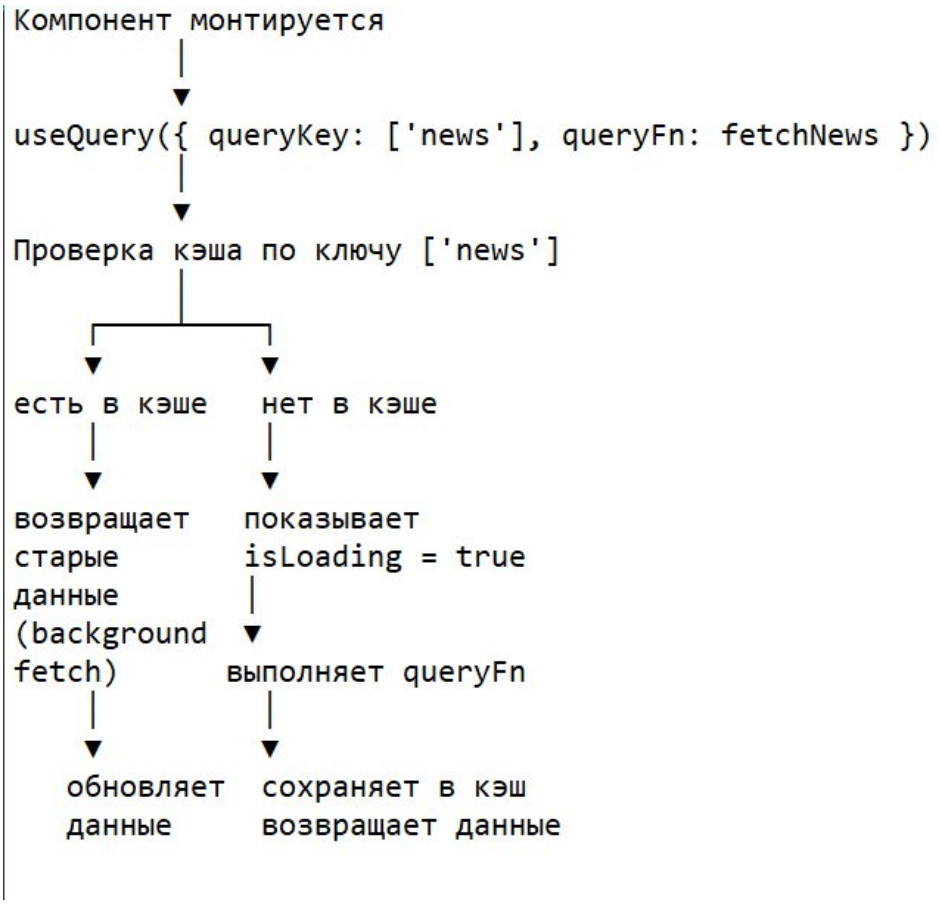
  if (isLoading) return <div>Загрузка...</div>;
  if (error) return <div>Ошибка: {error.message}</div>;
  return <div>{data.map(...)}</div>;
}
```

2.4. Основные понятия React Query

Понятие	Что это	Аналогия
QueryClient	Менеджер кэша, хранит все данные	База данных на клиенте
QueryClientProvider	Провайдер, дающий доступ к клиенту	Розетка, к которой подключаются компоненты

Понятие	Что это	Аналогия
useQuery	Хук для получения данных (GET)	Заказ книги в библиотеке
useMutation	Хук для изменения данных (POST, PUT, DELETE)	Сдача книги в библиотеку
queryKey	Уникальный ключ для данных	Штрих-код книги
queryFn	Функция, которая делает запрос	Библиотекарь, который ищет книгу
staleTime	Время, в течение которого данные считаются свежими	Срок годности продукта
invalidateQueries	Инвалидация (сброс) кэша	Уведомление, что книга устарела

2.5. Жизненный цикл запроса в React Query



3. Практический пример

Что мы будем делать: Создадим приложение для управления новостями, которое взаимодействует с Django API. Реализуем получение списка новостей, добавление новой новости и удаление.

Шаг 1. Создание React-проекта и установка зависимостей

```
# Создаём проект
npm create vite@latest react-query-demo -- --template react
cd react-query-demo

# Устанавливаем зависимости
npm install

# Устанавливаем React Query
npm install @tanstack/react-query
```

```
# Устанавливаем axios для удобной работы с API (опционально)
npm install axios

# Запускаем проект
npm run dev
```

Шаг 2. Настройка QueryClient (главный файл main.jsx)

Файл src/main.jsx:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import { QueryClient, QueryClientProvider } from '@tanstack/react-query';
import App from './App';

// 1. Создаём экземпляр QueryClient с настройками по умолчанию
const queryClient = new QueryClient({
  defaultOptions: {
    queries: {
      staleTime: 1000 * 60 * 5,   // 5 минут данные считаются свежими
      gcTime: 1000 * 60 * 10,     // 10 минут данные хранятся в кэше
      retry: 3,                   // при ошибке повторяем 3 раза
      retryDelay: 1000,           // задержка между повторами 1 секунда
      refetchOnWindowFocus: false,  // не обновлять при фокусе окна
    },
  },
});

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    {/* 2. Оборачиваем приложение в провайдер */}
    <QueryClientProvider client={queryClient}>
```

```
    <App />
  </QueryClientProvider>
</React.StrictMode>
);
```

Шаг 3. Создаём API-функции для работы с бэкендом

Файл src/api/newsApi.js:

```
import axios from 'axios';

// Базовый URL вашего Django API
const API_URL = 'http://localhost:8000/api';

// Получение списка новостей
export const fetchNews = async () => {
  const response = await axios.get(`${API_URL}/news/`);
  return response.data;
};

// Получение одной новости по ID
export const fetchNewsById = async (id) => {
  const response = await axios.get(`${API_URL}/news/${id}/`);
  return response.data;
};

// Добавление новой новости
export const addNews = async (newsData) => {
  const response = await axios.post(`${API_URL}/news/`, newsData);
  return response.data;
};
```

```
// Удаление новости
export const deleteNews = async (id) => {
  await axios.delete(`${API_URL}/news/${id}/`);
  return id;
};

// Обновление новости
export const updateNews = async ({ id, ...data }) => {
  const response = await axios.patch(`${API_URL}/news/${id}/`, data);
  return response.data;
};
```

Шаг 4. Создаём компонент списка новостей (с useQuery)

Файл src/components/NewsList.jsx:

```
import { useQuery } from '@tanstack/react-query';
import { fetchNews } from '../api/newsApi';

const NewsList = () => {
  // useQuery: получение данных
  const { data: news, isLoading, error, refetch, isFetching } = useQuery({
    queryKey: ['news'],          // уникальный ключ для этих данных
    queryFn: fetchNews,          // функция-запрос
  });

  if (isLoading) {
    return <div>🌀 Загрузка новостей...</div>;
  }

  if (error) {
    return (
```

```
        <div>
          <p style={{ color: 'red' }}>✕ Ошибка: {error.message}</p>
          <button onClick={() => refetch()}>Повторить</button>
        </div>
      );
    }

    return (
      <div>
        <div style={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center' }}>
          <h2>Новости ({news?.length || 0})</h2>
          <button onClick={() => refetch()} disabled={isFetching}>
            {isFetching ? 'Обновление...' : 'Обновить'}
          </button>
        </div>

        {news?.map(item => (
          <div key={item.id} style={{
            border: '1px solid #ccc',
            borderRadius: '8px',
            padding: '15px',
            marginBottom: '10px'
          }}>
            <h3>{item.title}</h3>
            <p>{item.content.substring(0, 100)}...</p>
            <small>Категория: {item.category_name}</small>
            <small style={{ marginLeft: '10px' }}>👁 {item.views} просмотров</small>
```

```
        </div>
      ))}
    </div>
  );
};

export default NewsList;
```

Шаг 5. Создаём компонент добавления новости (с useMutation)

Файл src/components/AddNewsForm.jsx:

```
import { useState } from 'react';
import { useMutation, useQueryClient } from '@tanstack/react-query';
import { addNews } from '../api/newsApi';

const AddNewsForm = () => {
  const queryClient = useQueryClient();  // доступ к кэшу
  const [title, setTitle] = useState('');
  const [content, setContent] = useState('');

  // useMutation: изменение данных
  const mutation = useMutation({
    mutationFn: addNews,
    onSuccess: () => {
      // После успешного добавления очищаем форму
      setTitle('');
      setContent('');

      // Инвалидируем кэш новостей - данные устарели, нужно перезагрузить
      queryClient.invalidateQueries({ queryKey: ['news'] });
```

```
    alert('Новость успешно добавлена!');
  },
  onError: (error) => {
    alert('Ошибка: ' + error.message);
  },
});

const handleSubmit = (e) => {
  e.preventDefault();
  if (!title.trim() || !content.trim()) {
    alert('Заполните все поля');
    return;
  }

  mutation.mutate({ title, content });
};

return (
  <form onSubmit={handleSubmit} style={{
    border: '1px solid #ccc',
    borderRadius: '8px',
    padding: '20px',
    marginBottom: '30px'
  }}>
    <h3>Добавить новость</h3>

    <div style={{ marginBottom: '10px' }}>
      <input
```

```
    type="text"
    placeholder="Заголовок"
    value={title}
    onChange={(e) => setTitle(e target value)}
    disabled={mutation isPending}
    style={{ width: '100%', padding: '8px' }}
  />
</div>

<div style={{ marginBottom: '10px' }}>
  <textarea
    placeholder="Содержание"
    value={content}
    onChange={(e) => setContent(e target value)}
    disabled={mutation isPending}
    rows={4}
    style={{ width: '100%', padding: '8px' }}
  />
</div>

<button
  type="submit"
  disabled={mutation isPending}
  style={{
    padding: '8px 16px',
    backgroundColor: '#007bff',
    color: 'white',
    border: 'none',
    borderRadius: '4px',
```

```
            cursor: 'pointer'
          }}
        >
          {mutation.isPending ? 'Добавление...' : '+ Добавить новость'}
        </button>

        {mutation.isError && (
          <p style={{ color: 'red', marginTop: '10px' }}>✕ {mutation.error.message}</p>
        )}
      </form>
    );
  };

  export default AddNewsForm;
```

Шаг 6. Создаём компонент удаления новости

Файл src/components/DeleteNewsButton.jsx:

```
import { useMutation, useQueryClient } from '@tanstack/react-query';
import { deleteNews } from '../api/newsApi';

const DeleteNewsButton = ({ newsId, newsTitle }) => {
  const queryClient = useQueryClient();

  const mutation = useMutation({
    mutationFn: deleteNews,
    onSuccess: () => {
      // Инвалидируем кэш после удаления
      queryClient.invalidateQueries({ queryKey: ['news'] });
      alert(`Новость "${newsTitle}" удалена`);
```

```
  },
  onError: (error) => {
    alert('Ошибка удаления: ' + error.message);
  },
});

const handleDelete = () => {
  if (window.confirm(`Удалить новость "${newsTitle}"?`)) {
    mutation.mutate(newsId);
  }
};

return (
  <button
    onClick={handleDelete}
    disabled={mutation.isPending}
    style={{
      padding: '5px 10px',
      backgroundColor: '#dc3545',
      color: 'white',
      border: 'none',
      borderRadius: '4px',
      cursor: 'pointer'
    }}
  >
    {mutation.isPending ? 'Удаление...' : '🗑 Удалить'}
  </button>
);
};
```

```
export default DeleteNewsButton;
```

Шаг 7. Создаём детальную страницу новости (с параметром)

Файл src/components/NewsDetail.jsx:

```
import { useQuery } from '@tanstack/react-query';
import { useParams } from 'react-router-dom';
import { fetchNewsById } from '../api/newsApi';
import DeleteNewsButton from './DeleteNewsButton';

const NewsDetail = () => {
  const { id } = useParams();  // получаем ID из URL /news/5

  const { data: news, isLoading, error } = useQuery({
    queryKey: ['news', id],    // ключ включает ID новости
    queryFn: () => fetchNewsById(id),
  });

  if (isLoading) return <div>🔄 Загрузка новости...</div>;
  if (error) return <div>✕ Новость не найдена</div>;

  return (
    <div>
      <h1>{news.title}</h1>
      <p>{news.content}</p>
      <div>Категория: {news.category_name}</div>
      <div>Просмотров: {news.views}</div>
      <div>Дата: {new Date(news.created_at).toLocaleDateString()}</div>
      <hr />
      <DeleteNewsButton newsId={news.id} newsTitle={news.title} />
```

```
    </div>
  );
};

export default NewsDetail;
```

Шаг 8. Собираем всё в App.jsx

Файл src/App.jsx:

```
import { BrowserRouter, Routes, Route, Link } from 'react-router-dom';
import NewsList from './components/NewsList';
import AddNewsForm from './components/AddNewsForm';
import NewsDetail from './components/NewsDetail';

function App() {
  return (
    <BrowserRouter>
      <div style={{ maxWidth: '800px', margin: '0 auto', padding: '20px' }}>
        <nav style={{ display: 'flex', gap: '20px', marginBottom: '20px', borderBottom: '1px solid #ccc', paddingBottom: '10px' }}>
          <Link to="/">Главная</Link>
          <Link to="/add">Добавить новость</Link>
        </nav>

        <Routes>
          <Route path="/" element={
            <>
              <NewsList />
            </>
          } />
          <Route path="/add" element={<AddNewsForm />} />
```

```
          <Route path="/news/:id" element={<NewsDetail />} />
        </Routes>
      </div>
    </BrowserRouter>
  );
}

export default App;
```

Шаг 9. Настройка CORS на Django бэкенде (если нужно)

Если ваш Django API работает на другом порту (например, 8000), а React на 5173, нужно настроить CORS:

```
pip install django-cors-headers
```

```
python
# settings.py
INSTALLED_APPS = [
    'corsheaders',
    # ...
]

MIDDLEWARE = [
    'corsheaders.middleware.CorsMiddleware',
    # ...
]

CORS_ALLOWED_ORIGINS = [
    "http://localhost:5173",  # порт React-приложения
]
```

4. Продвинутые возможности React Query

4.1. Пагинация с keepPreviousData

```
import { useQuery } from '@tanstack/react-query';
import { useState } from 'react';

const fetchPaginatedNews = async ({ page }) => {
  const response = await fetch(`/api/news/?page=${page}`);
  return response.json();
};

function PaginatedNews() {
  const [page, setPage] = useState(1);

  const { data, isLoading, isFetching } = useQuery({
    queryKey: ['news', { page }],
    queryFn: () => fetchPaginatedNews({ page }),
    keepPreviousData: true,   // сохраняем предыдущие данные во время загрузки новых
  });

  return (
    <div>
      {data?.results.map(news => <div key={news.id}>{news.title}</div>)}

      <div>
        <button onClick={() => setPage(p => p - 1)} disabled={page === 1}>
          ← Назад
        </button>
        <span> Страница {page} </span>
        <button onClick={() => setPage(p => p + 1)} disabled={!data?.next}>
          Вперёд →
```

```
      </button>
      {isFetching && <span> (обновление...)</span>}
    </div>
  </div>
 );
}
```

4.2. Зависимые запросы

```
function CategoryWithNews({ categoryId }) {
 // Сначала загружаем категорию
 const { data: category } = useQuery({
  queryKey: ['category', categoryId],
  queryFn: () => fetchCategory(categoryId),
 });

 // Затем загружаем новости категории (ЗАВИСИТ от category)
 const { data: news } = useQuery({
  queryKey: ['news', { category: categoryId }],
  queryFn: () => fetchNewsByCategory(categoryId),
  enabled: !!category,  // запрос выполнится только когда есть category
 });

 if (!category) return <div>Загрузка категории...</div>;

 return (
  <div>
   <h1>{category name}</h1>
   {news?.map(item => <div key={item.id}>{item title}</div>)}
  </div>
 );
```

```
}
```

4.3. Предзагрузка данных (prefetch)

```
import { useQueryClient } from '@tanstack/react-query';

function NewsListWithPrefetch() {
  const queryClient = useQueryClient();

  // Предзагружаем следующую страницу при наведении на кнопку
  const prefetchNextPage = () => {
    queryClient.prefetchQuery({
      queryKey: ['news', { page: currentPage + 1 }],
      queryFn: () => fetchPaginatedNews({ page: currentPage + 1 }),
    });
  };

  return (
    <div>
      {/* ... список новостей ... */}
      <button onMouseEnter={prefetchNextPage}>
        Следующая страница
      </button>
    </div>
  );
}
```

5. Индивидуальные задания

5.1. Базовый уровень (15 вариантов)

Требования: Создать приложение с получением данных через useQuery. Реализовать отображение списка, индикатор загрузки и обработку ошибок.

Вариант	Тип данных	API эндпоинт (пример)
1	Список пользователей	/api/users/
2	Список товаров	/api/products/
3	Список заказов	/api/orders/
4	Список комментариев	/api/comments/
5	Список категорий	/api/categories/
6	Список фильмов	/api/movies/
7	Список книг	/api/books/
8	Список городов	/api/cities/
9	Список сотрудников	/api/employees/
10	Список курсов	/api/courses/
11	Список событий	/api/events/
12	Список рецептов	/api/recipes/
13	Список игр	/api/games/

Вариант	Тип данных	API эндпоинт (пример)
14	Список песен	/api/songs/
15	Список автомобилей	/api/cars/

5.2. Средний уровень (15 вариантов)

Требования: Добавить useMutation для создания/редактирования/удаления и инвалидацию кэша.

Вариант	Функционал
1	CRUD для пользователей (создание, чтение, обновление, удаление)
2	CRUD для товаров + фильтрация по категории
3	CRUD для заказов + изменение статуса
4	CRUD для комментариев + модерация
5	CRUD для категорий + вложенные категории
6	CRUD для фильмов + рейтинг
7	CRUD для книг + поиск по автору
8	CRUD для городов + сортировка по населению
9	CRUD для сотрудников + поиск по отделу

Вариант	Функционал
10	CRUD для курсов + регистрация на курс
11	CRUD для событий + бронирование билетов
12	CRUD для рецептов + поиск по ингредиентам
13	CRUD для игр + рейтинг и отзывы
14	CRUD для песен + плейлисты
15	CRUD для автомобилей + фильтрация по марке

5.3. Продвинутый уровень (15 вариантов)

Требования: Реализовать пагинацию или бесконечную прокрутку, предзагрузку данных, зависимые запросы.

Вариант	Тип приложения	Продвинутая функциональность
1	Блог	Бесконечная прокрутка постов + предзагрузка следующей страницы
2	Интернет-магазин	Пагинация товаров + фильтрация по цене
3	Админ-панель	Зависимые запросы (выбор страны → города → улицы)

Вариант	Тип приложения	Продвинутая функциональность
4	Стриминг-платформа	Предзагрузка информации о следующем фильме
5	Доска объявлений	Пагинация + поиск по названию
6	Погодное приложение	Зависимые запросы (выбор города → прогноз)
7	Расписание	Пагинация по дням недели
8	Каталог книг	Бесконечная прокрутка + фильтрация по жанру
9	Недвижимость	Пагинация + сортировка по цене
10	Фото-галерея	Бесконечная прокрутка изображений
11	Форум	Пагинация тем + предзагрузка ответов
12	Биржа фриланса	Пагинация проектов + фильтрация по бюджету

Вариант	Тип приложения	Продвинутая функциональность
13	Справочник лекарств	Поиск с debounce + предзагрузка деталей
14	Календарь	Зависимые запросы (выбор года → месяца → дня → события)
15	Канбан-доска	Зависимые запросы (проект → колонки → задачи)

6. Контрольные вопросы

1. Что такое React Query и какие проблемы он решает?
2. Как установить и настроить React Query в проекте?
3. Для чего нужен компонент QueryClientProvider?
4. Как получить данные с сервера с помощью useQuery?
5. Какие значения возвращает хук useQuery?
6. Что такое queryKey и для чего он нужен?
7. Как отобразить индикатор загрузки и ошибку?
8. Как отправить данные на сервер с помощью useMutation?
9. Для чего нужна инвалидация кэша (invalidateQueries)?
10. Как получить доступ к QueryClient в компоненте?
11. В чём разница между isLoading и isFetching?
12. Как настроить повторные попытки при ошибке?
13. Что такое staleTime и как его настроить?
14. Как реализовать пагинацию с keepPreviousData?
15. Как реализовать зависимые запросы с enabled?

16. Как предзагрузить данные с помощью prefetchQuery?
17. Что такое useInfiniteQuery и для чего он нужен?
18. Как обновить кэш после мутации без полной инвалидации?
19. Как использовать React Query с TypeScript?

7. Требования к отчёту

Титульный лист (название работы, ФИО, группа, вариант)

Цель работы

Теоретическое обоснование (своими словами)

Выполнение практического примера (код, скриншоты)

Выполнение индивидуального задания (код, скриншоты)

Ответы на контрольные вопросы (не менее 5 по своему уровню)

Вывод

8. Критерии оценивания

Критерий	Базовый (max 35)	Средний (max 50)	Продвинутый (max 65)
Установка и настройка React Query	5	5	5
Использование useQuery (GET)	15	10	10
Использование useMutation (POST/PUT/DELETE)	—	15	10

Критерий	Базовый (max 35)	Средний (max 50)	Продвинутый (max 65)
Инвалидация кэша	—	10	10
Обработка загрузки и ошибок	10	5	5
Пагинация / keepPreviousData	—	—	10
Зависимые запросы / prefetch	—	—	10
Качество кода	5	5	5
Итого	35	50	65

Шкала перевода баллов в оценку:

Оценка	Базовый (35)	Средний (50)	Продвинутый (65)
5 (отлично)	32-35	46-50	59-65
4 (хорошо)	28-31	40-45	51-58
3 (удовлетворительно)	22-27	32-39	42-50
2 (неудовлетворительно)	0-21	0-31	0-41

Лабораторная работа №19

Аутентификация и защита роутов

1. Цель работы

Освоить реализацию аутентификации в React-приложениях с использованием JWT-токенов. Научиться безопасно хранить токены, создавать защищённые маршруты (PrivateRoute), автоматически обновлять access-токен и обрабатывать ошибку 401 (Unauthorized).

2. Теоретическое обоснование

2.1. Что такое JWT-аутентификация?

JWT (JSON Web Token) - это стандарт для создания токенов доступа, которые содержат информацию о пользователе и подписаны сервером.

Структура JWT:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoxMjMsInVzZXJuYW1lIjoiaXZhbiIsImV4cCI6MTY4MDAwMDA
wMH0.sQnKp0J3UkIYJkZ0L9X8KjK5W0X4cY9Kz5X8KjK5W0
      ^header                    ^payload                          ^signature
```

Часть	Содержание	Пример
Header	Алгоритм шифрования, тип токена	{"alg": "HS256", "typ": "JWT"}
Payload	Данные пользователя (user_id, username, exp)	{"user_id": 123, "username": "ivan", "exp":

Часть	Содержание	Пример
		1680000000}
Signature	Подпись для проверки целостности	зашифрованная строка

2.2. Два типа токенов

Тип токена	**Срок жизни**	**Назначение**	**Где хранить**
Access Token	15-30 минут	Доступ к защищённым API	память (переменная)
Refresh Token	7-30 дней	Получение нового access-токена	HttpOnly cookie / localStorage

Почему access-токен живёт так мало?

Если токен украдут, злоумышленник сможет пользоваться им ограниченное время

Короткий срок жизни повышает безопасность

Зачем нужен refresh-токен?

Чтобы пользователь не вводил логин и пароль каждые 15 минут

При истечении access-токена клиент автоматически запрашивает новый через refresh-токен

2.3. Схема работы аутентификации

text

1. ВХОД

Пользователь → POST /api/login/ {username, password}

Сервер → {access_token, refresh_token}

Клиент → сохраняет токены

2. ЗАЩИЩЁННЫЙ ЗАПРОС

Клиент → GET /api/profile/ + Header: Authorization: Bearer <access_token>

Сервер → проверяет токен, возвращает данные

3. ИСТЕЧЕНИЕ ACCESS-ТОКЕНА

Клиент → GET /api/profile/ (access_token истёк)

Сервер → 401 Unauthorized

Клиент → POST /api/token/refresh/ + refresh_token

Сервер → {new_access_token}

Клиент → повторяет исходный запрос с новым токеном

4. ВЫХОД

Клиент → удаляет токены из хранилища

2.4. Где хранить токены?

Способ хранения	Плюсы	Минусы	Для какого токена
localStorage	Просто, живёт после перезагрузки	Уязвим для XSS-атак	access (не рекомендуется)
sessionStorage	Исчезает при закрытии вкладки	Та же XSS-	access (лучше)

Способ хранения	Плюсы	Минусы	Для какого токена
		уязвимость	
Память (React state)	Безопасен (XSS не достанет)	Теряется при обновлении страницы	access (лучший)
HttpOnly cookie	Защищён от XSS, автоматически отправляется	Сложнее с SPA, нужна настройка CORS	refresh (идеально)

В этой лабораторной работе для простоты будем использовать localStorage, но в реальных проектах access-токен рекомендуется хранить в памяти, а refresh-токен - в HttpOnly cookie.

2.5. Защита роутов (PrivateRoute)

PrivateRoute - это компонент-обёртка, которая проверяет, авторизован ли пользователь, и либо показывает запрашиваемую страницу, либо перенаправляет на страницу входа.

```
// Как это работает
<PrivateRoute>
  <Dashboard />  {/* Покажется только если есть токен */}
</PrivateRoute>

// Если токена нет → перенаправление на /login
```

3. Практический пример

Что мы будем делать: Создадим приложение с формой входа, защищённым профилем пользователя и страницей "О сайте" (публичная). Реализуем хранение токена, автоматическое обновление access-токена и обработку 401 ошибки.

Шаг 1. Создание проекта и установка зависимостей

```
# Создаём проект
npm create vite@latest auth-demo -- --template react
cd auth-demo

# Устанавливаем зависимости
npm install
npm install react-router-dom
npm install axios

# Запускаем проект
npm run dev
```

Шаг 2. Создаём API-функции для работы с аутентификацией

Файл src/api/authApi.js:

```
import axios from 'axios';

const API_URL = 'http://localhost:8000/api';

// Создаём экземпляр axios с настройками по умолчанию
const api = axios.create({
  baseURL: API_URL,
```

```
  headers: {
    'Content-Type': 'application/json',
  },
});

// Вход в систему
export const login = async (username, password) => {
  const response = await api.post('/token/', { username, password });
  return response.data;  // { access, refresh }
};

// Обновление access-токена
export const refreshToken = async (refresh) => {
  const response = await api.post('/token/refresh/', { refresh });
  return response.data;  // { access }
};

// Получение профиля пользователя (защищённый эндпоинт)
export const getProfile = async (accessToken) => {
  const response = await api.get('/profile/', {
    headers: {
      'Authorization': `Bearer ${accessToken}`
    }
  });
  return response.data;
};

// Выход из системы (опционально, добавляет refresh в чёрный список)
export const logout = async (refresh) => {
```

```
  await api.post('/logout/', { refresh });
};
```

Шаг 3. Создаём сервис для управления токенами

Файл src/services/tokenService.js:

```
const TOKEN_KEY = 'access_token';
const REFRESH_KEY = 'refresh_token';

// Сохранение токенов
export const saveTokens = (access, refresh) => {
  localStorage.setItem(TOKEN_KEY, access);
  localStorage.setItem(REFRESH_KEY, refresh);
};

// Получение access-токена
export const getAccessToken = () => {
  return localStorage.getItem(TOKEN_KEY);
};

// Получение refresh-токена
export const getRefreshToken = () => {
  return localStorage.getItem(REFRESH_KEY);
};

// Удаление токенов (выход)
export const removeTokens = () => {
  localStorage.removeItem(TOKEN_KEY);
  localStorage.removeItem(REFRESH_KEY);
```

```
};

// Проверка, авторизован ли пользователь
export const isAuthenticated = () => {
  const token = getAccessToken();
  if (!token) return false;

  // Проверка, не истёк ли токен (опционально)
  try {
    const payload = JSON.parse(atob(token.split('.')[1]));
    const exp = payload.exp * 1000; // в миллисекундах
    return Date.now() < exp;
  } catch {
    return false;
  }
};
```

Шаг 4. Создаём контекст аутентификации

Файл src/contexts/AuthContext.jsx:

```
import { createContext, useContext, useState, useEffect } from 'react';
import { login, refreshToken, getProfile } from '../api/authApi';
import { saveTokens, getAccessToken, getRefreshToken, removeTokens,
isAuthenticated } from '../services/tokenService';

const AuthContext = createContext();

export const AuthProvider = ({ children }) => {
  const [user, setUser] = useState(null);
```

```
  const [isLoading, setIsLoading] = useState(true);
  const [accessToken, setAccessToken] = useState(getAccessToken());

  // Функция входа
  const handleLogin = async (username, password) => {
    try {
      const data = await login(username, password);
      saveTokens(data.access, data.refresh);
      setAccessToken(data.access);

      // Загружаем профиль пользователя
      const profile = await getProfile(data.access);
      setUser(profile);

      return { success: true };
    } catch (error) {
      return { success: false, error: error.response?.data?.detail || 'Ошибка входа' };
    }
  };

  // Функция выхода
  const handleLogout = () => {
    removeTokens();
    setAccessToken(null);
    setUser(null);
  };

  // Функция обновления access-токена
```

```
const refreshAccessToken = async () => {
  const refresh = getRefreshToken();
  if (!refresh) return false;

  try {
    const data = await refreshToken(refresh);
    saveTokens(data.access, refresh);
    setAccessToken(data.access);
    return true;
  } catch (error) {
    handleLogout();
    return false;
  }
};

// При загрузке приложения проверяем, есть ли сохранённый токен
useEffect(() => {
  const checkAuth = async () => {
    if (isAuthenticated()) {
      try {
        const profile = await getProfile(getAccessToken());
        setUser(profile);
      } catch (error) {
        // Если токен истёк, пробуем обновить
        if (error.response?.status === 401) {
          const refreshed = await refreshAccessToken();
          if (refreshed) {
            const profile = await getProfile(getAccessToken());
            setUser(profile);
```

```
        }
      }
    }
  }
  setIsLoading(false);
 };

 checkAuth();
}, []);

const value = {
 user,
 isLoading,
 isAuthenticated: !!user,
 login: handleLogin,
 logout: handleLogout,
 refreshToken: refreshAccessToken,
 accessToken,
};

return (
 <AuthContext.Provider value={value}>
  {children}
 </AuthContext.Provider>
);
};

export const useAuth = () => {
 const context = useContext(AuthContext);
```

```
  if (!context) {
    throw new Error('useAuth must be used within AuthProvider');
  }
  return context;
};
```

Шаг 5. Создаём axios-интерсептор для автоматического обновления токена

Файл src/api/axiosConfig.js:

```
import axios from 'axios';
import { getAccessToken, saveTokens, getRefreshToken, removeTokens } from '../services/tokenService';

const API_URL = 'http://localhost:8000/api';

// Создаём экземпляр axios
export const api = axios.create({
  baseURL: API_URL,
  headers: {
    'Content-Type': 'application/json',
  },
});

// Интерсептор для добавления access-токена в каждый запрос
api.interceptors.request.use(
  (config) => {
    const token = getAccessToken();
    if (token) {
```

```
      config.headers.Authorization = `Bearer ${token}`;
    }
    return config;
  },
  (error) => Promise.reject(error)
);

// Интерсептор для обработки 401 ошибок (истёкший токен)
api.interceptors.response.use(
  (response) => response,
  async (error) => {
    const originalRequest = error.config;

    // Если ошибка 401 и это не повторный запрос
    if (error.response?.status === 401 && !originalRequest._retry) {
      originalRequest._retry = true;

      try {
        const refresh = getRefreshToken();
        if (!refresh) {
          removeTokens();
          window.location.href = '/login';
          return Promise.reject(error);
        }

        // Запрашиваем новый access-токен
        const response = await axios.post(`${API_URL}/token/refresh/`, { refresh });
        const { access } = response.data;
```

```
        // Сохраняем новый токен
        saveTokens(access, refresh);

        // Повторяем исходный запрос с новым токеном
        originalRequest.headers.Authorization = `Bearer ${access}`;
        return api(originalRequest);
      } catch (refreshError) {
        // Если не удалось обновить токен - выходим
        removeTokens();
        window.location.href = '/login';
        return Promise.reject(refreshError);
      }
    }

    return Promise.reject(error);
  }
);
```

Шаг 6. Создаём компонент защищённого маршрута (PrivateRoute)

Файл src/components/PrivateRoute.jsx:

```
import { Navigate } from 'react-router-dom';
import { useAuth } from '../contexts/AuthContext';

const PrivateRoute = ({ children }) => {
  const { isAuthenticated, isLoading } = useAuth();

  // Пока проверяется аутентификация, показываем загрузку
  if (isLoading) {
```

```
    return <div style={{ textAlign: 'center', padding: '50px' }}>⏳ Проверка авторизации...</div>;
  }

  // Если не авторизован - перенаправляем на страницу входа
  if (!isAuthenticated) {
    return <Navigate to="/login" replace />;
  }

  // Если авторизован - показываем запрашиваемую страницу
  return children;
};

export default PrivateRoute
```

Шаг 7. Создаём страницу входа (LoginPage)

Файл src/pages/LoginPage.jsx:

```
import { useState } from 'react';
import { useNavigate, Navigate } from 'react-router-dom';
import { useAuth } from '../contexts/AuthContext';

const LoginPage = () => {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');
  const [isLoading, setIsLoading] = useState(false);

  const { login, isAuthenticated } = useAuth();
```

```
  const navigate = useNavigate();

  // Если уже авторизован - перенаправляем на профиль
  if (isAuthenticated) {
    return <Navigate to="/profile" replace />;
  }

  const handleSubmit = async (e) => {
    e preventDefault();
    setError('');
    setIsLoading(true);

    const result = await login(username, password);

    if (result success) {
      navigate('/profile');
    } else {
      setError(result error);
    }

    setIsLoading(false);
  };

  return (
    <div style={{ maxWidth: '400px', margin: '50px auto', padding: '20px',
border: '1px solid #ccc', borderRadius: '8px' }}>
      <h2>Вход в систему</h2>

      {error && (
```

```
            <div style={{ backgroundColor: '#ffebee', color: '#c62828', padding: '10px', borderRadius: '4px', marginBottom: '15px' }}>
              {error}
            </div>
          )}

          <form onSubmit={handleSubmit}>
            <div style={{ marginBottom: '15px' }}>
              <label style={{ display: 'block', marginBottom: '5px' }}>Имя пользователя:</label>
              <input
                type="text"
                value={username}
                onChange={(e) => setUsername(e.target.value)}
                required
                style={{ width: '100%', padding: '8px', border: '1px solid #ccc', borderRadius: '4px' }}
              />
            </div>

            <div style={{ marginBottom: '20px' }}>
              <label style={{ display: 'block', marginBottom: '5px' }}>Пароль:</label>
              <input
                type="password"
                value={password}
                onChange={(e) => setPassword(e.target.value)}
                required
```

```
                style={{ width: '100%', padding: '8px', border: '1px solid #ccc', borderRadius: '4px' }}
                />
              </div>

              <button
                type="submit"
                disabled={isLoading}
                style={{
                  width: '100%',
                  padding: '10px',
                  backgroundColor: '#007bff',
                  color: 'white',
                  border: 'none',
                  borderRadius: '4px',
                  cursor: 'pointer',
                  opacity: isLoading ? 0.7 : 1
                }}
              >
                {isLoading ? 'Вход...' : 'Войти'}
              </button>
            </form>

            <div style={{ marginTop: '15px', textAlign: 'center', fontSize: '14px', color: '#666' }}>
              <p>Тестовые данные:</p>
              <p>Логин: admin | Пароль: admin123</p>
            </div>
          </div>
```

```
  );
};

export default LoginPage;
```

Шаг 8. Создаём защищённую страницу профиля

Файл src/pages/ProfilePage.jsx:

```
import { useAuth } from '../contexts/AuthContext';

const ProfilePage = () => {
  const { user, logout, accessToken } = useAuth();

  return (
    <div style={{ maxWidth: '600px', margin: '50px auto', padding: '20px',
border: '1px solid #ccc', borderRadius: '8px' }}>
      <h2>Мой профиль</h2>

      {user && (
        <div>
          <p><strong>ID:</strong> {user.id}</p>
          <p><strong>Имя пользователя:</strong> {user.username}</p>
          <p><strong>Email:</strong> {user.email}</p>
          <p><strong>Имя:</strong> {user.first_name || '—'}</p>
          <p><strong>Фамилия:</strong> {user.last_name || '—'}</p>
          <p><strong>Активен:</strong> {user.is_active ? 'Да' : 'Нет'}</p>

          <hr />
```

```
            <details style={{ marginTop: '15px' }}>
              <summary>Информация о токене</summary>
              <pre style={{ fontSize: '12px', overflowX: 'auto', maxWidth:
'100%' }}>
                {accessToken?.substring(0, 100)}...
              </pre>
            </details>

            <button
              onClick={logout}
              style={{
                marginTop: '20px',
                padding: '10px 20px',
                backgroundColor: '#dc3545',
                color: 'white',
                border: 'none',
                borderRadius: '4px',
                cursor: 'pointer'
              }}
            >
              Выйти
            </button>
          </div>
        )}
      </div>
    );
  };

export default ProfilePage;
```

Шаг 9. Создаём публичную страницу "О сайте"

Файл src/pages/AboutPage.jsx:

```
const AboutPage = () => {
  return (
    <div style={{ maxWidth: '600px', margin: '50px auto', padding: '20px' }}>
      <h2>О сайте</h2>
      <p>Это демонстрационное приложение для изучения аутентификации в React.</p>
      <p>Технологии:</p>
      <ul>
        <li>React 18</li>
        <li>React Router v6</li>
        <li>JWT аутентификация</li>
        <li>Axios интерсепторы</li>
        <li>Context API</li>
      </ul>
      <p>Эта страница доступна всем пользователям без авторизации.</p>
    </div>
  );
};

export default AboutPage;
```

Шаг 10. Создаём компонент навигации

Файл src/components/Navbar.jsx:

```
import { Link, useNavigate } from 'react-router-dom';
import { useAuth } from '../contexts/AuthContext';
```

```
const Navbar = () => {
  const { isAuthenticated, user, logout } = useAuth();
  const navigate = useNavigate();

  const handleLogout = () => {
    logout();
    navigate('/login');
  };

  return (
    <nav style={{
      display: 'flex',
      justifyContent: 'space-between',
      alignItems: 'center',
      padding: '10px 20px',
      backgroundColor: '#333',
      color: 'white'
    }}>
      <div style={{ display: 'flex', gap: '20px' }}>
        <Link to="/" style={{ color: 'white', textDecoration: 'none' }}>Главная</Link>
        <Link to="/about" style={{ color: 'white', textDecoration: 'none' }}>О сайте</Link>
        {isAuthenticated && (
          <Link to="/profile" style={{ color: 'white', textDecoration: 'none' }}>Профиль</Link>
        )}
      </div>
```

```
      <div>
        {isAuthenticated ? (
          <div style={{ display: 'flex', gap: '15px', alignItems: 'center' }}>
            <span>👤 {user?.username}</span>
            <button onClick={handleLogout} style={{
              padding: '5px 10px',
              backgroundColor: '#dc3545',
              color: 'white',
              border: 'none',
              borderRadius: '4px',
              cursor: 'pointer'
            }}>
              Выйти
            </button>
          </div>
        ) : (
          <Link to="/login" style={{ color: 'white', textDecoration: 'none' }}>
            Войти
          </Link>
        )}
      </div>
    </nav>
  );
};

export default Navbar;
```

Шаг 11. Создаём домашнюю страницу (публичная)

Файл src/pages/HomePage.jsx:

```
import { Link } from 'react-router-dom';
import { useAuth } from '../contexts/AuthContext';

const HomePage = () => {
  const { isAuthenticated } = useAuth();

  return (
    <div style={{ maxWidth: '600px', margin: '50px auto', padding: '20px', textAlign: 'center' }}>
      <h1>Добро пожаловать!</h1>
      <p>Это демонстрационное приложение для изучения аутентификации в React.</p>

      {!isAuthenticated ? (
        <div>
          <p>Пожалуйста, войдите в систему, чтобы получить доступ к профилю.</p>
          <Link to="/login">
            <button style={{
              padding: '10px 20px',
              backgroundColor: '#007bff',
              color: 'white',
              border: 'none',
              borderRadius: '4px',
              cursor: 'pointer'
            }}>
              Войти
```

```
          </button>
        </Link>
      </div>
    ) : (
      <div>
        <p>Вы уже авторизованы! Перейдите в профиль.</p>
        <Link to="/profile">
          <button style={{
            padding: '10px 20px',
            backgroundColor: '#28a745',
            color: 'white',
            border: 'none',
            borderRadius: '4px',
            cursor: 'pointer'
          }}>
            Перейти в профиль
          </button>
        </Link>
      </div>
    )}
  </div>
);
};

export default HomePage;
```

Шаг 12. Собираем всё в App.jsx

Файл src/App.jsx:

```
import { BrowserRouter, Routes, Route } from 'react-router-dom';
```

```jsx
import { AuthProvider } from './contexts/AuthContext';
import PrivateRoute from './components/PrivateRoute';
import Navbar from './components/Navbar';
import HomePage from './pages/HomePage';
import LoginPage from './pages/LoginPage';
import ProfilePage from './pages/ProfilePage';
import AboutPage from './pages/AboutPage';

function App() {
  return (
    <BrowserRouter>
      <AuthProvider>
        <Navbar />
        <Routes>
          {/* Публичные маршруты */}
          <Route path="/" element={<HomePage />} />
          <Route path="/about" element={<AboutPage />} />
          <Route path="/login" element={<LoginPage />} />

          {/* Защищённый маршрут - доступен только авторизованным */}
          <Route path="/profile" element={
            <PrivateRoute>
              <ProfilePage />
            </PrivateRoute>
          } />
        </Routes>
      </AuthProvider>
    </BrowserRouter>
  );
```

```
}

export default App;
```

4. Настройка бэкенда (Django) для тестирования

Для работы этой лабораторной необходим Django API с JWT-аутентификацией.

```
# settings.py
INSTALLED_APPS = [
    'rest_framework',
    'rest_framework_simplejwt',
]

REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework_simplejwt.authentication.JWTAuthentication',
    ),
}

from datetime import timedelta
SIMPLE_JWT = {
    'ACCESS_TOKEN_LIFETIME': timedelta(minutes=15),
    'REFRESH_TOKEN_LIFETIME': timedelta(days=7),
}

# urls.py
from django.urls import path
```

```
from rest_framework_simplejwt.views import TokenObtainPairView, TokenRefreshView
from .views import ProfileView

urlpatterns = [
    path('api/token/', TokenObtainPairView.as_view(), name='token_obtain_pair'),
    path('api/token/refresh/', TokenRefreshView.as_view(), name='token_refresh'),
    path('api/profile/', ProfileView.as_view(), name='profile'),
]

# views.py
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework.permissions import IsAuthenticated

class ProfileView(APIView):
    permission_classes = [IsAuthenticated]

    def get(self, request):
        user = request.user
        return Response({
            'id': user.id,
            'username': user.username,
            'email': user.email,
            'first_name': user.first_name,
            'last_name': user.last_name,
            'is_active': user.is_active,
```

})

5. Индивидуальные задания

5.1. Базовый уровень (15 вариантов)

Требования: Реализовать форму входа, защищённый маршрут и страницу профиля.

Вариант	**Дополнительная страница**	**Требование**
1	Дашборд пользователя	Показывать статистику активности
2	История заказов	Список предыдущих заказов
3	Избранное	Список избранных товаров
4	Настройки профиля	Возможность изменить email
5	Мои статьи	Список статей пользователя
6	Мои комментарии	Список комментариев пользователя
7	Мои подписки	Кого подписан пользователь
8	Мои подписчики	Кто подписан на пользователя

Вариант	**Дополнительная страница**	**Требование**
9	Мои уведомления	Список уведомлений
10	Мои сообщения	Список сообщений
11	Мои друзья	Список друзей
12	Мои события	Календарь событий пользователя
13	Мои задачи	Список задач пользователя
14	Мои финансы	Баланс, транзакции
15	Мои достижения	Бейджи и достижения

5.2. Средний уровень (15 вариантов)

Требования: Добавить автоматическое обновление токена через интерсептор и обработку 401 ошибки.

Вариант	Дополнительный функционал
1	Таймер обратного отсчёта до истечения токена
2	Уведомление о скором истечении токена
3	Сохранение сессии после перезагрузки страницы

Вариант	Дополнительный функционал
4	Роль пользователя (admin/user) и разные права
5	Регистрация нового пользователя
6	Смена пароля
7	Восстановление пароля через email
8	Двухфакторная аутентификация (2FA)
9	OAuth вход (Google, GitHub)
10	Логирование всех действий пользователя
11	Блокировка аккаунта при 5 неудачных входах
12	Капча на форме входа
13	"Запомнить меня" (длительная сессия)
14	Подтверждение email при регистрации
15	Привязка номера телефона

5.3. Продвинутый уровень (15 вариантов)

Требования: Реализовать полную систему аутентификации с ролями, правами доступа и защитой маршрутов в зависимости от роли.

Вариант	**Тип приложения**	**Роли**
1	Админ-панель	Админ, менеджер, пользователь
2	Интернет-магазин	Покупатель, продавец, админ
3	Блог	Автор, редактор, читатель
4	Форум	Модератор, пользователь, гость
5	CRM	Владелец, менеджер, сотрудник
6	Образовательная платформа	Студент, преподаватель, админ
7	Медицинская система	Пациент, врач, администратор
8	Библиотека	Читатель, библиотекарь, админ
9	HR-система	Кандидат, HR-менеджер, директор
10	Банковская	Клиент, оператор,

Вариант	Тип приложения	Роли
	система	администратор
11	Система бронирования	Гость, пользователь, менеджер
12	Стриминг-платформа	Зритель, автор контента, модератор
13	Доска объявлений	Пользователь, модератор, админ
14	Канбан-доска	Участник, менеджер проекта, админ
15	Система управления задачами	Исполнитель, наблюдатель, руководитель

6. Контрольные вопросы

1. Что такое JWT-токен? Из каких частей он состоит?
2. В чём разница между access-токеном и refresh-токеном?
3. Как хранить токены на клиенте? Какие есть варианты?
4. Как создать защищённый маршрут (PrivateRoute)?
5. Как получить данные пользователя после входа?
6. Как автоматически добавлять токен в заголовок каждого запроса?
7. Как обработать ошибку 401 и обновить токен?
8. Как работает axios-интерсептор?

9. Как проверить, не истёк ли токен?
10. Как реализовать автоматический выход при истечении refresh-токена?
11. Как реализовать аутентификацию с ролями и правами?
12. Как защитить маршруты в зависимости от роли пользователя?
13. Как реализовать "запомнить меня" (длительную сессию)?
14. Как безопасно хранить refresh-токен (HttpOnly cookie)?
15. Как реализовать выход из всех устройств?

7. Требования к отчёту

Титульный лист (название работы, ФИО, группа, вариант)

Цель работы

Теоретическое обоснование (своими словами)

Выполнение практического примера (код, скриншоты)

Выполнение индивидуального задания (код, скриншоты)

Ответы на контрольные вопросы (не менее 5 по своему уровню)

Вывод

8. Критерии оценивания

Критерий	Базовый (max 35)	Средний (max 50)	Продвинутый (max 65)
Форма входа и выход	10	10	10
Хранение токена	5	5	5
Создание PrivateRoute	10	5	5

Критерий	Базовый (max 35)	Средний (max 50)	Продвинутый (max 65)
Получение профиля пользователя	10	5	5
Axios-интерсептор для токена	—	10	5
Автоматическое обновление токена	—	10	10
Обработка 401 ошибки	—	5	5
Роли пользователя и права доступа	—	—	10
Качество кода	—	—	10
Итого	35	50	65

Шкала перевода баллов в оценку:

Оценка	Базовый (35)	Средний (50)	Продвинутый (65)
5 (отлично)	32-35	46-50	59-65
4 (хорошо)	28-31	40-45	51-58

Оценка	Базовый (35)	Средний (50)	Продвинутый (65)
3 (удовлетворительно)	22-27	32-39	42-50
2 (неудовлетворительно)	0-21	0-31	0-41

Лабораторная работа №20

Формы с Formik и Yup

1. Цель работы

Научиться создавать сложные формы в React с использованием библиотек Formik и Yup. Освоить управление состоянием формы, валидацию данных, отображение ошибок и обработку отправки данных на сервер.

2. Теоретическое обоснование

2.1. Проблемы нативных форм в React

При создании форм в React без дополнительных библиотек разработчики сталкиваются с рядом проблем:

Проблема	Описание	Пример
Много повторяющегося кода	Для каждого поля нужно создавать useState, обработчик onChange	Форма с 10 полями = 30 строк кода
Ручная валидация	Нужно писать функции проверки для	Проверка email, пароля, совпадения

Проблема	Описание	Пример
	каждого поля	паролей
Отображение ошибок	Нужно управлять состоянием ошибок для каждого поля	emailError, passwordError
Обработка отправки	Нужно блокировать кнопку во время отправки	isSubmitting
Сброс формы	Нужно очищать все поля после отправки	setEmail(''), setPassword('')

2.2. Что такое Formik?

Formik - это библиотека для управления формами в React. Она берёт на себя:

1. Управление состоянием всех полей формы
2. Валидацию (синхронную и асинхронную)
3. Отслеживание изменений (dirty) и отправок (submitting)
4. Обработку отправки формы

2.3. Что такое Yup?

Yup - это библиотека для валидации схем (схемная валидация). Она позволяет объявить правила валидации в декларативном стиле.

Сравнение подходов к валидации:

Ручная валидация	Yup
if (!email) setError('Email обязателен')	yup.string().required('Email обязателен')

Ручная валидация	Yup
if (!email.includes('@')) setError('Неверный email')	yup.string().email('Неверный email')
if (password.length < 8) setError('Слишком короткий')	yup.string().min(8, 'Слишком короткий')
Много кода, легко ошибиться	Декларативно, меньше кода

2.4. Сравнение подходов к формам

```
// ✕ Без Formik (много кода)
function RegisterForm() {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [emailError, setEmailError] = useState('');
  const [isSubmitting, setIsSubmitting] = useState(false);

  const handleSubmit = async () => {
    if (!email) setEmailError('Email обязателен');
    if (!email.includes('@')) setEmailError('Неверный email');
    // ... много проверок
    setIsSubmitting(true);
    await api.register({ email, password });
    setIsSubmitting(false);
  };
```

```
  return (...);
}

// ✅ С Formik + Yup (меньше кода)
import { useFormik } from 'formik';
import * as yup from 'yup';

const validationSchema = yup.object({
  email: yup.string().email('Неверный email').required('Email обязателен'),
  password: yup.string().min(8, 'Минимум 8 символов').required('Пароль обязателен'),
});

function RegisterForm() {
  const formik = useFormik({
    initialValues: { email: '', password: '' },
    validationSchema,
    onSubmit: async (values) => {
      await api.register(values);
    },
  });

  return (...);
}
```

3. Практический пример

Что мы будем делать: Создадим форму регистрации пользователя с валидацией полей, отображением ошибок, обработкой отправки и подключением к Django API.

Шаг 1. Создание проекта и установка зависимостей

```
# Создаём проект
npm create vite@latest formik-demo -- --template react
cd formik-demo

# Устанавливаем зависимости
npm install
npm install formik yup axios
npm install react-router-dom  # для навигации

# Запускаем проект
npm run dev
```

Шаг 2. Создаём схему валидации

Файл src/schemas/validationSchemas.js:

```
import * as yup from 'yup';

// Схема валидации для формы регистрации
export const registerSchema = yup.object({
  username: yup
    .string()
    .min(3, 'Имя пользователя должно содержать минимум 3 символа')
    .max(20, 'Имя пользователя должно содержать максимум 20 символов')
    .matches(/^[a-zA-Z0-9_]+$/, 'Имя пользователя может содержать только буквы, цифры и подчеркивания')
    .required('Имя пользователя обязательно'),
```

```
  email: yup
    .string()
    .email('Введите корректный email адрес')
    .required('Email обязателен'),

  password: yup
    .string()
    .min(8, 'Пароль должен содержать минимум 8 символов')
    .matches(/[A-Z]/, 'Пароль должен содержать хотя бы одну заглавную букву')
    .matches(/[a-z]/, 'Пароль должен содержать хотя бы одну строчную букву')
    .matches(/[0-9]/, 'Пароль должен содержать хотя бы одну цифру')
    .required('Пароль обязателен'),

  confirmPassword: yup
    .string()
    .oneOf([yup.ref('password')], 'Пароли не совпадают')
    .required('Подтверждение пароля обязательно'),

  firstName: yup
    .string()
    .max(50, 'Имя слишком длинное')
    .optional(),

  lastName: yup
    .string()
    .max(50, 'Фамилия слишком длинная')
```

```
    .optional(),

  age: yup
    .number()
    .min(18, 'Вы должны быть старше 18 лет')
    .max(100, 'Введите корректный возраст')
    .typeError('Возраст должен быть числом')
    .optional(),

  agreeToTerms: yup
    .boolean()
    .oneOf([true], 'Вы должны принять условия использования')
    .required('Необходимо принять условия'),
});

// Схема валидации для формы входа
export const loginSchema = yup.object({
  username: yup
    .string()
    .required('Имя пользователя обязательно'),

  password: yup
    .string()
    .required('Пароль обязателен'),
});

// Схема валидации для формы профиля
export const profileSchema = yup.object({
  firstName: yup.string().max(50, 'Имя слишком длинное'),
```

```
  lastName: yup.string().max(50, 'Фамилия слишком длинная'),
  email: yup.string().email('Введите корректный email').required('Email обязателен'),
  bio: yup.string().max(500, 'Описание не может быть длиннее 500 символов'),
});
```

Шаг 3. Создаём API-функции

Файл src/api/authApi.js:

```
import axios from 'axios';

const API_URL = 'http://localhost:8000/api';

export const registerUser = async (userData) => {
  const response = await axios.post(`${API_URL}/register/`, userData);
  return response.data;
};

export const loginUser = async (credentials) => {
  const response = await axios.post(`${API_URL}/login/`, credentials);
  return response.data;
};
```

Шаг 4. Создаём компонент формы регистрации

Файл src/components/RegisterForm.jsx:

```
import { useState } from 'react';
import { useFormik } from 'formik';
import { registerSchema } from '../schemas/validationSchemas';
import { registerUser } from '../api/authApi';
```

```
const RegisterForm = () => {
  const [serverError, setServerError] = useState('');
  const [successMessage, setSuccessMessage] = useState('');

  const formik = useFormik({
    initialValues: {
      username: '',
      email: '',
      password: '',
      confirmPassword: '',
      firstName: '',
      lastName: '',
      age: '',
      agreeToTerms: false,
    },
    validationSchema: registerSchema,
    onSubmit: async (values, { setSubmitting, resetForm }) => {
      setServerError('');
      setSuccessMessage('');

      try {
        // Убираем confirmPassword перед отправкой на сервер
        const { confirmPassword, ...userData } = values;

        const response = await registerUser(userData);

        setSuccessMessage('Регистрация прошла успешно! Теперь вы можете войти.');
```

```
            resetForm();  // Очищаем форму

            // Через 3 секунды убираем сообщение об успехе
            setTimeout(() => setSuccessMessage(''), 3000);

          } catch (error) {
            if (error.response?.data?.username) {
              setServerError(`Имя пользователя: ${error.response.data.username}`);
            } else if (error.response?.data?.email) {
              setServerError(`Email: ${error.response.data.email}`);
            } else {
              setServerError('Ошибка регистрации. Попробуйте позже.');
            }
          } finally {
            setSubmitting(false);
          }
        },
      });

      // Получаем ошибки полей
      const getFieldError = (fieldName) => {
        return formik.touched[fieldName] && formik.errors[fieldName];
      };

      return (
        <div style={{ maxWidth: '500px', margin: '0 auto', padding: '20px' }}>
          <h2>Регистрация</h2>
```

```
        {successMessage && (
          <div style={{
            backgroundColor: '#d4edda',
            color: '#155724',
            padding: '10px',
            borderRadius: '4px',
            marginBottom: '20px'
          }}>
            ✔ {successMessage}
          </div>
        )}

        {serverError && (
          <div style={{
            backgroundColor: '#f8d7da',
            color: '#721c24',
            padding: '10px',
            borderRadius: '4px',
            marginBottom: '20px'
          }}>
            ✖ {serverError}
          </div>
        )}

        <form onSubmit={formik.handleSubmit}>
          {/* Поле: Имя пользователя */}
          <div style={{ marginBottom: '15px' }}>
            <label htmlFor="username" style={{ display: 'block', marginBottom:
'5px', fontWeight: 'bold' }}>
```

```
            Имя пользователя *
          </label>
          <input
            id="username"
            name="username"
            type="text"
            onChange={formik.handleChange}
            onBlur={formik.handleBlur}
            value={formik.values.username}
            style={{
              width: '100%',
              padding: '10px',
              border: `1px solid ${getFieldError('username') ? '#dc3545' : '#ced4da'}`,
              borderRadius: '4px'
            }}
          />
          {getFieldError('username') && (
            <div style={{ color: '#dc3545', fontSize: '12px', marginTop: '5px' }}>
              {formik.errors.username}
            </div>
          )}
          <div style={{ fontSize: '12px', color: '#6c757d', marginTop: '5px' }}>
            Только буквы, цифры и подчеркивания. От 3 до 20 символов.
          </div>
        </div>

        {/* Поле: Email */}
```

```
            <div style={{ marginBottom: '15px' }}>
              <label htmlFor="email" style={{ display: 'block', marginBottom: '5px', fontWeight: 'bold' }}>
                Email *
              </label>
              <input
                id="email"
                name="email"
                type="email"
                onChange={formik.handleChange}
                onBlur={formik.handleBlur}
                value={formik.values.email}
                style={{
                  width: '100%',
                  padding: '10px',
                  border: `1px solid ${getFieldError('email') ? '#dc3545' : '#ced4da'}`,
                  borderRadius: '4px'
                }}
              />
              {getFieldError('email') && (
                <div style={{ color: '#dc3545', fontSize: '12px', marginTop: '5px' }}>
                  {formik.errors.email}
                </div>
              )}
            </div>

            {/* Поле: Пароль */}
            <div style={{ marginBottom: '15px' }}>
```

```
              <label htmlFor="password" style={{ display: 'block', marginBottom:
'5px', fontWeight: 'bold' }}>
                Пароль *
              </label>
              <input
                id="password"
                name="password"
                type="password"
                onChange={formik handleChange}
                onBlur={formik handleBlur}
                value={formik values password}
                style={{
                  width: '100%',
                  padding: '10px',
                  border: `1px solid ${getFieldError('password') ? '#dc3545' :
'#ced4da'}`,
                  borderRadius: '4px'
                }}
              />
              {getFieldError('password') && (
                <div style={{ color: '#dc3545', fontSize: '12px', marginTop:
'5px' }}>
                  {formik errors password}
                </div>
              )}
              <div style={{ fontSize: '12px', color: '#6c757d', marginTop: '5px' }}>
                Минимум 8 символов, заглавная и строчная буквы, цифра.
              </div>
            </div>
```

```
{/* Поле: Подтверждение пароля */}
<div style={{ marginBottom: '15px' }}>
 <label htmlFor="confirmPassword" style={{ display: 'block', marginBottom: '5px', fontWeight: 'bold' }}>
  Подтвердите пароль *
 </label>
 <input
  id="confirmPassword"
  name="confirmPassword"
  type="password"
  onChange={formik handleChange}
  onBlur={formik handleBlur}
  value={formik values confirmPassword}
  style={{
   width: '100%',
   padding: '10px',
   border: `1px solid ${getFieldError('confirmPassword') ? '#dc3545' : '#ced4da'}`,
   borderRadius: '4px'
  }}
 />
 {getFieldError('confirmPassword') && (
  <div style={{ color: '#dc3545', fontSize: '12px', marginTop: '5px' }}>
   {formik errors confirmPassword}
  </div>
 )}
</div>
```

```
            {/* Поле: Имя (опционально) */}
            <div style={{ marginBottom: '15px' }}>
              <label htmlFor="firstName" style={{ display: 'block', marginBottom: '5px' }}>
                Имя (опционально)
              </label>
              <input
                id="firstName"
                name="firstName"
                type="text"
                onChange={formik.handleChange}
                onBlur={formik.handleBlur}
                value={formik.values.firstName}
                style={{
                  width: '100%',
                  padding: '10px',
                  border: `1px solid ${getFieldError('firstName') ? '#dc3545' : '#ced4da'}`,
                  borderRadius: '4px'
                }}
              />
              {getFieldError('firstName') && (
                <div style={{ color: '#dc3545', fontSize: '12px', marginTop: '5px' }}>
                  {formik.errors.firstName}
                </div>
              )}
            </div>
```

```
            {/* Поле: Фамилия (опционально) */}
            <div style={{ marginBottom: '15px' }}>
              <label htmlFor="lastName" style={{ display: 'block', marginBottom: '5px' }}>
                Фамилия (опционально)
              </label>
              <input
                id="lastName"
                name="lastName"
                type="text"
                onChange={formik.handleChange}
                onBlur={formik.handleBlur}
                value={formik.values.lastName}
                style={{
                  width: '100%',
                  padding: '10px',
                  border: `1px solid ${getFieldError('lastName') ? '#dc3545' : '#ced4da'}`,
                  borderRadius: '4px'
                }}
              />
              {getFieldError('lastName') && (
                <div style={{ color: '#dc3545', fontSize: '12px', marginTop: '5px' }}>
                  {formik.errors.lastName}
                </div>
              )}
            </div>
```

```
            {/* Поле: Возраст (опционально) */}
            <div style={{ marginBottom: '15px' }}>
              <label htmlFor="age" style={{ display: 'block', marginBottom: '5px' }}>
                Возраст (опционально)
              </label>
              <input
                id="age"
                name="age"
                type="number"
                onChange={formik.handleChange}
                onBlur={formik.handleBlur}
                value={formik.values.age}
                style={{
                  width: '100%',
                  padding: '10px',
                  border: `1px solid ${getFieldError('age') ? '#dc3545' : '#ced4da'}`,
                  borderRadius: '4px'
                }}
              />
              {getFieldError('age') && (
                <div style={{ color: '#dc3545', fontSize: '12px', marginTop: '5px' }}>
                  {formik.errors.age}
                </div>
              )}
            </div>
```

```
{/* Чекбокс: Согласие с условиями */}
<div style={{ marginBottom: '20px' }}>
 <label style={{ display: 'flex', alignItems: 'center', gap: '10px' }}>
  <input
   name="agreeToTerms"
   type="checkbox"
   onChange={formik handleChange}
   onBlur={formik handleBlur}
   checked={formik values agreeToTerms}
  />
  <span>Я принимаю <a href="/terms" target="_blank">условия использования</a> *</span>
 </label>
 {getFieldError('agreeToTerms') && (
  <div style={{ color: '#dc3545', fontSize: '12px', marginTop: '5px' }}>
   {formik errors agreeToTerms}
  </div>
 )}
</div>

{/* Кнопка отправки */}
<button
 type="submit"
 disabled={formik isSubmitting}
 style={{
  width: '100%',
  padding: '10px',
  backgroundColor: '#007bff',
```

```
            color: 'white',
            border: 'none',
            borderRadius: '4px',
            cursor: 'pointer',
            opacity: formik.isSubmitting ? 0.7 : 1
          }}
        >
          {formik.isSubmitting ? 'Регистрация...' : 'Зарегистрироваться'}
        </button>
      </form>

      {/* Отображение прогресса заполнения */}
      <div style={{ marginTop: '20px', padding: '10px', backgroundColor: '#f8f9fa', borderRadius: '4px' }}>
        <h4>Статус формы:</h4>
        <ul style={{ margin: 0, paddingLeft: '20px' }}>
          <li>Форма была изменена: {formik.dirty ? 'Да' : 'Нет'}</li>
          <li>Форма валидна: {formik.isValid ? 'Да' : 'Нет'}</li>
          <li>Количество ошибок: {Object.keys(formik.errors).length}</li>
          <li>Отправляется: {formik.isSubmitting ? 'Да' : 'Нет'}</li>
        </ul>
      </div>
    </div>
  );
};

export default RegisterForm;
```

Шаг 5. Создаём форму входа (сравнение с регистрацией)

Файл src/components/LoginForm.jsx:

```
import { useFormik } from 'formik';
import { loginSchema } from '../schemas/validationSchemas';
import { loginUser } from '../api/authApi';

const LoginForm = ({ onLoginSuccess }) => {
  const formik = useFormik({
    initialValues: {
      username: '',
      password: '',
    },
    validationSchema: loginSchema,
    onSubmit: async (values, { setSubmitting, setFieldError }) => {
      try {
        const response = await loginUser(values);

        // Сохраняем токены
        localStorage setItem('access_token', response access);
        localStorage setItem('refresh_token', response refresh);

        // Вызываем колбэк при успешном входе
        if (onLoginSuccess) {
          onLoginSuccess(response user);
        }
      } catch (error) {
        setFieldError('general', 'Неверное имя пользователя или пароль');
      } finally {
        setSubmitting(false);
```

```
    }
  },
});

return (
  <form onSubmit={formik.handleSubmit} style={{ maxWidth: '400px', margin: '0 auto', padding: '20px' }}>
    <h2>Вход в систему</h2>

    {formik.errors.general && (
      <div style={{ backgroundColor: '#f8d7da', color: '#721c24', padding: '10px', borderRadius: '4px', marginBottom: '20px' }}>
        ✕ {formik.errors.general}
      </div>
    )}

    <div style={{ marginBottom: '15px' }}>
      <label htmlFor="username">Имя пользователя</label>
      <input
        id="username"
        name="username"
        type="text"
        onChange={formik.handleChange}
        onBlur={formik.handleBlur}
        value={formik.values.username}
        style={{ width: '100%', padding: '8px', marginTop: '5px' }}
      />
      {formik.touched.username && formik.errors.username && (
```

```
            <div style={{ color: 'red', fontSize: '12px' }}>{formik errors username}</div>
            )}
          </div>

          <div style={{ marginBottom: '20px' }}>
            <label htmlFor="password">Пароль</label>
            <input
              id="password"
              name="password"
              type="password"
              onChange={formik handleChange}
              onBlur={formik handleBlur}
              value={formik values password}
              style={{ width: '100%', padding: '8px', marginTop: '5px' }}
            />
            {formik touched password && formik errors password && (
            <div style={{ color: 'red', fontSize: '12px' }}>{formik errors password}</div>
            )}
          </div>

          <button
            type="submit"
            disabled={formik isSubmitting}
            style={{ width: '100%', padding: '10px', backgroundColor: '#28a745', color: 'white', border: 'none', borderRadius: '4px' }}
          >
            {formik isSubmitting ? 'Вход...' : 'Войти'}
```

```
      </button>
    </form>
  );
};

export default LoginForm;
```

Шаг 6. Создаём форму обратной связи (с кастомной валидацией)

Файл src/components/ContactForm.jsx:

```
import { useFormik } from 'formik';
import * as yup from 'yup';

// Кастомный валидатор для проверки spam-слов
const spamWords = ['спам', 'реклама', 'казино', 'бонус'];

const contactSchema = yup.object({
  name: yup.string().min(2, 'Имя должно содержать минимум 2 символа').required('Имя обязательно'),
  email: yup.string().email('Неверный email').required('Email обязателен'),
  subject: yup.string().min(3, 'Тема должна содержать минимум 3 символа').required('Тема обязательна'),
  message: yup
    .string()
    .min(10, 'Сообщение должно содержать минимум 10 символов')
    .max(1000, 'Сообщение не может быть длиннее 1000 символов')
    .test('no-spam', 'Сообщение содержит запрещённые слова', (value) => {
      if (!value) return true;
      return !spamWords.some(word => value.toLowerCase().includes(word));
```

```
    })
    .required('Сообщение обязательно'),
  priority: yup.string().oneOf(['low', 'medium', 'high'], 'Выберите приоритет'),
});

const ContactForm = () => {
  const formik = useFormik({
    initialValues: {
      name: '',
      email: '',
      subject: '',
      message: '',
      priority: 'medium',
      subscribe: false,
    },
    validationSchema: contactSchema,
    onSubmit: async (values, { setSubmitting, resetForm }) => {
      // Имитация отправки на сервер
      await new Promise(resolve => setTimeout(resolve, 1500));

      console.log('Отправлено:', values);
      alert('Сообщение отправлено!');
      resetForm();
      setSubmitting(false);
    },
  });

  return (
```

```
<form onSubmit={formik.handleSubmit} style={{ maxWidth: '500px', margin: '0 auto', padding: '20px' }}>
  <h2>Обратная связь</h2>

  <div style={{ marginBottom: '15px' }}>
    <label>Имя *</label>
    <input
      name="name"
      onChange={formik handleChange}
      onBlur={formik handleBlur}
      value={formik values name}
      style={{ width: '100%', padding: '8px', marginTop: '5px' }}
    />
    {formik touched name && formik errors name && (
      <div style={{ color: 'red', fontSize: '12px' }}>{formik errors name}</div>
    )}
  </div>

  <div style={{ marginBottom: '15px' }}>
    <label>Email *</label>
    <input
      name="email"
      type="email"
      onChange={formik handleChange}
      onBlur={formik handleBlur}
      value={formik values email}
      style={{ width: '100%', padding: '8px', marginTop: '5px' }}
    />
```

```
          {formik.touched.email && formik.errors.email && (
            <div style={{ color: 'red', fontSize: '12px' }}>{formik.errors.email}</div>
          )}
        </div>

        <div style={{ marginBottom: '15px' }}>
          <label>Тема *</label>
          <input
            name="subject"
            onChange={formik.handleChange}
            onBlur={formik.handleBlur}
            value={formik.values.subject}
            style={{ width: '100%', padding: '8px', marginTop: '5px' }}
          />
          {formik.touched.subject && formik.errors.subject && (
            <div style={{ color: 'red', fontSize: '12px' }}>{formik.errors.subject}</div>
          )}
        </div>

        <div style={{ marginBottom: '15px' }}>
          <label>Приоритет</label>
          <select
            name="priority"
            onChange={formik.handleChange}
            value={formik.values.priority}
            style={{ width: '100%', padding: '8px', marginTop: '5px' }}
          >
```

```
            <option value="low">Низкий</option>
            <option value="medium">Средний</option>
            <option value="high">Высокий</option>
          </select>
        </div>

        <div style={{ marginBottom: '15px' }}>
          <label>Сообщение *</label>
          <textarea
            name="message"
            rows={5}
            onChange={formik.handleChange}
            onBlur={formik.handleBlur}
            value={formik.values.message}
            style={{ width: '100%', padding: '8px', marginTop: '5px' }}
          />
          {formik.touched.message && formik.errors.message && (
            <div style={{ color: 'red', fontSize: '12px' }}>{formik.errors.message}</div>
          )}
        </div>

        <div style={{ marginBottom: '20px' }}>
          <label style={{ display: 'flex', alignItems: 'center', gap: '10px' }}>
            <input
              name="subscribe"
              type="checkbox"
              onChange={formik.handleChange}
              checked={formik.values.subscribe}
```

```
          />
          Подписаться на новости
        </label>
      </div>

      <button
        type="submit"
        disabled={formik.isSubmitting}
        style={{
          width: '100%',
          padding: '10px',
          backgroundColor: '#007bff',
          color: 'white',
          border: 'none',
          borderRadius: '4px',
          cursor: 'pointer'
        }}
      >
        {formik.isSubmitting ? 'Отправка...' : 'Отправить'}
      </button>
    </form>
  );
};

export default ContactForm;
```

4. Сравнение подходов к формам

Аспект	Без библиотек	Formik + Yup
Код для 5 полей	~150 строк	~50 строк
Валидация	Ручная, много if	Декларативная схема
Ошибки	Ручное управление useState	Автоматическое
Отслеживание изменений	Ручное	Автоматическое (dirty)
Отправка (submitting)	Ручной флаг	Автоматический
Сброс формы	Ручной	resetForm()

5. Индивидуальные задания

5.1. Базовый уровень (15 вариантов)

Требования: Создать форму регистрации с валидацией полей (имя, email, пароль, подтверждение пароля).

Вариант	Дополнительное поле
1	Телефон (маска +7 999 999-99-99)
2	Дата рождения (проверка возраста 18+)

Вариант	Дополнительное поле
3	Город (выбор из списка)
4	Пол (радио-кнопки)
5	Аватар (загрузка файла)
6	Ссылка на соцсети
7	Код приглашения
8	Вопрос безопасности
9	Капча (простая математическая)
10	Подтверждение email (повторный ввод)
11	Промокод
12	Никнейм (уникальность через API)
13	Страна (выбор из списка)
14	Язык интерфейса
15	Уровень (джун/мидл/син开发者)

5.2. Средний уровень (15 вариантов)

Требования: Создать многоступенчатую форму (wizard) с валидацией на каждом шаге.

Вариант	Тип приложения	Шаги
1	Регистрация	Личные данные → Контакты → Пароль
2	Заказ товара	Товары → Доставка → Оплата
3	Анкета	Основное → Образование → Опыт
4	Бронирование	Даты → Гости → Услуги
5	Опрос	Вопрос 1 → Вопрос 2 → Вопрос 3
6	Настройки профиля	Личное → Уведомления → Безопасность
7	Создание объявления	Категория → Описание → Контакты
8	Подача заявки	Тип заявки → Данные → Файлы
9	Регистрация мероприятия	Участник → Билет → Оплата
10	Создание курса	Информация → Уроки → Цена
11	Настройка магазина	Данные → Товары → Доставка

Вариант	Тип приложения	Шаги
12	Создание портфолио	Инфо → Проекты → Контакты
13	Регистрация вакансии	Должность → Требования → Условия
14	Создание рассылки	Тема → Контент → Получатели
15	Настройка чат-бота	Приветствие → Правила → Ответы

5.3. Продвинутый уровень (15 вариантов)

Требования: Создать форму с динамическими полями (массивы полей, добавление/удаление) и интеграцией с API.

Вариант	Тип приложения	Динамические поля
1	Создание резюме	Список мест работы (+/-)
2	Добавление товаров	Характеристики (+/-)
3	Создание опроса	Варианты ответов (+/-)
4	Редактор расписания	Временные слоты (+/-)

Вариант	Тип приложения	Динамические поля
5	Карточка команды	Участники проекта (+/-)
6	Создание плейлиста	Список треков (+/-)
7	Меню ресторана	Блюда и цены (+/-)
8	Система бронирования	Гости (+/-)
9	Управление задачами	Подзадачи (+/-)
10	Создание галереи	Изображения (+/-)
11	Форма заказа	Товары (+/-)
12	Добавление курса	Модули и уроки (+/-)
13	Создание сайта	Страницы (+/-)
14	Форма контактов	Номера телефонов (+/-)
15	Настройка вопросов	Варианты ответов (+/-)

6. Контрольные вопросы

1. Какие проблемы решают Formik и Yup?

2. Как установить Formik и Yup в проект?
3. Как создать схему валидации с помощью Yup?
4. Как использовать хук useFormik?
5. Как получить доступ к значениям полей формы?
6. Как отображать ошибки валидации под полями?
7. Как обработать отправку формы?
8. Как сбросить форму после отправки?
9. Как добавить кастомный валидатор в Yup?
10. Как создать многоступенчатую форму (wizard)?
11. Как управлять динамическими полями (массивами)?
12. Как интегрировать Formik с React-Query?
13. Как использовать Formik с Material-UI?
14. Как оптимизировать производительность больших форм?

7. Требования к отчёту

Титульный лист (название работы, ФИО, группа, вариант)

Цель работы

Теоретическое обоснование (своими словами)

Выполнение практического примера (код, скриншоты)

Выполнение индивидуального задания (код, скриншоты)

Ответы на контрольные вопросы (не менее 5 по своему уровню)

Вывод

8. Критерии оценивания

Критерий	Базовый (max 35)	Средний (max 50)	Продвинутый (max 65)
Установка	5	5	5

Критерий	Базовый (max 35)	Средний (max 50)	Продвинутый (max 65)
Formik и Yup			
Создание схемы валидации	10	10	10
Использование useFormik	10	10	10
Отображение ошибок	10	10	5
Многоступенчатая форма	—	15	5
Динамические поля (массивы)	—	—	15
Асинхронная валидация	—	—	10
Качество кода	—	—	5
Итого	35	50	65

Шкала перевода баллов в оценку:

Оценка	Базовый (35)	Средний (50)	Продвинутый (65)
5 (отлично)	32-35	46-50	59-65

Оценка	Базовый (35)	Средний (50)	Продвинутый (65)
4 (хорошо)	28-31	40-45	51-58
3 (удовлетворительно)	22-27	32-39	42-50
2 (неудовлетворительно)	0-21	0-31	0-41

Лабораторная работа №21

UI-библиотека и компонентный дизайн (Material-UI)

1. Цель работы

Освоить использование библиотеки Material-UI (MUI) для создания современного, адаптивного и удобного пользовательского интерфейса. Научиться устанавливать и настраивать Material-UI, создавать кастомные темы, использовать основные компоненты (Button, TextField, Modal, Snackbar, Card, AppBar и др.), а также реализовывать адаптивный дизайн.

2. Теоретическое обоснование

2.1. Что такое UI-библиотека?

UI-библиотека - это набор готовых, переиспользуемых компонентов для построения пользовательского интерфейса. Вместо того чтобы писать кнопки, поля ввода, модальные окна с нуля, вы используете готовые компоненты, которые уже стилизованы, адаптивны и протестированы.

2.2. Почему Material-UI?

Material-UI (сейчас называется MUI) - это самая популярная UI-библиотека для React, реализующая дизайн-систему Google Material Design.

Альтернативы	Особенности
M a t e r i a l - U I (MUI)	Самый популярный, большое сообщество, много компонентов
Ant Design	Популярен в Китае, корпоративный стиль
Chakra UI	Проще, легче, лучше для небольших проектов
Bootstrap React	На основе Bootstrap, привычен веб-разработчикам
Semantic UI	Красивый, но менее популярный

2.3. Преимущества Material-UI

Преимущество	Описание
Готовые компоненты	Кнопки, поля ввода, модальные окна, таблицы и т.д.
Адаптивность	Компоненты адаптируются под разные размеры экранов
Темизация	Легко менять цвета, шрифты, скругления
Тёмная тема	Встроенная поддержка светлой/тёмной темы
TypeScript	Отличная поддержка типов

Преимущество	Описание
Большое сообщество	Много примеров, документации, компонентов

2.4. Основные компоненты Material-UI

Компонент	Назначение	Аналог в HTML
Button	Кнопка	<button>
TextField	Поле ввода	<input>, <textarea>
Modal	Модальное окно	dialog
Snackbar	Всплывающее уведомление	—
Card	Карточка контента	div с тенями и скруглениями
AppBar	Верхняя панель	header
Drawer	Боковое меню	—

Компонент	Назначение	Аналог в HTML
Table	Таблица	<table>
Select	Выпадающий список	<select>
Checkbox	Чекбокс	<input type="checkbox">
Radio	Радио-кнопка	<input type="radio">
Switch	Переключатель	—
Grid	Сетка (адаптивная)	—
Box	Универсальный контейнер	
Typography	Текст с предустановленными стилями	<h1>, <p>

3. Практический пример

Что мы будем делать: Создадим приложение для управления задачами (To-Do лист) с использованием Material-UI. Реализуем:

1. Адаптивную сетку
2. Кастомную тему (цвета, шрифты)
3. Компоненты: Button, TextField, Modal, Snackbar, Card, AppBar
4. Поддержку светлой и тёмной темы

Шаг 1. Создание проекта и установка зависимостей

```
# Создаём проект
npm create vite@latest mui-demo -- --template react
cd mui-demo

# Устанавливаем зависимости
npm install

# Устанавливаем Material-UI и иконки
npm install @mui/material @emotion/react @emotion/styled
npm install @mui/icons-material

# Устанавливаем дополнительно (для темной темы и утилит)
npm install @mui/material @mui/styled-engine-sc

# Запускаем проект
npm run dev
```

Шаг 2. Настройка темы и провайдера

Файл src/theme/theme.js:

```
import { createTheme } from '@mui/material/styles';

// Светлая тема
export const lightTheme = createTheme({
  palette: {
    mode: 'light',
    primary: {
      main: '#1976d2',      // основной синий
```

```
    light: '#42a5f5',
    dark: '#1565c0',
  },
  secondary: {
    main: '#9c27b0',       // фиолетовый
    light: '#ba68c8',
    dark: '#7b1fa2',
  },
  background: {
    default: '#f5f5f5',
    paper: '#ffffff',
  },
  error: {
    main: '#d32f2f',
  },
  success: {
    main: '#2e7d32',
  },
  warning: {
    main: '#ed6c02',
  },
  info: {
    main: '#0288d1',
  },
},
typography: {
  fontFamily: '"Roboto", "Helvetica", "Arial", sans-serif',
  h1: {
    fontSize: '2.5rem',
```

```
    fontWeight: 600,
  },
  h2: {
    fontSize: '2rem',
    fontWeight: 500,
  },
  button: {
    textTransform: 'none',  // кнопки не должны быть в верхнем регистре
  },
},
shape: {
  borderRadius: 8,
},
components: {
  MuiButton: {
    styleOverrides: {
      root: {
        borderRadius: 20,     // скруглённые кнопки
      },
    },
  },
  MuiCard: {
    styleOverrides: {
      root: {
        borderRadius: 12,
      },
    },
  },
},
```

```
});

// Тёмная тема
export const darkTheme = createTheme({
  palette: {
    mode: 'dark',
    primary: {
      main: '#90caf9',
      light: '#e3f2fd',
      dark: '#42a5f5',
    },
    secondary: {
      main: '#ce93d8',
      light: '#f3e5f5',
      dark: '#ab47bc',
    },
    background: {
      default: '#121212',
      paper: '#1e1e1e',
    },
  },
  typography: {
    fontFamily: '"Roboto", "Helvetica", "Arial", sans-serif',
    button: {
      textTransform: 'none',
    },
  },
  shape: {
    borderRadius: 8,
```

```
  },
  components: {
    MuiButton: {
      styleOverrides: {
        root: {
          borderRadius: 20,
        },
      },
    },
    MuiCard: {
      styleOverrides: {
        root: {
          borderRadius: 12,
        },
      },
    },
  },
});
```

Шаг 3. Создаём контекст для темы

Файл src/contexts/ThemeContext.jsx:

```
import { createContext, useContext, useState, useMemo } from 'react';
import { ThemeProvider as MuiThemeProvider } from '@mui/material/styles';
import CssBaseline from '@mui/material/CssBaseline';
import { lightTheme, darkTheme } from '../theme/theme';

const ThemeContext = createContext();
```

```
export const ThemeProvider = ({ children }) => {
  const [isDarkMode, setIsDarkMode] = useState(false);

  const toggleTheme = () => {
    setIsDarkMode(prev => !prev);
  };

  const theme = useMemo(() => isDarkMode ? darkTheme : lightTheme, [isDarkMode]);

  return (
    <ThemeContext.Provider value={{ isDarkMode, toggleTheme }}>
      <MuiThemeProvider theme={theme}>
        <CssBaseline /> {/* Сброс стилей и применение темы */}
        {children}
      </MuiThemeProvider>
    </ThemeContext.Provider>
  );
};

export const useTheme = () => {
  const context = useContext(ThemeContext);
  if (!context) {
    throw new Error('useTheme must be used within ThemeProvider');
  }
  return context;
};
```

Шаг 4. Создаём компонент модального окна (TaskModal)

Файл src/components/TaskModal.jsx:

```
import { useState, useEffect } from 'react';
import {
  Dialog,
  DialogTitle,
  DialogContent,
  DialogActions,
  TextField,
  Button,
  FormControlLabel,
  Switch,
  Box,
  Alert,
} from '@mui/material';

const TaskModal = ({ open, onClose, onSave, task, title }) => {
  const [formData, setFormData] = useState({
    title: '',
    description: '',
    priority: 'medium',
    completed: false,
  });
  const [errors, setErrors] = useState({});

  // Заполняем форму при редактировании
  useEffect(() => {
    if (task) {
```

```
      setFormData({
        title: task.title || '',
        description: task.description || '',
        priority: task.priority || 'medium',
        completed: task.completed || false,
      });
    } else {
      setFormData({
        title: '',
        description: '',
        priority: 'medium',
        completed: false,
      });
    }
    setErrors({});
  }, [task, open]);

  const handleChange = (field) => (event) => {
    const value = field === 'completed' ? event.target.checked : event.target.value;
    setFormData(prev => ({ ...prev, [field]: value }));
    if (errors[field]) {
      setErrors(prev => ({ ...prev, [field]: '' }));
    }
  };

  const validate = () => {
    const newErrors = {};
    if (!formData.title.trim()) {
```

```
    newErrors title = 'Название задачи обязательно';
  }
  setErrors(newErrors);
  return Object keys(newErrors) length === 0;
};

const handleSave = () => {
  if (validate()) {
    onSave(formData);
    onClose();
  }
};

return (
  <Dialog open={open} onClose={onClose} maxWidth="sm" fullWidth>
    <DialogTitle>{title || 'Создать задачу'}</DialogTitle>
    <DialogContent>
      <Box sx={{ display: 'flex', flexDirection: 'column', gap: 2, mt: 1 }}>
        <TextField
          label="Название задачи"
          value={formData title}
          onChange={handleChange('title')}
          error={!!errors title}
          helperText={errors title}
          fullWidth
          required
          autoFocus
        />
```

```
<TextField
  label="Описание"
  value={formData description}
  onChange={handleChange('description')}
  multiline
  rows={3}
  fullWidth
/>

<TextField
  label="Приоритет"
  value={formData priority}
  onChange={handleChange('priority')}
  select
  fullWidth
  SelectProps={{
    native: true,
  }}
>
  <option value="low">Низкий</option>
  <option value="medium">Средний</option>
  <option value="high">Высокий</option>
</TextField>

<FormControlLabel
  control={
    <Switch
      checked={formData completed}
      onChange={handleChange('completed')}
```

```
            />
          }
          label="Задача выполнена"
        />
      </Box>
    </DialogContent>
    <DialogActions>
      <Button onClick={onClose} color="inherit">
        Отмена
      </Button>
      <Button onClick={handleSave} variant="contained" color="primary">
        Сохранить
      </Button>
    </DialogActions>
  </Dialog>
);
};

export default TaskModal;
```

Шаг 5. Создаём карточку задачи (TaskCard)

Файл src/components/TaskCard.jsx:

```
import {
  Card,
  CardContent,
  CardActions,
  Typography,
  Button,
```

```
  Chip,
  Box,
  IconButton,
} from '@mui/material';
import EditIcon from '@mui/icons-material/Edit';
import DeleteIcon from '@mui/icons-material/Delete';
import CheckCircleIcon from '@mui/icons-material/CheckCircle';
import RadioButtonUncheckedIcon from '@mui/icons-material/RadioButtonUnchecked';

const priorityColors = {
  low: '#4caf50',    // зелёный
  medium: '#ff9800', // оранжевый
  high: '#f44336',   // красный
};

const priorityLabels = {
  low: 'Низкий',
  medium: 'Средний',
  high: 'Высокий',
};

const TaskCard = ({ task, onEdit, onDelete, onToggleComplete }) => {
  return (
    <Card
      sx={{
        height: '100%',
        display: 'flex',
        flexDirection: 'column',
```

```
            opacity: task completed ? 0.7 : 1,
            transition: 'all 0.3s ease',
            '&:hover': {
              transform: 'translateY(-4px)',
              boxShadow: 4,
            },
          }}
        >
          <CardContent sx={{ flexGrow: 1 }}>
            <Box sx={{ display: 'flex', alignItems: 'center', justifyContent: 'space-between', mb: 1 }}>
              <Typography variant="h6" component="h3" sx={{ textDecoration: task.completed ? 'line-through' : 'none' }}>
                {task title}
              </Typography>
              <Chip
                label={priorityLabels[task priority]}
                size="small"
                sx={{
                  backgroundColor: priorityColors[task priority],
                  color: 'white',
                  fontWeight: 'bold',
                }}
              />
            </Box>

            {task description && (
              <Typography variant="body2" color="text.secondary" sx={{ mb: 2 }}>
```

```
            {task.description}
          </Typography>
        )}

        <Typography variant="caption" color="text.secondary">
          Создано: {new Date(task.createdAt).toLocaleDateString()}
        </Typography>
      </CardContent>

      <CardActions>
        <IconButton
          onClick={() => onToggleComplete(task.id)}
          color={task.completed ? 'success' : 'default'}
          size="small"
        >
          {task.completed ? <CheckCircleIcon /> : <RadioButtonUncheckedIcon />}
        </IconButton>

        <IconButton onClick={() => onEdit(task)} color="primary" size="small">
          <EditIcon />
        </IconButton>

        <IconButton onClick={() => onDelete(task.id)} color="error" size="small">
          <DeleteIcon />
        </IconButton>
```

```
        <Typography variant="caption" sx={{ ml: 'auto' }}>
          {task.completed ? '✓ Выполнено' : '○ В процессе'}
        </Typography>
      </CardActions>
    </Card>
  );
};

export default TaskCard;
```

Шаг 6. Создаём главный компонент (AppBar и список задач)
Файл src/App.jsx:

```
import { useState, useEffect } from 'react';
import {
  Container,
  AppBar,
  Toolbar,
  Typography,
  Button,
  Grid,
  Box,
  TextField,
  InputAdornment,
  Fab,
  Snackbar,
  Alert,
  useMediaQuery,
  IconButton,
```

```
} from '@mui/material';
import {
  Add as AddIcon,
  Search as SearchIcon,
  Brightness4 as DarkModeIcon,
  Brightness7 as LightModeIcon,
} from '@mui/icons-material';
import { useTheme as useMuiTheme } from '@mui/material/styles';
import TaskCard from './components/TaskCard';
import TaskModal from './components/TaskModal';
import { useTheme } from './contexts/ThemeContext';

function App() {
  const [tasks, setTasks] = useState([]);
  const [filteredTasks, setFilteredTasks] = useState([]);
  const [searchQuery, setSearchQuery] = useState('');
  const [modalOpen, setModalOpen] = useState(false);
  const [editingTask, setEditingTask] = useState(null);
  const [snackbar, setSnackbar] = useState({ open: false, message: '', severity:
'success' });
  const [filter, setFilter] = useState('all'); // all, active, completed

  const { isDarkMode, toggleTheme } = useTheme();
  const muiTheme = useMuiTheme();
  const isMobile = useMediaQuery(muiTheme.breakpoints.down('sm'));

  // Загрузка задач из localStorage
  useEffect(() => {
    const savedTasks = localStorage.getItem('tasks');
```

```
if (savedTasks) {
  setTasks(JSON.parse(savedTasks));
} else {
  // Пример данных для демонстрации
  const demoTasks = [
    {
      id: 1,
      title: 'Изучить Material-UI',
      description: 'Освоить основные компоненты и тему',
      priority: 'high',
      completed: false,
      createdAt: new Date().toISOString(),
    },
    {
      id: 2,
      title: 'Создать кастомную тему',
      description: 'Настроить цвета и шрифты',
      priority: 'medium',
      completed: false,
      createdAt: new Date().toISOString(),
    },
    {
      id: 3,
      title: 'Реализовать адаптивный дизайн',
      description: 'Использовать Grid и useMediaQuery',
      priority: 'low',
      completed: true,
      createdAt: new Date().toISOString(),
    },
```

```
    ];
    setTasks(demoTasks);
  }
}, []);

// Сохранение в localStorage
useEffect(() => {
  localStorage.setItem('tasks', JSON.stringify(tasks));
}, [tasks]);

// Фильтрация и поиск
useEffect(() => {
  let result = tasks;

  // Фильтр по статусу
  if (filter === 'active') {
    result = result.filter(task => !task.completed);
  } else if (filter === 'completed') {
    result = result.filter(task => task.completed);
  }

  // Поиск
  if (searchQuery) {
    result = result.filter(task =>
      task.title.toLowerCase().includes(searchQuery.toLowerCase()) ||
      task.description?.toLowerCase().includes(searchQuery.toLowerCase())
    );
  }
```

```
    setFilteredTasks(result);
  }, [tasks, filter, searchQuery]);

  const addTask = (taskData) => {
    const newTask = {
      ...taskData,
      id: Date.now(),
      createdAt: new Date().toISOString(),
    };
    setTasks([newTask, ...tasks]);
    setSnackbar({ open: true, message: 'Задача добавлена', severity: 'success' });
  };

  const updateTask = (taskData) => {
    setTasks(tasks.map(task =>
      task.id === editingTask.id
        ? { ...task, ...taskData }
        : task
    ));
    setEditingTask(null);
    setSnackbar({ open: true, message: 'Задача обновлена', severity: 'success' });
  };

  const deleteTask = (id) => {
    setTasks(tasks.filter(task => task.id !== id));
    setSnackbar({ open: true, message: 'Задача удалена', severity: 'warning' });
```

```
};

const toggleComplete = (id) => {
  setTasks(tasks.map(task =>
    task.id === id
      ? { ...task, completed: !task.completed }
      : task
  ));
};

const handleSaveTask = (taskData) => {
  if (editingTask) {
    updateTask(taskData);
  } else {
    addTask(taskData);
  }
};

const handleEdit = (task) => {
  setEditingTask(task);
  setModalOpen(true);
};

const handleCloseSnackbar = () => {
  setSnackbar({ ...snackbar, open: false });
};

const getTaskCounts = () => ({
  total: tasks.length,
```

```
    active: tasks.filter(t => !t.completed).length,
    completed: tasks.filter(t => t.completed).length,
  });

  const counts = getTaskCounts();

  return (
    <Box sx={{ display: 'flex', flexDirection: 'column', minHeight: '100vh' }}>
      <AppBar position="sticky">
        <Toolbar>
          <Typography variant="h6" sx={{ flexGrow: 1 }}>
            📋 Менеджер задач
          </Typography>

          <IconButton color="inherit" onClick={toggleTheme}>
            {isDarkMode ? <LightModeIcon /> : <DarkModeIcon />}
          </IconButton>
        </Toolbar>
      </AppBar>

      <Container maxWidth="lg" sx={{ mt: 4, mb: 4, flex: 1 }}>
        {/* Поиск и фильтры */}
        <Grid container spacing={2} sx={{ mb: 4 }}>
          <Grid item xs={12} md={6}>
            <TextField
              fullWidth
              placeholder="Поиск задач..."
              value={searchQuery}
```

```
            onChange={(e) => setSearchQuery(e.target.value)}
            InputProps={{
              startAdornment: (
                <InputAdornment position="start">
                  <SearchIcon />
                </InputAdornment>
              ),
            }}
          />
        </Grid>

        <Grid item xs={12} md={6}>
          <Box sx={{ display: 'flex', gap: 1, justifyContent: { xs: 'flex-start', md: 'flex-end' } }}>
            <Button
              variant={filter === 'all' ? 'contained' : 'outlined'}
              onClick={() => setFilter('all')}
            >
              Все ({counts.total})
            </Button>
            <Button
              variant={filter === 'active' ? 'contained' : 'outlined'}
              onClick={() => setFilter('active')}
              color="warning"
            >
              Активные ({counts.active})
            </Button>
            <Button
              variant={filter === 'completed' ? 'contained' : 'outlined'}
```

```
                onClick={() => setFilter('completed')}
                color="success"
              >
                Выполненные ({counts.completed})
              </Button>
            </Box>
          </Grid>
        </Grid>

        {/* Список задач */}
        {filteredTasks.length === 0 ? (
          <Box sx={{ textAlign: 'center', py: 8 }}>
            <Typography variant="h5" color="text.secondary" gutterBottom>
              {searchQuery ? 'Ничего не найдено' : 'Задач пока нет'}
            </Typography>
            <Typography variant="body2" color="text.secondary">
              {searchQuery ? 'Попробуйте изменить поисковый запрос' :
'Нажмите на кнопку +, чтобы добавить задачу'}
            </Typography>
          </Box>
        ) : (
          <Grid container spacing={3}>
            {filteredTasks.map(task => (
              <Grid item xs={12} sm={6} md={4} key={task.id}>
                <TaskCard
                  task={task}
                  onEdit={handleEdit}
                  onDelete={deleteTask}
                  onToggleComplete={toggleComplete}
```

```
          />
        </Grid>
      ))}
    </Grid>
  )}
</Container>

{/* Кнопка добавления (адаптивная) */}
<Fab
  color="primary"
  aria-label="add"
  onClick={() => {
    setEditingTask(null);
    setModalOpen(true);
  }}
  sx={{
    position: 'fixed',
    bottom: 16,
    right: 16,
    ...(isMobile && { bottom: 80 }),
  }}
>
  <AddIcon />
</Fab>

{/* Модальное окно */}
<TaskModal
  open={modalOpen}
  onClose={() => {
```

```
            setModalOpen(false);
            setEditingTask(null);
          }}
          onSave={handleSaveTask}
          task={editingTask}
          title={editingTask ? 'Редактировать задачу' : 'Новая задача'}
        />

        {/* Уведомления */}
        <Snackbar
          open={snackbar.open}
          autoHideDuration={3000}
          onClose={handleCloseSnackbar}
          anchorOrigin={{ vertical: 'bottom', horizontal: 'center' }}
        >
          <Alert onClose={handleCloseSnackbar} severity={snackbar.severity}
sx={{ width: '100%' }}>
            {snackbar.message}
          </Alert>
        </Snackbar>
      </Box>
    );
  }

export default App;
```

Шаг 7. Главный файл main.jsx

Файл src/main.jsx:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import { ThemeProvider } from './contexts/ThemeContext';
import App from './App';

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <ThemeProvider>
      <App />
    </ThemeProvider>
  </React.StrictMode>
);
```

Шаг 8. Запуск и проверка

```
npm run dev
```

Что должно работать:

Приложение с Material-UI компонентами

Переключение светлой/тёмной темы

Добавление, редактирование, удаление задач

Поиск по названию и описанию

Фильтрация по статусу

Адаптивный дизайн (меняется на мобильных устройствах)

Всплывающие уведомления

Сохранение задач в localStorage

4. Адаптивный дизайн с Material-UI

4.1. Grid система (адаптивная сетка)

```
<Grid container spacing={2}>
  <Grid item xs={12} sm={6} md={4} lg={3}>
    {/* На мобильных (xs) - 12 колонок (полная ширина) */}
    {/* На планшетах (sm) - 6 колонок (2 элемента в ряд) */}
    {/* На десктопах (md) - 4 колонки (3 элемента в ряд) */}
    {/* На больших экранах (lg) - 3 колонки (4 элемента в ряд) */}
  </Grid>
</Grid>
```

4.2. useMediaQuery (условный рендеринг)

```
const theme = useTheme();
const isMobile = useMediaQuery(theme.breakpoints.down('sm'));

return (
  <Box>
    {isMobile ? (
      <MobileMenu />
    ) : (
      <DesktopMenu />
    )}
  </Box>
);
```

4.3. Breakpoints (точки перелома)

Breakpoint	Ширина экрана	Описание
xs	0px - 600px	Телефоны
sm	600px - 900px	Планшеты
md	900px - 1200px	Десктопы
lg	1200px - 1536px	Большие десктопы
xl	1536px+	Очень большие экраны

5. Индивидуальные задания

5.1. Базовый уровень (15 вариантов)

Требования: Создать приложение с использованием Material-UI: AppBar, Grid, Button, Card, TextField.

Вариант	Тип приложения	Компоненты
1	Карточки товаров	Card, Button, Grid
2	Список пользователей	Avatar, List, Divider
3	Галерея изображений	ImageList, CardMedia

Вариант	Тип приложения	Компоненты
4	Контактная форма	TextField, Button, Alert
5	Калькулятор	TextField, Button, Typography
6	Книжный каталог	Card, Rating, Chip
7	Список фильмов	Card, Modal, Chip
8	Рецепты	Card, Accordion, Chip
9	Расписание	Table, Button, Chip
10	Погода	Card, Divider, Typography
11	Статистика	Card, LinearProgress, Chip
12	Админ-панель	Table, Chip, IconButton
13	Музыкальный плейлист	List, Avatar, IconButton
14	Блог	Card, Avatar, Button

Вариант	Тип приложения	Компоненты
15	Чат	List, TextField, IconButton

5.2. Средний уровень (15 вариантов)

Требования: Добавить кастомную тему, тёмную/светлую тему, модальное окно (Modal) и уведомления (Snackbar).

Вариант	Дополнительный функционал
1	Просмотр деталей товара в модальном окне
2	Редактирование профиля пользователя
3	Создание и редактирование постов
4	Добавление комментариев
5	Загрузка и предпросмотр изображений
6	Экспорт данных в PDF/Excel
7	Импорт данных из файла
8	Валидация форм с уведомлениями
9	Подтверждение удаления через диалог
10	Фильтрация с использованием слайдера (Slider)
11	Сортировка с помощью Select

Вариант	Дополнительный функционал
12	Пагинация с Pagination
13	Степпер (Stepper) для оформления заказа
14	Вкладки (Tabs) для категорий
15	Древо (TreeView) для категорий

5.3. Продвинутый уровень (15 вариантов)

Требования: Реализовать полноценное приложение с Drawer (боковое меню), Data Grid (таблица), Charts (графики), адаптивным дизайном.

Вариант	Тип приложения	Продвинутые компоненты
1	Админ-панель	Drawer, DataGrid, Charts
2	Дашборд	Drawer, Charts, Cards
3	CRM-система	Drawer, DataGrid, Tabs
4	Система тикетов	Drawer, DataGrid, Stepper
5	Складской учёт	Drawer, DataGrid, Slider
6	Управление проектами	Drawer, TreeView, Stepper

Вариант	Тип приложения	Продвинутые компоненты
7	HR-система	Drawer, DataGrid, AvatarGroup
8	Система бронирования	Drawer, DatePicker, Stepper
9	Платформа обучения	Drawer, Accordion, Rating
10	Финансовый трекер	Drawer, Charts, DataGrid
11	Система заказов	Drawer, Stepper, Timeline
12	Медицинская система	Drawer, DataGrid, Rating
13	Недвижимость	Drawer, Slider, DataGrid
14	Маркетплейс	Drawer, Card, DataGrid
15	Социальная сеть	Drawer, AvatarGroup, Timeline

6. Контрольные вопросы

1. Что такое UI-библиотека и для чего она нужна?
2. Как установить Material-UI в проект?
3. Как создать кастомную тему в Material-UI?
4. Какие основные компоненты Material-UI вы знаете?
5. Как использовать компонент Button?
6. Как добавить иконки в Material-UI?

7. Как реализовать переключение между светлой и тёмной темой?
8. Как сделать адаптивный дизайн с помощью Grid?
9. Как использовать useMediaQuery для адаптации?
10. Как создать модальное окно (Dialog/Modal)?
11. Как добавить всплывающие уведомления (Snackbar)?
12. Как стилизовать компоненты через sx prop?
13. Как создать боковое меню (Drawer)?
14. Как использовать Data Grid для сложных таблиц?
15. Как интегрировать Charts (графики) в Material-UI?

7. Требования к отчёту

Титульный лист (название работы, ФИО, группа, вариант)

Цель работы

Теоретическое обоснование (своими словами)

Выполнение практического примера (код, скриншоты)

Выполнение индивидуального задания (код, скриншоты)

Ответы на контрольные вопросы (не менее 5 по своему уровню)

Вывод

8. Критерии оценивания

Критерий	Базовый (max 35)	Средний (max 50)	Продвинутый (max 65)
Установка и настройка MUI	5	5	5
Создание кастомной темы	10	10	10
Использование базовых компонентов	15	10	5
Модальное окно (Modal)	—	10	5
Тёмная/светлая тема	—	10	5
Snackbar (уведомления)	—	5	5
Drawer (боковое меню)	—	—	10
Data Grid / Charts	—	—	15
Качество кода	5	5	5
Итого	35	50	65

Шкала перевода баллов в оценку:

Оценка	Базовый (35)	Средний (50)	Продвинутый (65)
5 (отлично)	32-35	46-50	59-65
4 (хорошо)	28-31	40-45	51-58
3 (удовлетворительно)	22-27	32-39	42-50
2 (неудовлетворительно)	0-21	0-31	0-41

Лабораторная работа №22

Тестирование React-компонентов

1. Цель работы

Научиться писать unit-тесты для React-компонентов с использованием Jest и React Testing Library. Освоить настройку тестового окружения, тестирование отрисовки компонентов, проверку взаимодействий (клики, ввод данных) и мокирование API-запросов с помощью MSW (Mock Service Worker).

2. Теоретическое обоснование

2.1. Что такое тестирование и зачем оно нужно?

Тестирование - это процесс проверки корректности работы программного кода. В React-приложениях тесты помогают убедиться, что компоненты правильно отображаются, корректно обрабатывают события пользователя и правильно взаимодействуют с API.

Почему важно тестировать:

Причина	Объяснение
Обнаружение багов	Тесты находят ошибки до того, как они попадут в продакшн
Рефакторинг	Можно безопасно изменять код, зная, что тесты его не сломают
Документация	Тесты показывают, как должны работать компоненты
Уверенность	Уверенность, что приложение работает корректно

2.2. Виды тестов

Вид теста	Что проверяет	Кого тестирует	Пример
Unit-тесты	Отдельные функции, компоненты	Разработчик	Проверка, что кнопка рендерится
Интеграционные тесты	Взаимодействие компонентов	Разработчик	Проверка формы отправки данных
E2E-тесты	Полный сценарий пользователя	Пользователь	Регистрация → логин → покупка

2.3. Инструменты для тестирования React

Инструмент	Назначение	Почему используем
Jest	Тестовый фреймворк (запуск тестов, assertions, моки)	Встроен в Vite/React
React Testing Library (RTL)	Тестирование компонентов (отрисовка, события)	Поощряет тестирование поведения, а не реализации
MSW (Mock Service Worker)	Мокирование API-запросов	Перехватывает реальные запросы на сетевом уровне

2.4. Принципы React Testing Library

Главный принцип: тестировать поведение, а не реализацию.

Плохо (тестируем реализацию)	**Хорошо (тестируем поведение)**
Проверка внутреннего state	Проверка того, что видит пользователь
Поиск по классам CSS	Поиск по тексту, ролям, aria-меткам

Плохо (тестируем реализацию)	Хорошо (тестируем поведение)
Вызов методов компонента	Имитация действий пользователя

3. Практический пример

Что мы будем делать: Создадим приложение с формой входа и списком задач, а затем напишем для него тесты.

Шаг 1. Создание проекта с тестовым окружением

```
# Создаём проект с шаблоном, включающим тесты
npm create vite@latest testing-demo -- --template react
cd testing-demo

# Устанавливаем зависимости
npm install

# Устанавливаем дополнительные инструменты для тестирования
npm install -D @testing-library/jest-dom @testing-library/user-event
npm install -D msw

# Запускаем тесты (для проверки)
npm test
```

Примечание: Vite по умолчанию использует Vitest (аналог Jest). Команда npm test должна запустить тестовый runner.

Шаг 2. Создаём простой компонент Button

Файл src/components/Button.jsx:

```
const Button = ({ children, onClick, disabled = false, variant = 'primary' }) => {
  const styles = {
    primary: {
      backgroundColor: '#007bff',
      color: 'white',
    },
    secondary: {
      backgroundColor: '#6c757d',
      color: 'white',
    },
    danger: {
      backgroundColor: '#dc3545',
      color: 'white',
    },
  };

  return (
    <button
      onClick={onClick}
      disabled={disabled}
      style={{
        padding: '8px 16px',
        border: 'none',
        borderRadius: '4px',
        cursor: disabled ? 'not-allowed' : 'pointer',
        opacity: disabled ? 0.6 : 1,
        ...styles[variant],
```

```
      }}
      data-testid="custom-button"
    >
      {children}
    </button>
  );
};

export default Button;
```

Файл src/components/Button.test.jsx (тесты для кнопки):

```
import { render, screen, fireEvent } from '@testing-library/react';
import userEvent from '@testing-library/user-event';
import Button from './Button';

describe('Button Component', () => {
  // 1. Тест отрисовки
  test('renders button with children text', () => {
    render(<Button>Click me</Button>);
    const button = screen.getByText('Click me');
    expect(button).toBeInTheDocument();
  });

  // 2. Тест обработки клика
  test('calls onClick handler when clicked', () => {
    const handleClick = jest.fn();  // создаём функцию-шпион
    render(<Button onClick={handleClick}>Click me</Button>);
```

```
  const button = screen.getByText('Click me');
  fireEvent.click(button);

  expect(handleClick).toHaveBeenCalledTimes(1);
});

// 3. Тест disabled состояния
test('button is disabled when disabled prop is true', () => {
  render(<Button disabled>Click me</Button>);
  const button = screen.getByText('Click me');

  expect(button).toBeDisabled();
});

// 4. Тест разных вариантов (primary, secondary, danger)
test('applies correct styles for primary variant', () => {
  render(<Button variant="primary">Primary</Button>);
  const button = screen.getByText('Primary');

  expect(button).toHaveStyle({ backgroundColor: '#007bff' });
});

// 5. Тест с userEvent (более реалистичное взаимодействие)
test('handles click with userEvent', async () => {
  const handleClick = jest.fn();
  const user = userEvent.setup();

  render(<Button onClick={handleClick}>Click me</Button>);
  const button = screen.getByText('Click me');
```

```
    await user.click(button);

    expect(handleClick).toHaveBeenCalledTimes(1);
  });
});
```

Шаг 3. Создаём компонент формы входа (LoginForm)

Файл src/components/LoginForm.jsx:

```
import { useState } from 'react';

const LoginForm = ({ onLogin }) => {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');

  const handleSubmit = (e) => {
    e.preventDefault();

    if (!email || !password) {
      setError('Заполните все поля');
      return;
    }

    if (!email.includes('@')) {
      setError('Неверный формат email');
      return;
    }
```

```
  if (password.length < 6) {
    setError('Пароль должен содержать минимум 6 символов');
    return;
  }

  setError('');
  onLogin({ email, password });
};

return (
  <form onSubmit={handleSubmit} data-testid="login-form">
    <div>
      <label htmlFor="email">Email</label>
      <input
        id="email"
        type="email"
        value={email}
        onChange={(e) => setEmail(e.target.value)}
        placeholder="Введите email"
        data-testid="email-input"
      />
    </div>

    <div>
      <label htmlFor="password">Пароль</label>
      <input
        id="password"
        type="password"
```

```
          value={password}
          onChange={(e) => setPassword(e.target.value)}
          placeholder="Введите пароль"
          data-testid="password-input"
        />
      </div>

      {error && (
        <div style={{ color: 'red' }} data-testid="error-message">
          {error}
        </div>
      )}

      <button type="submit" data-testid="submit-button">
        Войти
      </button>
    </form>
  );
};

export default LoginForm;
```

Файл src/components/LoginForm.test.jsx (тесты формы):

```
import { render, screen, fireEvent } from '@testing-library/react';
import userEvent from '@testing-library/user-event';
import LoginForm from './LoginForm';

describe('LoginForm Component', () => {
```

```
// 1. Тест отрисовки всех полей
test('renders email and password inputs and submit button', () => {
  render(<LoginForm onLogin={() => {}} />);

  expect(screen.getByLabelText(/email/i)).toBeInTheDocument();
  expect(screen.getByLabelText(/пароль/i)).toBeInTheDocument();
  expect(screen.getByRole('button', { name: /войти/i })).toBeInTheDocument();
});

// 2. Тест отображения ошибки при пустых полях
test('shows error when email and password are empty', () => {
  render(<LoginForm onLogin={() => {}} />);

  const submitButton = screen.getByRole('button', { name: /войти/i });
  fireEvent.click(submitButton);

  expect(screen.getByTestId('error-message')).toHaveTextContent('Заполните все поля');
});

// 3. Тест валидации email
test('shows error when email is invalid', async () => {
  render(<LoginForm onLogin={() => {}} />);
  const user = userEvent.setup();

  await user.type(screen.getByTestId('email-input'), 'invalid-email');
  await user.type(screen.getByTestId('password-input'), 'password123');
  await user.click(screen.getByTestId('submit-button'));
```

```
    expect(screen.getByTestId('error-message')).toHaveTextContent('Неверный формат email');
  });

  // 4. Тест валидации пароля
  test('shows error when password is too short', async () => {
    render(<LoginForm onLogin={() => {}} />);
    const user = userEvent.setup();

    await user.type(screen.getByTestId('email-input'), 'test@example.com');
    await user.type(screen.getByTestId('password-input'), '123');
    await user.click(screen.getByTestId('submit-button'));

    expect(screen.getByTestId('error-message')).toHaveTextContent('Пароль должен содержать минимум 6 символов');
  });

  // 5. Тест успешной отправки
  test('calls onLogin with correct data when form is valid', async () => {
    const mockOnLogin = jest.fn();
    render(<LoginForm onLogin={mockOnLogin} />);
    const user = userEvent.setup();

    await user.type(screen.getByTestId('email-input'), 'test@example.com');
    await user.type(screen.getByTestId('password-input'), 'password123');
    await user.click(screen.getByTestId('submit-button'));

    expect(mockOnLogin).toHaveBeenCalledWith({
```

```
      email: 'test@example.com',
      password: 'password123',
    });
    expect(screen.queryByTestId('error-message')).not.toBeInTheDocument();
  });

  // 6. Тест обновления значений полей
  test('updates email input value when user types', async () => {
    render(<LoginForm onLogin={() => {}} />);
    const user = userEvent.setup();
    const emailInput = screen.getByTestId('email-input');

    await user.type(emailInput, 'test@example.com');

    expect(emailInput).toHaveValue('test@example.com');
  });
});
```

Шаг 4. Создаём компонент TodoList с API-запросами

Файл src/api/todoApi.js:

```
import axios from 'axios';

const API_URL = 'https://jsonplaceholder.typicode.com/todos';

export const fetchTodos = async () => {
  const response = await axios.get(`${API_URL}?_limit=5`);
  return response.data;
};
```

```
export const addTodo = async (title) => {
  const response = await axios.post(API_URL, {
    title,
    completed: false,
    userId: 1,
  });
  return response.data;
};

export const deleteTodo = async (id) => {
  await axios.delete(`${API_URL}/${id}`);
  return id;
};
```

Файл src/components/TodoList.jsx:

```
import { useState, useEffect } from 'react';
import { fetchTodos, addTodo, deleteTodo } from '../api/todoApi';

const TodoList = () => {
  const [todos, setTodos] = useState([]);
  const [newTodo, setNewTodo] = useState('');
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    loadTodos();
  }, []);
```

```
const loadTodos = async () => {
  try {
    setLoading(true);
    const data = await fetchTodos();
    setTodos(data);
    setError(null);
  } catch (err) {
    setError('Ошибка загрузки задач');
  } finally {
    setLoading(false);
  }
};

const handleAddTodo = async () => {
  if (!newTodo.trim()) return;

  try {
    const todo = await addTodo(newTodo);
    setTodos([...todos, todo]);
    setNewTodo('');
  } catch (err) {
    setError('Ошибка добавления задачи');
  }
};

const handleDeleteTodo = async (id) => {
  try {
    await deleteTodo(id);
```

```
    setTodos(todos.filter(todo => todo.id !== id));
  } catch (err) {
    setError('Ошибка удаления задачи');
  }
};

if (loading) return <div data-testid="loading">Загрузка...</div>;
if (error) return <div data-testid="error">{error}</div>;

return (
  <div data-testid="todo-list">
    <h2>Список задач</h2>

    <div>
      <input
        type="text"
        value={newTodo}
        onChange={(e) => setNewTodo(e.target.value)}
        placeholder="Новая задача"
        data-testid="todo-input"
      />
      <button onClick={handleAddTodo} data-testid="add-button">
        Добавить
      </button>
    </div>

    <ul>
      {todos.map(todo => (
        <li key={todo.id} data-testid={`todo-item-${todo.id}`}>
```

```
            <span>{todo.title}</span>
            <button
              onClick={() => handleDeleteTodo(todo.id)}
              data-testid={`delete-button-${todo.id}`}
            >
              Удалить
            </button>
          </li>
        ))}
      </ul>
    </div>
  );
};

export default TodoList;
```

Шаг 5. Настройка MSW для мокирования API

Файл src/mocks/handlers.js (определяем моковые обработчики):

```
import { http, HttpResponse } from 'msw';

// Моковые данные
const mockTodos = [
  { id: 1, title: 'Изучить React Testing Library', completed: false, userId: 1 },
  { id: 2, title: 'Написать тесты для компонентов', completed: false, userId: 1 },
  { id: 3, title: 'Настроить MSW', completed: false, userId: 1 },
];
```

```
let nextId = 4;

export const handlers = [
  // GET запрос - получение списка задач
  http.get('https://jsonplaceholder.typicode.com/todos', ({ request }) => {
    const url = new URL(request.url);
    const limit = url.searchParams.get('_limit');
    const todos = limit ? mockTodos.slice(0, parseInt(limit)) : mockTodos;

    return HttpResponse.json(todos);
  }),

  // POST запрос - добавление задачи
  http.post('https://jsonplaceholder.typicode.com/todos', async ({ request })
=> {
    const newTodo = await request.json();
    const todo = {
      id: nextId++,
      ...newTodo,
    };
    mockTodos.push(todo);
    return HttpResponse.json(todo, { status: 201 });
  }),

  // DELETE запрос - удаление задачи
  http.delete('https://jsonplaceholder.typicode.com/todos/:id', ({ params }) =>
{
    const id = parseInt(params.id);
    const index = mockTodos.findIndex(todo => todo.id === id);
```

```
    if (index !== -1) {
      mockTodos.splice(index, 1);
      return new HttpResponse(null, { status: 200 });
    }

    return new HttpResponse(null, { status: 404 });
  }),
];
```

Файл src/mocks/server.js (настройка сервера MSW):

```
import { setupServer } from 'msw/node';
import { handlers } from './handlers';

// Создаём сервер для перехвата запросов
export const server = setupServer(...handlers);
```

Файл src/setupTests.js (настройка тестового окружения):

```
import '@testing-library/jest-dom';
import { server } from './mocks/server';

// Запускаем сервер MSW перед всеми тестами
beforeAll(() => server.listen());

// Сбрасываем обработчики после каждого теста
afterEach(() => server.resetHandlers());
```

```
// Закрываем сервер после всех тестов
afterAll(() => server.close());
```

Файл vite.config.js (обновляем конфигурацию для Vitest):

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'

export default defineConfig({
  plugins: [react()],
  test: {
    globals: true,
    environment: 'jsdom',
    setupFiles: './src/setupTests.js',
  },
})
```

Шаг 6. Тесты TodoList с мокированием API

Файл src/components/TodoList.test.jsx:

```
import { render, screen, waitFor } from '@testing-library/react';
import userEvent from '@testing-library/user-event';
import TodoList from './TodoList';
import { server } from '../mocks/server';
import { http, HttpResponse } from 'msw';

describe('TodoList Component', () => {
  // 1. Тест загрузки и отображения задач
  test('fetches and displays todos', async () => {
    render(<TodoList />);
```

```
    // Проверяем, что сначала показывается загрузка
    expect(screen.getByTestId('loading')).toBeInTheDocument();

    // Ждём загрузки данных
    await waitFor(() => {
      expect(screen.getByText('Изучить React Testing Library')).toBeInTheDocument();
      expect(screen.getByText('Написать тесты для компонентов')).toBeInTheDocument();
    });

    // Проверяем, что загрузки больше нет
    expect(screen.queryByTestId('loading')).not.toBeInTheDocument();
  });

  // 2. Тест добавления новой задачи
  test('adds a new todo', async () => {
    const user = userEvent.setup();
    render(<TodoList />);

    // Ждём загрузки данных
    await waitFor(() => {
      expect(screen.getByText('Изучить React Testing Library')).toBeInTheDocument();
    });

    // Добавляем новую задачу
    const input = screen.getByTestId('todo-input');
```

```
    const addButton = screen.getByTestId('add-button');

    await user.type(input, 'Новая тестовая задача');
    await user.click(addButton);

    // Проверяем, что задача появилась в списке
    expect(screen.getByText('Новая тестовая задача')).toBeInTheDocument();
    expect(input).toHaveValue(''); // поле должно очиститься
  });

  // 3. Тест удаления задачи
  test('deletes a todo', async () => {
    const user = userEvent.setup();
    render(<TodoList />);

    // Ждём загрузки данных
    await waitFor(() => {
      expect(screen.getByText('Изучить React Testing Library')).toBeInTheDocument();
    });

    // Находим кнопку удаления для первой задачи
    const deleteButton = screen.getByTestId('delete-button-1');
    await user.click(deleteButton);

    // Проверяем, что задача удалена
    await waitFor(() => {
```

```
    expect(screen.queryByText('Изучить React Testing Library')).not.toBeInTheDocument();
  });
});

// 4. Тест ошибки при загрузке
test('displays error message when API fails', async () => {
  // Переопределяем обработчик для этого теста (возвращаем ошибку)
  server.use(
    http.get('https://jsonplaceholder.typicode.com/todos', () => {
      return new HttpResponse(null, { status: 500 });
    })
  );

  render(<TodoList />);

  await waitFor(() => {
    expect(screen.getByTestId('error')).toBeInTheDocument();
    expect(screen.getByTestId('error')).toHaveTextContent('Ошибка загрузки задач');
  });
});

// 5. Тест добавления пустой задачи (валидация)
test('does not add empty todo', async () => {
  const user = userEvent.setup();
  const initialTodosCount = 3;

  render(<TodoList />);
```

```
    await waitFor(() => {

expect(screen.getAllByRole('listitem')).toHaveLength(initialTodosCount);
    });

    // Пытаемся добавить пустую задачу
    const addButton = screen.getByTestId('add-button');
    await user.click(addButton);

    // Количество задач не изменилось
    expect(screen.getAllByRole('listitem')).toHaveLength(initialTodosCount);
  });
});
```

Шаг 7. Запуск тестов

```
# Запуск тестов один раз
npm test

# Запуск в режиме watch (автоматический перезапуск при изменениях)
npm test -- --watch

# Запуск с покрытием кода
npm test -- --coverage
```

Ожидаемый результат: Все тесты должны проходить успешно.

4. Полезные приёмы тестирования

4.1. Поиск элементов

```
// По тексту
screen.getByText('Привет');
screen.getByRole('button', { name: 'Отправить' });

// По метке (label)
screen.getByLabelText(/email/i);

// По placeholder
screen.getByPlaceholderText('Введите имя');

// По test-id
screen.getByTestId('submit-button');
```

4.2. Асинхронные запросы

```
// Ожидание появления элемента
await waitFor(() => {
  expect(screen.getByText('Загружено')).toBeInTheDocument();
});

// Ожидание исчезновения элемента
await waitFor(() => {
  expect(screen.queryByText('Загрузка...')).not.toBeInTheDocument();
});

// findBy* (комбинация getBy + waitFor)
const element = await screen.findByText('Загружено');
```

4.3. fireEvent vs userEvent

fireEvent	userEvent
Более низкоуровневый	Более высокоуровневый
Не имитирует focus/blur	Имитирует реальное взаимодействие
Быстрее	Медленнее, но реалистичнее
Рекомендуется для простых случаев	Рекомендуется для сложных взаимодействий

5. Индивидуальные задания

5.1. Базовый уровень (15 вариантов)

Требования: Написать unit-тесты для простого компонента: отрисовка, props, события.

Вариант	Компонент	Что тестировать
1	Counter (счётчик)	+, -, сброс
2	Toggle (переключатель)	Вкл/Выкл, текст
3	Alert (уведомление)	Рендер, закрытие
4	Accordion (аккордеон)	Открытие/закрытие
5	Tabs (вкладки)	Переключение вкладок

Вариант	Компонент	Что тестировать
6	Modal (модальное окно)	Открытие, закрытие
7	Dropdown (выпадающий список)	Выбор элемента
8	Slider (слайдер)	Изменение значения
9	Checkbox (чекбокс)	Изменение состояния
10	RadioGroup (радио-кнопки)	Выбор опции
11	Rating (рейтинг)	Клик по звёздам
12	SearchInput (поиск)	Ввод, очистка
13	Pagination (пагинация)	Переключение страниц
14	ProgressBar (прогресс)	Обновление значения
15	Tooltip (подсказка)	Появление при наведении

5.2. Средний уровень (15 вариантов)

Требования: Написать тесты для формы с валидацией и API-запросами (с мокированием).

Вариант	Тип формы	API запросы
1	Регистрация	POST /register
2	Вход	POST /login
3	Профиль	GET /profile, PUT /profile
4	Добавление товара	POST /products
5	Комментарии	POST /comments, GET /comments
6	Заказ	POST /orders
7	Отзыв	POST /reviews
8	Подписка	POST /subscribe
9	Восстановление пароля	POST /reset-password
10	Смена пароля	POST /change-password
11	Загрузка файла	POST /upload
12	Поиск	GET /search
13	Фильтрация	GET /filter
14	Пагинация	GET /items?page=1

Вариант	Тип формы	API запросы
15	Бесконечная прокрутка	GET /items?cursor=...

5.3. Продвинутый уровень (15 вариантов)

Требования: Написать комплексные тесты для полноценного приложения (список + форма + API).

Вариант	Приложение	Компоненты для тестирования
1	Todo-лист	TodoList, TodoForm, TodoItem, API
2	Контакты	ContactList, ContactForm, Search, API
3	Заметки	NoteList, NoteEditor, NoteCard, API
4	Корзина	Cart, ProductList, CartItem, API
5	Чат	MessageList, MessageInput, UserList, API
6	Календарь	Calendar, EventForm, EventList, API
7	Погода	WeatherCard, SearchCity, Forecast, API

Вариант	Приложение	Компоненты для тестирования
8	Курсы	CourseList, CourseCard, Enrollment, API
9	Блог	PostList, PostDetail, CommentForm, API
10	Форум	TopicList, TopicDetail, ReplyForm, API
11	Заказы	OrderList, OrderDetail, StatusUpdate, API
12	Сотрудники	EmployeeList, EmployeeForm, Department, API
13	Товары	ProductGrid, ProductCard, Filter, API
14	Библиотека	BookList, BookDetail, Search, API
15	Рецепты	RecipeList, RecipeDetail, IngredientList, API

6. Контрольные вопросы

1. Что такое unit-тестирование и зачем оно нужно?
2. Какие инструменты используются для тестирования React?
3. Как установить и настроить React Testing Library?
4. Как найти элемент в DOM для тестирования?

5. Что такое fireEvent и как его использовать?
6. В чём разница между getBy, queryBy и findBy?
7. Как тестировать асинхронные операции (API-запросы)?
8. Что такое мокирование и зачем оно нужно?
9. Как работает MSW (Mock Service Worker)?
10. Как протестировать форму с валидацией?
11. В чём разница между fireEvent и userEvent?
12. Как тестировать компоненты с хуками (useEffect, useState)?
13. Как протестировать обработку ошибок API?
14. Как измерить покрытие кода тестами?
15. Как организовать тесты для больших приложений?

7. Требования к отчёту

Титульный лист (название работы, ФИО, группа, вариант)

Цель работы

Теоретическое обоснование (своими словами)

Выполнение практического примера (код, скриншоты)

Выполнение индивидуального задания (код, скриншоты тестов)

Ответы на контрольные вопросы (не менее 5 по своему уровню)

Вывод

8. Критерии оценивания

Критерий	Базовый (max 35)	Средний (max 50)	Продвинутый (max 65)
Настройка тестового окружения	5	5	5
Тестирование	10	5	5

Критерий	Базовый (max 35)	Средний (max 50)	Продвинутый (max 65)
отрисовки компонента			
Тестирование событий (клики, ввод)	15	10	5
Тестирование формы с валидацией	—	15	10
Мокирование API (MSW)	—	15	15
Тестирование загрузки и ошибок	5	—	10
Комплексное тестирование приложения	—	—	15
Итого	**35**	**50**	**65**

Шкала перевода баллов в оценку:

Оценка	Базовый (35)	Средний (50)	Продвинутый (65)
5 (отлично)	32-35	46-50	59-65
4 (хорошо)	28-31	40-45	51-58

Оценка	Базовый (35)	Средний (50)	Продвинутый (65)
3 (удовлетворительно)	22-27	32-39	42-50
2 (неудовлетворительно)	0-21	0-31	0-41

Лабораторная работа №23

Оптимизация производительности React

1. Цель работы

Освоить методы оптимизации производительности React-приложений. Научиться использовать React.memo для предотвращения лишних перерисовок, хуки useMemo и useCallback для мемоизации значений и функций, реализовывать ленивую загрузку компонентов с помощью React.lazy и Suspense, выполнять код-сплиттинг, а также профилировать приложение для выявления проблем производительности.

2. Теоретическое обоснование

2.1. Почему React-приложения могут работать медленно?

Проблема	Описание	Пример
Лишние перерисовки	Компонент перерисовывается, хотя его props не изменились	Родительский компонент перерисовывается → все дети

Проблема	Описание	Пример
		перерисовываются
Тяжёлые вычисления	Вычисления выполняются при каждой перерисовке	Фильтрация большого списка, сложные математические расчёты
Большой бандл	Все компоненты загружаются при первом открытии	Пользователь ждёт загрузки страницы, которой может не пользоваться
Неоптимизированные списки	Каждый элемент списка перерисовывается при изменении одного	При добавлении нового элемента весь список перерисовывается

2.2. Когда React перерисовывает компонент?

Компонент перерисовывается, когда:

1. Изменилось его состояние (useState, useReducer)

2. Изменились его props (родитель передал новые значения)

3. Родительский компонент перерисовался (даже если props не изменились!)

2.3. Инструменты оптимизации

Инструмент	Что делает	Когда использовать
React.memo	Запоминает результат рендера компонента, перерисовывает только при изменении props	Компонент перерисовывается часто с одними и теми же props
useMemo	Запоминает результат вычисления	Тяжёлые вычисления, которые не нужно повторять при каждой перерисовке
useCallback	Запоминает функцию	Функция передаётся в дочерний компонент, обёрнутый в React.memo
React.lazy + Suspense	Ленивая загрузка компонентов	Большие компоненты, которые не нужны сразу при загрузке страницы
Profiler	Измеряет производительность	Поиск узких мест в приложении

3. Практический пример

Что мы будем делать: Создадим приложение, демонстрирующее проблемы производительности и способы их решения.

Шаг 1. Создание проекта

```
# Создаём проект
npm create vite@latest perf-demo -- --template react
cd perf-demo

# Устанавливаем зависимости
npm install

# Устанавливаем дополнительные утилиты (для генерации данных)
npm install @faker-js/faker

# Запускаем проект
npm run dev
```

Шаг 2. Создаём компонент для демонстрации проблем

Файл src/components/ExpensiveComponent.jsx (компонент с тяжёлыми вычислениями):

```
import { useState, useMemo, useCallback } from 'react';
import { faker } from '@faker-js/faker';

// Генерация большого списка данных (1000 элементов)
const generateData = () => {
  console.log('🔄 Генерация данных...');
  return Array.from({ length: 1000 }, (_, i) => ({
    id: i,
    name: faker.person.fullName(),
    email: faker.internet.email(),
    city: faker.location.city(),
    age: Math.floor(Math.random() * 50) + 18,
```

```
  }));
};

// Тяжёлая функция фильтрации
const filterData = (data, searchTerm) => {
  console.log('🔍 Выполняется фильтрация...');
  // Имитация тяжёлых вычислений
  const start = performance.now();
  while (performance.now() - start < 5) { /* задержка */ }

  if (!searchTerm) return data;

  return data.filter(item =>
    item.name.toLowerCase().includes(searchTerm.toLowerCase()) ||
    item.email.toLowerCase().includes(searchTerm.toLowerCase())
  );
};

// Компонент строки таблицы (обычный)
const TableRow = ({ item }) => {
  console.log(`📄 Рендер строки ${item.id}`);
  return (
    <tr>
      <td>{item.id}</td>
      <td>{item.name}</td>
      <td>{item.email}</td>
      <td>{item.city}</td>
      <td>{item.age}</td>
    </tr>
```

```
  );
};

// Компонент строки таблицы с React.memo (мемоизированный)
const MemoizedTableRow = React.memo(({ item }) => {
  console.log(`✨ Рендер строки ${item.id} (мемоизированная)`);
  return (
    <tr>
      <td>{item.id}</td>
      <td>{item.name}</td>
      <td>{item.email}</td>
      <td>{item.city}</td>
      <td>{item.age}</td>
    </tr>
  );
});

const ExpensiveComponent = () => {
  const [data] = useState(() => generateData());
  const [searchTerm, setSearchTerm] = useState('');
  const [counter, setCounter] = useState(0);
  const [useMemoization, setUseMemoization] = useState(true);

  // Без useMemo — фильтрация будет выполняться при КАЖДОЙ перерисовке
  const filteredDataNoMemo = filterData(data, searchTerm);

  // С useMemo — фильтрация выполняется только при изменении data или searchTerm
```

```
  const filteredDataWithMemo = useMemo(() => {
    return filterData(data, searchTerm);
  }, [data, searchTerm]);

  const filteredData = useMemoization ? filteredDataWithMemo : filteredDataNoMemo;

  // Обработчик инкремента (без useCallback)
  const handleIncrementNoCallback = () => {
    setCounter(c => c + 1);
  };

  // Обработчик инкремента (с useCallback)
  const handleIncrementWithCallback = useCallback(() => {
    setCounter(c => c + 1);
  }, []);

  const handleIncrement = useMemoization ? handleIncrementWithCallback : handleIncrementNoCallback;

  const RowComponent = useMemoization ? MemoizedTableRow : TableRow;

  return (
    <div>
      <h2>Демонстрация производительности</h2>

      <div style={{ marginBottom: '20px', padding: '10px', background: '#f0f0f0', borderRadius: '8px' }}>
```

```
      <h3>Настройки</h3>
      <label>
        <input
          type="checkbox"
          checked={useMemoization}
          onChange={(e) => setUseMemoization(e.target.checked)}
        />
        Использовать оптимизации (useMemo, React.memo, useCallback)
      </label>
    </div>

    <div style={{ marginBottom: '20px' }}>
      <h3>Счётчик (для вызова перерисовки)</h3>
      <p>Значение: {counter}</p>
      <button onClick={handleIncrement}>
        Увеличить счётчик (вызовет перерисовку)
      </button>
    </div>

    <div style={{ marginBottom: '20px' }}>
      <h3>Поиск</h3>
      <input
        type="text"
        placeholder="Поиск по имени или email..."
        value={searchTerm}
        onChange={(e) => setSearchTerm(e.target.value)}
        style={{ width: '300px', padding: '8px' }}
      />
    </div>
```

```
            <div>
              <h3>Результатов: {filteredData.length}</h3>
              <div style={{ maxHeight: '400px', overflow: 'auto', border: '1px solid #ccc' }}>
                <table border="1" cellPadding="8" cellSpacing="0" style={{ width: '100%' }}>
                  <thead style={{ position: 'sticky', top: 0, background: '#fff' }}>
                    <tr>
                      <th>ID</th>
                      <th>Имя</th>
                      <th>Email</th>
                      <th>Город</th>
                      <th>Возраст</th>
                    </tr>
                  </thead>
                  <tbody>
                    {filteredData.map(item => (
                      <RowComponent key={item.id} item={item} />
                    ))}
                  </tbody>
                </table>
              </div>
            </div>

            <div style={{ marginTop: '20px', padding: '10px', background: '#f8f9fa', borderRadius: '8px' }}>
              <h3>Инструкция</h3>
              <ol>
```

```
            <li>Откройте консоль браузера (F12)</li>
            <li>Снимите галочку "Использовать оптимизации"</li>
            <li>Несколько раз нажмите "Увеличить счётчик" — увидите много логов в консоли</li>
            <li>Поставьте галочку обратно и снова нажмите на счётчик — логов станет меньше</li>
            <li>Введите текст в поле поиска и сравните производительность</li>
          </ol>
        </div>
      </div>
    );
  };

  export default ExpensiveComponent;
```

Шаг 3. Создаём большой компонент для демонстрации lazy loading

Файл src/components/HeavyComponent.jsx:

```
  // Это "тяжёлый" компонент, который загружается только по требованию
  const HeavyComponent = () => {
    // Имитация тяжёлых вычислений при загрузке
    const startTime = performance.now();
    const heavyData = Array.from({ length: 10000 }, (_, i) => ({
      id: i,
      value: Math.random() * 1000,
    }));
    const endTime = performance.now();
```

```
    return (
      <div style={{ padding: '20px', border: '1px solid #ccc', borderRadius: '8px', marginTop: '20px' }}>
        <h3>📦 Тяжёлый компонент (загружен лениво)</h3>
        <p>Время генерации данных: {(endTime - startTime).toFixed(2)} мс</p>
        <p>Сгенерировано элементов: {heavyData.length}</p>
        <details>
          <summary>Показать пример данных</summary>
          <pre>{JSON.stringify(heavyData.slice(0, 5), null, 2)}</pre>
        </details>
      </div>
    );
  };

export default HeavyComponent;
```

Шаг 4. Создаём главное приложение с lazy loading

Файл src/App.jsx:

```
import { useState, Suspense, lazy, useCallback, useEffect } from 'react';
import ExpensiveComponent from './components/ExpensiveComponent';

// Ленивая загрузка тяжёлого компонента
const HeavyComponent = lazy(() => import('./components/HeavyComponent'));

// Ленивая загрузка с индикатором загрузки
```

```
const Dashboard = lazy(() =>
  import('./components/Dashboard').then(module => {
    // Имитация задержки загрузки
    return new Promise(resolve => {
      setTimeout(() => resolve(module), 1000);
    });
  })
);

function App() {
  const [showHeavy, setShowHeavy] = useState(false);
  const [showDashboard, setShowDashboard] = useState(false);
  const [logs, setLogs] = useState([]);

  const addLog = useCallback((message) => {
    setLogs(prev => [...prev, { time: new Date().toLocaleTimeString(), message }]);
  }, []);

  useEffect(() => {
    addLog('Приложение загружено');
  }, [addLog]);

  const handleLoadHeavy = () => {
    addLog('Загрузка тяжёлого компонента...');
    setShowHeavy(true);
  };

  const handleLoadDashboard = () => {
```

```
    addLog('Загрузка дашборда...');
    setShowDashboard(true);
  };

  return (
    <div style={{ maxWidth: '1200px', margin: '0 auto', padding: '20px' }}>
      <h1>Оптимизация производительности React</h1>

      {/* Секция 1: React.memo, useMemo, useCallback */}
      <section style={{ marginBottom: '40px', border: '1px solid #ddd',
padding: '20px', borderRadius: '8px' }}>
        <h2>1. React.memo, useMemo, useCallback</h2>
        <ExpensiveComponent />
      </section>

      {/* Секция 2: React.lazy и Suspense */}
      <section style={{ marginBottom: '40px', border: '1px solid #ddd',
padding: '20px', borderRadius: '8px' }}>
        <h2>2. Ленивая загрузка (React.lazy + Suspense)</h2>

        <div style={{ display: 'flex', gap: '10px', marginBottom: '20px' }}>
          <button onClick={handleLoadHeavy} disabled={showHeavy}>
            {showHeavy ? '✓ Загружен' : '📦 Загрузить тяжёлый компонент
(300KB)'}
          </button>
          <button                                onClick={handleLoadDashboard}
disabled={showDashboard}>
            {showDashboard ? '✓ Загружен' : '📊 Загрузить дашборд (с
задержкой)'}
```

```
          </button>
        </div>

        <Suspense fallback={<div style={{ padding: '20px', textAlign: 'center' }}>⏳ Загрузка компонента...</div>}>
          {showHeavy && <HeavyComponent />}
        </Suspense>

        <Suspense fallback={<div style={{ padding: '20px', textAlign: 'center' }}>⏳ Загрузка дашборда...</div>}>
          {showDashboard && <Dashboard />}
        </Suspense>
      </section>

      {/* Секция 3: Логи */}
      <section style={{ border: '1px solid #ddd', padding: '20px', borderRadius: '8px' }}>
        <h2>3. Логи загрузки</h2>
        <div style={{ maxHeight: '200px', overflow: 'auto', background: '#f5f5f5', padding: '10px', fontFamily: 'monospace' }}>
          {logs.map((log, index) => (
            <div key={index}>[{log.time}] {log.message}</div>
          ))}
        </div>
        <button onClick={() => setLogs([])} style={{ marginTop: '10px' }}>Очистить логи</button>
      </section>

      {/* Информационный блок */}
```

```
        <div style={{ marginTop: '40px', padding: '20px', background: '#e3f2fd', borderRadius: '8px' }}>
          <h3>📚 Что мы демонстрируем:</h3>
          <ul>
            <li><strong>React.memo</strong> — предотвращает перерисовку дочерних компонентов, если props не изменились</li>
            <li><strong>useMemo</strong> — кэширует результат тяжёлых вычислений</li>
            <li><strong>useCallback</strong> — кэширует функции, чтобы они не создавались заново</li>
            <li><strong>React.lazy + Suspense</strong> — загружает компоненты только когда они нужны</li>
          </ul>
          <p>Откройте <strong>консоль браузера (F12)</strong> и сравните количество логов при включённых/выключенных оптимизациях.</p>
        </div>
      </div>
    );
  }

  export default App;
```

Шаг 5. Создаём компонент дашборда для демонстрации

Файл src/components/Dashboard.jsx:

```
import { useState, useEffect } from 'react';
import { faker } from '@faker-js/faker';

// Генерация статистики
const generateStats = () => ({
  users: faker.number.int({ min: 1000, max: 10000 }),
```

```
    posts: faker.number.int({ min: 500, max: 5000 }),
    comments: faker.number.int({ min: 2000, max: 20000 }),
    revenue: faker.number.int({ min: 10000, max: 100000 }),
  });

  // Компонент карточки статистики (с React.memo)
  const StatCard = React.memo(({ title, value, icon, color }) => {
    console.log(`📊 Рендер карточки ${title}`);
    return (
      <div style={{
        padding: '20px',
        background: color,
        color: 'white',
        borderRadius: '8px',
        textAlign: 'center',
        minWidth: '150px',
      }}>
        <div style={{ fontSize: '32px' }}>{icon}</div>
        <div style={{ fontSize: '24px', fontWeight: 'bold' }}>{value.toLocaleString()}</div>
        <div>{title}</div>
      </div>
    );
  });

  const Dashboard = () => {
    const [stats, setStats] = useState(generateStats);
    const [refreshCounter, setRefreshCounter] = useState(0);
```

```
// Обновление статистики (имитация получения новых данных)
const refreshStats = () => {
  console.log('🔄 Обновление статистики...');
  setStats(generateStats());
  setRefreshCounter(prev => prev + 1);
};

useEffect(() => {
  console.log('📊 Дашборд загружен и отрендерен');
}, []);

return (
  <div style={{ marginTop: '20px', padding: '20px', border: '1px solid #ddd', borderRadius: '8px' }}>
    <h3>📊 Дашборд</h3>
    <p>Обновлений: {refreshCounter}</p>

    <div style={{ display: 'flex', gap: '20px', flexWrap: 'wrap', marginBottom: '20px' }}>
      <StatCard title="Пользователи" value={stats.users} icon="👥" color="#4caf50" />
      <StatCard title="Посты" value={stats.posts} icon="📝" color="#2196f3" />
      <StatCard title="Комментарии" value={stats.comments} icon="💬" color="#ff9800" />
      <StatCard title="Выручка" value={stats.revenue} icon="💰" color="#9c27b0" />
    </div>
```

```
        <button onClick={refreshStats}>
          🔄 Обновить статистику
        </button>

        <div style={{ marginTop: '15px', fontSize: '12px', color: '#666' }}>
          <p>Каждая карточка обёрнута в React.memo — при обновлении статистики перерисовываются только карточки с изменившимися значениями.</p>
          <p>Смотрите консоль для отслеживания рендеров.</p>
        </div>
      </div>
    );
  };

  export default Dashboard;
```

Шаг 6. Конфигурация для профилирования

Файл src/components/ProfilerExample.jsx (опционально):

```
import { Profiler } from 'react';

// Функция обратного вызова для Profiler
const onRenderCallback = (
  id,           // ID компонента (из props)
  phase,        // "mount" (монтирование) или "update" (обновление)
  actualDuration,      // время рендера
  baseDuration,        // оценочное время без мемоизации
  startTime,           // время начала рендера
  commitTime,          // время коммита
```

```
    interactions        // взаимодействия
  ) => {
    console.log(`[Profiler] ${id} (${phase}) - ${actualDuration.toFixed(2)}ms`);

    // Отправка метрик в аналитику
    if (actualDuration > 10) {
      console.warn(`⚠ Медленный рендер: ${id} - ${actualDuration.toFixed(2)}ms`);
    }
  };

  const SlowComponent = () => {
    // Имитация медленного рендера
    const start = performance.now();
    while (performance.now() - start < 20) { /* задержка */ }

    return <div>Медленный компонент</div>;
  };

  const ProfilerExample = () => {
    const [count, setCount] = useState(0);

    return (
      <Profiler id="App" onRender={onRenderCallback}>
        <div>
          <h3>Профилирование производительности</h3>
          <button onClick={() => setCount(c => c + 1)}>
            Перерендерить ({count})
```

```
        </button>

        <Profiler id="SlowComponent" onRender={onRenderCallback}>
          <SlowComponent />
        </Profiler>
      </div>
    </Profiler>
  );
};

export default ProfilerExample;
```

4. Сравнение оптимизаций

Метод	Что делает	Когда использовать	Побочные эффекты
React.memo	Кэширует результат рендера компонента	Компонент часто перерисовывается с одинаковыми props	Увеличивает потребление памяти
useMemo	Кэширует результат вычисления	Тяжёлые вычисления при каждом рендере	Усложняет код
useCallback	Кэширует функцию	Функция передаётся в мемоизированный компонент	Усложняет код

Метод	Что делает	Когда использовать	Побочные эффекты
React.lazy	Ленивая загрузка компонента	Большие компоненты, не нужные сразу	Задержка при первом рендере
Code splitting	Разделение бандла	Разные страницы/модули	Усложнение сборки

5. Индивидуальные задания

5.1. Базовый уровень (15 вариантов)

Требования: Написать компонент, демонстрирующий проблему лишних перерисовок, и решить её с помощью React.memo, useMemo, useCallback.

Вариант	Тип приложения	Что оптимизировать
1	Список товаров	Фильтрация и сортировка списка
2	Корзина	Подсчёт общей суммы
3	Календарь	Рендер ячеек календаря
4	Чат	Список сообщений
5	Таблица данных	Фильтрация и пагинация

Вариант	Тип приложения	Что оптимизировать
6	Древовидное меню	Раскрытие узлов
7	График	Перерисовка при изменении данных
8	Форма с валидацией	Валидация полей
9	Поиск с подсказками	Фильтрация подсказок
10	Слайдер	Обновление значения
11	Аккордеон	Раскрытие панелей
12	Вкладки (Tabs)	Переключение контента
13	Модальное окно	Открытие/закрытие
14	D r a g - a n d - drop	Перемещение элементов
15	Анимация	Кадры анимации

5.2. Средний уровень (15 вариантов)

Требования: Реализовать ленивую загрузку (React.lazy, Suspense) для крупных компонентов или целых страниц.

Вариант	Тип приложения	Компоненты для ленивой загрузки
1	Интернет-магазин	Каталог, Корзина, Профиль
2	Блог	Список постов, Пост, Комментарии
3	Админ-панель	Статистика, Пользователи, Настройки
4	Чат	Список чатов, История сообщений
5	Канбан-доска	Доска, Настройки, История
6	CRM	Клиенты, Сделки, Отчёты
7	LMS	Курсы, Тесты, Сертификаты
8	Финансовый трекер	Транзакции, Графики, Отчёты
9	Социальная сеть	Лента, Профиль, Сообщения

Вариант	Тип приложения	Компоненты для ленивой загрузки
10	Система тикетов	Тикеты, Статистика, Настройки
11	Медицинская система	Пациенты, Записи, Результаты
12	HR-система	Сотрудники, Вакансии, Отчёты
13	Недвижимость	Объекты, Карта, Ипотека
14	Форум	Разделы, Темы, Сообщения
15	Агрегатор новостей	Новости, Источники, Аналитика

5.3. Продвинутый уровень (15 вариантов)

Требования: Провести профилирование приложения, найти узкие места и оптимизировать их. Предоставить отчёт с метриками до и после оптимизации.

Вариант	Тип приложения	Метрики для анализа
1	E-commerce	Время загрузки страницы, FCP, LCP

Вариант	Тип приложения	Метрики для анализа
2	Социальная сеть	TTI, время интерактивности
3	Дашборд	Частота перерисовок, размер бандла
4	Онлайн-редактор	Время отклика на ввод
5	Карта	Время рендера маркеров
6	Таблица с 10000 строк	Скроллинг, фильтрация
7	Видеоплеер	Буферизация, перемотка
8	Игра	FPS, задержки ввода
9	Конструктор сайтов	Время превью
10	Система аналитики	Обработка больших данных
11	OCR-приложение	Время распознавания
12	Чат с файлами	Загрузка/отправка
13	Видеоконференция	Задержки, качество
14	Крипто-трейдинг	Обновление графиков

Вариант	Тип приложения	Метрики для анализа
15	Медицинская диагностика	Обработка изображений

6. Контрольные вопросы

1. Почему React-приложения могут работать медленно?
2. Что делает React.memo и когда его использовать?
3. В чём разница между useMemo и useCallback?
4. Как работает React.lazy?
5. Для чего нужен компонент Suspense?
6. Как проверить, что компонент перерисовывается лишний раз?
7. В каких случаях React.memo не помогает?
8. Какие проблемы могут возникнуть при использовании useCallback?
9. Как настроить код-сплиттинг в Vite?
10. Что такое "bundle analysis" и как его провести?
11. Как работает useTransition и для чего он нужен?
12. Что такое "Concurrent Mode" и "Concurrent Features"?
13. Как использовать useDeferredValue для оптимизации?
14. Как работает Virtual DOM и как его оптимизировать?
15. Как проводить профилирование с React DevTools?

7. Требования к отчёту

Титульный лист (название работы, ФИО, группа, вариант)

Цель работы

Теоретическое обоснование (своими словами)

Выполнение практического примера (код, скриншоты консоли с логами)

Выполнение индивидуального задания (код, метрики до/после)

Ответы на контрольные вопросы (не менее 5 по своему уровню)

Вывод

8. Критерии оценивания

Критерий	Базовый (max 35)	Средний (max 50)	Продвинутый (max 65)
Демонстрация проблемы перерисовок	10	5	5
Использование React.memo	10	10	5
Использование useMemo	5	10	5
Использование useCallback	5	10	5
Ленивая загрузка (React.lazy)	—	15	10
Код-сплиттинг	—	—	15
Профилирование и анализ	—	—	15

Критерий	Базовый (max 35)	Средний (max 50)	Продвинутый (max 65)
Качество кода	5	—	5
Итого	35	50	65

Шкала перевода баллов в оценку:

Оценка	Базовый (35)	Средний (50)	Продвинутый (65)
5 (отлично)	32-35	46-50	59-65
4 (хорошо)	28-31	40-45	51-58
3 (удовлетворительно)	22-27	32-39	42-50
2 (неудовлетворительно)	0-21	0-31	0-41

Лабораторная работа №24

Сборка и деплой статического фронтенда

1. Цель работы

Научиться собирать React-приложение для production, анализировать размер бандла, развертывать статический фронтенд на хостингах Vercel и Netlify, а также настраивать перенаправления для корректной работы SPA (Single Page Application) при обновлении страницы.

2. Теоретическое обоснование

2.1. Что такое сборка (build) React-приложения?

Сборка - это процесс преобразования исходного кода React-приложения (JSX, модули, компоненты) в оптимизированный набор статических файлов (HTML, CSS, JavaScript), которые можно разместить на веб-сервере.

Что происходит во время сборки:

Действие	Описание
Транспиляция	Преобразование JSX в обычный JavaScript
Минификация	Удаление пробелов, комментариев, сокращение имён переменных
Сжатие	Уменьшение размера файлов (gzip, brotli)
Код-сплиттинг	Разделение кода на отдельные чанки (chunks)
Tree shaking	Удаление неиспользуемого кода
Оптимизация изображений	Сжатие и оптимизация картинок

2.2. Зачем нужна сборка?

Без сборки	Со сборкой
Много маленьких файлов	Несколько оптимизированных файлов

Без сборки	Со сборкой
Код читаемый, но большой	Код минифицированный, маленький
Медленная загрузка	Быстрая загрузка
Не оптимизирован для production	Оптимизирован для production

2.3. Структура production-сборки (Vite)

```
dist/
├── index.html          # Главная HTML-страница
├── assets/
│   ├── index-abc123.js # Главный JS-бандл (с хешем в имени)
│   ├── index-def456.css# Главный CSS-файл
│   └── vendor-ghi789.js# Бандл сторонних библиотек (если настроен код-сплиттинг)
└── vite.svg            # Статические файлы
```

2.4. Проблема SPA и роутинга

Проблема: В SPA маршрутизация работает на стороне клиента. При обновлении страницы (F5) браузер отправляет запрос к серверу по текущему URL, но сервер не знает о клиентском роутинге и возвращает 404.

Решение: Настроить сервер так, чтобы для всех неизвестных маршрутов отдавался index.html (это называется "catch-all" или "fallback").

Запрос: /about

↓

Сервер ищет файл about.html → не находит

↓

Сервер отдаёт index.html

↓

React Router обрабатывает /about и показывает страницу About

3. Практический пример

Шаг 1. Подготовка React-приложения

Для демонстрации возьмём приложение из предыдущих лабораторных работ (например, менеджер задач из ЛР №21). Убедитесь, что оно работает локально.

```
# Перейдите в папку с вашим React-проектом
cd my-app

# Запустите режим разработки для проверки
npm run dev
# Приложение должно работать на http://localhost:5173
```

Шаг 2. Сборка проекта

```
# Выполните production-сборку
npm run build
```

Результат: В папке dist появится оптимизированная сборка.

```
dist/
├── index.html
├── assets/
```

```
│   ├── index-abc123.js
│   ├── index-def456.css
│   └── ...
└── vite.svg
```

Шаг 3. Анализ размера бандла

3.1. Простой анализ (размер файлов)

```
# Просмотр размера файлов в папке dist
ls -lh dist/
du -sh dist/
```

3.2. Визуальный анализ бандла

Установите плагин для анализа бандла:

bash

```
npm install --save-dev rollup-plugin-visualizer
```

Файл vite.config.js:

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'
import { visualizer } from 'rollup-plugin-visualizer'

export default defineConfig({
  plugins: [
    react(),
    visualizer({
      filename: 'dist/stats.html',  // куда сохранить отчёт
      open: true,                   // автоматически открыть после сборки
      gzipSize: true,               // показать размер после gzip
      brotliSize: true,             // показать размер после brotli
```

```
    }),
  ],
})
```

После сборки откроется интерактивный отчёт с визуализацией размера каждого модуля.

3.3. Анализ с помощью source-map-explorer

```
npm install --save-dev source-map-explorer
```

Добавьте скрипт в package.json:

```
{
  "scripts": {
    "analyze": "source-map-explorer 'dist/assets/*.js'"
  }
}
```

Запустите анализ:

```
npm run build
npm run analyze
```

Шаг 4. Локальная проверка сборки

Перед деплоем важно проверить, что сборка работает корректно.

bash

```
# Установите глобально serve
npm install -g serve

# Запустите локальный сервер для проверки сборки
serve -s dist -p 3000
```

Откройте http://localhost:3000 и проверьте:

Главная страница загружается

Навигация работает

Все страницы доступны

При обновлении нестандартного маршрута (например, /about) возникает ошибка 404 - это нормально для локального сервера, на хостинге мы это исправим.

Шаг 5. Деплой на Vercel

5.1. Подготовка проекта для Vercel

Vercel автоматически определяет React-проекты, но для SPA нужно настроить перенаправления.

```
{
 "rewrites": [
   {
    "source": "/(.*)",
    "destination": "/index.html"
   }
 ],
 "buildCommand": "npm run build",
 "outputDirectory": "dist",
 "devCommand": "npm run dev"
}
```

5.2. Деплой через CLI

```
# Установка Vercel CLI
npm install -g vercel
```

```
# Вход в аккаунт
vercel login

# Деплой (в корне проекта)
vercel

# Следуйте инструкциям:
# - Set up and deploy: Y
# - Scope: выберите аккаунт
# - Link to existing project: N
# - Project name: my-react-app
# - Directory: ./
# - Override settings: N

# Для production-деплоя:
vercel --prod
```

5.3. Деплой через Git (GitHub + Vercel)

Загрузите проект на GitHub:

```
git init
git add .
git commit -m "Initial commit"
git remote add origin https://github.com/ваш-username/название-репозитория.git
git push -u origin main
```

Зайдите на vercel.com

Нажмите "Add New" → "Project"

Импортируйте репозиторий из GitHub

Настройки обычно определяются автоматически

Нажмите "Deploy"

5.4. Проверка на Vercel

После деплоя вы получите URL вида: https://my-react-app.vercel.app

Проверьте:

Открытие главной страницы

Переход по ссылкам

Важно: Обновление страницы на нестандартном маршруте (например, /profile) - должно работать благодаря настройке rewrites в vercel.json.

Шаг 6. Деплой на Netlify

6.1. Подготовка проекта для Netlify

Файл netlify.toml (в корне проекта):

```
[build]
  command = "npm run build"
  publish = "dist"

[[redirects]]
  from = "/*"
  to = "/index.html"
  status = 200
```

6.2. Деплой через CLI

```
# Установка Netlify CLI
npm install -g netlify-cli

# Вход в аккаунт
netlify login
```

```
# Деплой
netlify deploy

# Выберите опции:
# - Create & configure a new site
# - Team: выберите команду
# - Site name: my-react-app

# Для production-деплоя:
netlify deploy --prod
```

6.3. Деплой через Drag-and-Drop

Соберите проект: npm run build

Зайдите на app.netlify.com

Перетащите папку dist в область "Drag and drop your site folder here"

Сайт будет опубликован

6.4. Настройка перенаправлений в Netlify UI

Если вы не используете netlify.toml:

Зайдите в настройки сайта на Netlify

Перейдите в "Redirects & Rewrites"

Добавьте правило:

From: /*

To: /index.html

Status: 200

Шаг 7. Деплой на GitHub Pages

7.1. Установка gh-pages

```
npm install --save-dev gh-pages
```

7.2. Настройка package.json

```
{
  "homepage": "https://ваш-username.github.io/название-репозитория",
  "scripts": {
    "predeploy": "npm run build",
    "deploy": "gh-pages -d dist"
  }
}
```

7.3. Настройка Vite для GitHub Pages

Файл vite.config.js:

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'

export default defineConfig({
  plugins: [react()],
  base: '/название-репозитория/',  // важно для GitHub Pages!
})
```

7.4. Деплой

```
npm run deploy
```

Сайт будет доступен по адресу: https://ваш-username.github.io/название-репозитория

Примечание для GitHub Pages: Необходимо настроить SPA-роутинг через 404.html. Создайте файл public/404.html с содержимым, аналогичным index.html, но с добавлением скрипта для перенаправления:

```
<!DOCTYPE html>
<html>
<head>
```

```
  <meta charset="utf-8">
  <title>My React App</title>
  <script>
    sessionStorage redirect = location href;
    location replace('/название-репозитория/');
  </script>
</head>
<body>
</body>
</html>
```

Шаг 8. Настройка CI/CD для автоматического деплоя (GitHub Actions)

Файл .github/workflows/deploy.yml:

```
name: Deploy to GitHub Pages

on:
  push:
    branches: [ main ]
  workflow_dispatch:

permissions:
  contents: read
  pages: write
  id-token: write

jobs:
  build:
    runs-on: ubuntu latest
    steps:
```

```
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: 20
          cache: 'npm'
      - run: npm ci
      - run: npm run build
      - uses: actions/upload-pages-artifact@v3
        with:
          path: ./dist

  deploy:
    environment:
      name: github-pages
      url: ${{ steps.deployment.outputs.page_url }}
    runs-on: ubuntu-latest
    needs: build
    steps:
      - uses: actions/deploy-pages@v4
```

4. Сравнение хостингов для статического фронтенда

Хостинг	Бесплатный тариф	Особенности	Настройка SPA	CI/CD
Vercel	Да	Простая настройка, автоматически	vercel.json (rewrites)	Git push

Хостинг	Бесплатный тариф	Особенности	Настройка SPA	CI/CD
		й HTTPS		
Netlify	Да	Drag-and-drop, формы	netlify.toml или UI	Git push
GitHub Pages	Да	Интеграция с GitHub	404.html или HashRouter	GitHub Actions
Cloudflare Pages	Да	Безлимитный трафик	_redirects файл	Git push
Render	Да	Простой интерфейс	redirects	Git push
Surge	Да	CLI-only	200.html	CLI
Firebase Hosting	Да	Интеграция с Firebase	firebase.json	CLI/Git

5. Индивидуальные задания

5.1. Базовый уровень (15 вариантов)

Требования: Собрать React-приложение, проанализировать размер бандла, развернуть на Vercel или Netlify.

Вариант	Дополнительное требование
1	Оптимизировать размер бандла (удалить неиспользуемые зависимости)
2	Настроить сжатие изображений при сборке
3	Добавить мета-теги для SEO
4	Настроить кастомный домен
5	Включить gzip/brotli сжатие
6	Настроить кэширование статики
7	Добавить favicon
8	Настроить Open Graph мета-теги
9	Добавить аналитику (Google или Yandex)
10	Настроить PWA (Service Worker)
11	Добавить Sitemap.xml
12	Настроить robots.txt
13	Добавить политику CSP (Content Security Policy)
14	Настроить мониторинг производительности
15	Добавить поддержку офлайн-режима

5.2. Средний уровень (15 вариантов)

Требования: Настроить автоматический деплой через GitHub Actions, подключить кастомный домен, настроить перенаправления.

Вариант	Дополнительное требование
1	Настроить деплой на два окружения (staging/production)
2	Добавить проверку линтера перед деплоем
3	Добавить запуск тестов перед деплоем
4	Настроить уведомления в Telegram о деплое
5	Добавить генерацию bundle-отчёта в артефакты
6	Настроить инвалидацию кэша CloudFlare после деплоя
7	Добавить проверку битых ссылок после деплоя
8	Настроить деплой на несколько хостингов одновременно
9	Добавить автоматическую проверку Lighthouse после деплоя
10	Настроить деплой на Netlify через GitHub Actions
11	Добавить кастомную страницу 404
12	Настроить асинхронную загрузку неиспользуемых

Вариант	Дополнительное требование
	чанков
13	Добавить оптимизацию загрузки шрифтов
14	Настроить предзагрузку критических ресурсов
15	Добавить поддержку HTTP/2 Server Push

5.3. Продвинутый уровень (15 вариантов)

Требования: Реализовать мультитенантное приложение (несколько сайтов на одной сборке) или развернуть на нескольких регионах.

Вариант	Тип конфигурации
1	Мультитенантная сборка (разные темы для разных клиентов)
2	Мультиязычное приложение (i18n) с разными деплоями
3	Разделение на микрофронтенды с Module Federation
4	A/B тестирование через разные деплои
5	Канареечный деплой (canary deployment)
6	Синий-зелёный деплой (blue-green deployment)
7	Деплой на Edge (Cloudflare Workers, Vercel Edge)
8	Деплой с изоляцией пользовательских данных

Вариант	Тип конфигурации
9	Деплой с поддержкой динамических мета-тегов (SSR через CDN)
10	Деплой на IPFS (децентрализованный хостинг)
11	Деплой с автоматическим восстановлением после сбоев
12	Деплой с автоматическим масштабированием
13	Деплой с балансировкой нагрузки
14	Деплой с гео-распределением
15	Деплой с WebAssembly модулями

6. Контрольные вопросы

1. Что такое production-сборка и зачем она нужна?
2. Как выполнить сборку React-приложения с Vite?
3. Какие команды используются для сборки?
4. Что находится в папке dist после сборки?
5. Как проанализировать размер бандла?
6. Что такое код-сплиттинг и как он работает?
7. Как настроить перенаправления для SPA на Vercel?
8. Как настроить перенаправления для SPA на Netlify?
9. В чём проблема SPA-роутинга при обновлении страницы?
10. Как развернуть React-приложение на GitHub Pages?
11. Как настроить CI/CD для автоматического деплоя?
12. Что такое Tree Shaking и как он работает?
13. Как настроить кастомный домен?

14. Как оптимизировать время первой загрузки (FCP, LCP)?
15. Как настроить мониторинг production-приложения?

7. Требования к отчёту

Титульный лист (название работы, ФИО, группа, вариант)

Цель работы

Теоретическое обоснование (своими словами)

Выполнение практического примера (код, скриншоты сборки, анализа бандла, деплоя)

Выполнение индивидуального задания (код, скриншоты, ссылка на развёрнутое приложение)

Ответы на контрольные вопросы (не менее 5 по своему уровню)

Вывод

8. Критерии оценивания

Критерий	Базовый (max 35)	Средний (max 50)	Продвинутый (max 65)
Сборка проекта (npm run build)	10	5	5
Анализ бандла	10	10	5
Настройка перенаправлений для SPA	—	10	5
Деплой на Vercel или Netlify	15	10	5

Критерий	Базовый (max 35)	Средний (max 50)	Продвинутый (max 65)
Настройка CI/CD (GitHub Actions)	—	15	10
Кастомный домен и HTTPS	—	—	10
Мультитенантность или микрофронтенды	—	—	15
Качество кода и документация	—	—	10
Итого	35	50	65

Шкала перевода баллов в оценку:

Оценка	Базовый (35)	Средний (50)	Продвинутый (65)
5 (отлично)	32-35	46-50	59-65
4 (хорошо)	28-31	40-45	51-58
3 (удовлетворительно)	22-27	32-39	42-50
2 (неудовлетворительно)	0-21	0-31	0-41

Лабораторная работа №25

Интеграция фронтенда и бэкенда. CORS

1. Цель работы

Научиться связывать React-приложение с Django API, настраивать CORS (Cross-Origin Resource Sharing) для обеспечения безопасного кросс-доменного взаимодействия, обрабатывать preflight-запросы и выполнять HTTP-запросы из React к бэкенду.

2. Теоретическое обоснование

2.1. Что такое CORS и почему он возникает?

CORS (Cross-Origin Resource Sharing) - это механизм безопасности браузеров, который ограничивает возможность веб-страницы отправлять запросы к другому домену (или порту), отличному от того, с которого она была загружена.

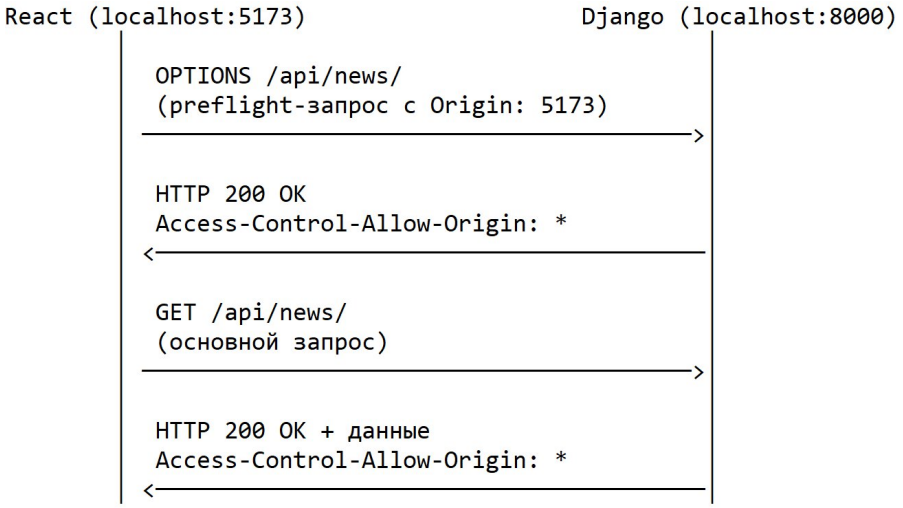
Когда возникает CORS-ошибка:

Сравниваемые части	Пример 1	Пример 2	Разрешено?
Протокол	http://	https://	Нет
Домен	localhost	api.example.com	Нет
Порт	:5173	:8000	Нет

Аналогия: Представьте, что вы живёте в одном доме (ваш React-приложение на порту 5173), а хотите получить посылку из соседнего дома (Django API на порту 8000). Охранник вашего дома (браузер) говорит: "Я не

пущу чужую посылку, если хозяин соседнего дома не даст разрешение". CORS - это как разрешение от соседнего дома.

2.2. Схема CORS-запроса



2.3. Preflight-запрос (предварительный запрос)

Перед отправкой "сложных" запросов (POST/PUT/DELETE с нестандартными заголовками или содержимым) браузер автоматически отправляет предварительный OPTIONS-запрос, чтобы узнать, разрешён ли основной запрос.

Когда отправляется preflight:

Метод: PUT, DELETE, PATCH (кроме GET и POST)

Заголовки: нестандартные (например, Authorization, X-Requested-With)

Content-Type: application/json (не application/x-www-form-urlencoded, multipart/form-data, text/plain)

2.4. Решение CORS-проблемы

На стороне Django необходимо настроить заголовки ответа:

Access-Control-Allow-Origin: http://localhost:5173

Access-Control-Allow-Methods: GET, POST, PUT, DELETE, OPTIONS

Access-Control-Allow-Headers: Content-Type, Authorization

Access-Control-Allow-Credentials: true

3. Практический пример

Что мы будем делать: Создадим React-приложение и Django API, настроим CORS для их взаимодействия, реализуем получение списка новостей и добавление новой новости.

Шаг 1. Настройка Django бэкенда

1.1. Установка необходимых пакетов

```
# Активируйте виртуальное окружение Django
pip install djangorestframework
pip install django-cors-headers
```

1.2. Настройка settings.py

python

```
# myproject/settings.py

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    # Third-party apps
    'rest_framework',
    'corsheaders',          # добавьте CORS
    'corsheaders.middleware.CorsMiddleware',

    # Your apps
    'api',
```

```
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',

    # CORS middleware должен быть как можно выше
    'corsheaders.middleware.CorsMiddleware',

    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

# Настройки CORS
# В разработке разрешаем все (ОСТОРОЖНО! Только для разработки!)
CORS_ALLOW_ALL_ORIGINS = True   # Только для разработки!

# В продакшене нужно указывать конкретные разрешённые источники:
# CORS_ALLOWED_ORIGINS = [
#     "http://localhost:5173",    # Vite
#     "http://localhost:3000",    # CRA
#     "https://myapp.vercel.app",
# ]

# Разрешаем отправку кук и заголовков авторизации
CORS_ALLOW_CREDENTIALS = True
```

```
# Разрешённые методы (опционально)
CORS_ALLOW_METHODS = [
    'GET',
    'POST',
    'PUT',
    'PATCH',
    'DELETE',
    'OPTIONS',
]

# Разрешённые заголовки (опционально)
CORS_ALLOW_HEADERS = [
    'accept',
    'accept-encoding',
    'authorization',
    'content-type',
    'dnt',
    'origin',
    'user-agent',
    'x-csrftoken',
    'x-requested-with',
]
```

1.3. Создание API

Файл api/models.py:

```
from django db import models

class News(models Model):
    title = models CharField(max_length=200)
```

```
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.title
```

Файл api/serializers.py:

```
from rest_framework import serializers
from .models import News

class NewsSerializer(serializers.ModelSerializer):
    class Meta:
        model = News
        fields = ['id', 'title', 'content', 'created_at']
```

Файл api/views.py:

```
from rest_framework import generics
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework import status
from .models import News
from .serializers import NewsSerializer

class NewsListCreateView(generics.ListCreateAPIView):
    queryset = News.objects.all().order_by('-created_at')
    serializer_class = NewsSerializer

class NewsDetailView(generics.RetrieveUpdateDestroyAPIView):
    queryset = News.objects.all()
    serializer_class = NewsSerializer
```

```
class HealthCheckView(APIView):
    def get(self, request):
        return Response({'status': 'ok', 'message': 'API работает'})
```

Файл api/urls.py:

```
from django.urls import path
from . import views

urlpatterns = [
    path('health/', views.HealthCheckView.as_view(), name='health'),
    path('news/', views.NewsListCreateView.as_view(), name='news-list'),
    path('news/<int:pk>/', views.NewsDetailView.as_view(), name='news-detail'),
]
```

Файл myproject/urls.py:

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include('api.urls')),
]
```

1.4. Запуск Django сервера

```
python manage.py makemigrations
python manage.py migrate
python manage.py runserver
```

Сервер Django запущен на http://localhost:8000.

Шаг 2. Настройка React фронтенда

2.1. Создание React-приложения

```
npm create vite@latest frontend -- --template react
cd frontend
npm install
npm install axios
npm run dev
```

Сервер React запущен на http://localhost:5173.

2.2. Настройка API-клиента

Файл src/api/axiosConfig.js:

```
import axios from 'axios';

const API_BASE_URL = 'http://localhost:8000/api';

// Создаём экземпляр axios с базовыми настройками
const api = axios.create({
  baseURL: API_BASE_URL,
  headers: {
    'Content-Type': 'application/json',
  },
  timeout: 10000,
});

// Интерсептор для логирования запросов
api.interceptors.request.use(
  (config) => {
    console.log(`📤 [${config.method?.toUpperCase()}] ${config.url}`);
    return config;
  },
  (error) => Promise.reject(error)
```

```
);

// Интерсептор для логирования ответов
api.interceptors.response.use(
  (response) => {
    console.log(`📥 [${response.status}] ${response.config.url}`);
    return response;
  },
  (error) => {
    console.error(`❌ Ошибка: ${error.message}`);
    return Promise.reject(error);
  }
);

export default api;
```

Файл src/api/newsApi.js:

```
import api from './axiosConfig';

// Проверка соединения с API
export const checkHealth = async () => {
  const response = await api.get('/health/');
  return response.data;
};

// Получение списка новостей
export const fetchNews = async () => {
  const response = await api.get('/news/');
  return response.data;
```

```
};

// Получение одной новости
export const fetchNewsById = async (id) => {
  const response = await api.get(`/news/${id}/`);
  return response.data;
};

// Создание новости
export const createNews = async (newsData) => {
  const response = await api.post('/news/', newsData);
  return response.data;
};

// Обновление новости
export const updateNews = async (id, newsData) => {
  const response = await api.put(`/news/${id}/`, newsData);
  return response.data;
};

// Удаление новости
export const deleteNews = async (id) => {
  await api.delete(`/news/${id}/`);
  return id;
};
```

2.3. Создание компонентов

Файл src/components/CORSStatus.jsx:

```
import { useState, useEffect } from 'react';
import { checkHealth } from '../api/newsApi';
```

```
const CORSStatus = () => {
  const [status, setStatus] = useState('checking');
  const [message, setMessage] = useState('');
  const [error, setError] = useState(null);

  useEffect(() => {
    const testConnection = async () => {
      try {
        const result = await checkHealth();
        setStatus('connected');
        setMessage(result.message);
        setError(null);
      } catch (err) {
        setStatus('failed');
        setError(err.message);
      }
    };

    testConnection();
  }, []);

  if (status === 'checking') {
    return <div style={{ padding: '10px', background: '#f0f0f0' }}>⏳ Проверка подключения к API...</div>;
  }

  if (status === 'failed') {
    return (
```

```
            <div style={{ padding: '10px', background: '#ffebee', color: '#c62828',
borderRadius: '4px' }}>
              <strong>✕ Ошибка CORS или подключения</strong>
              <p>Не удалось подключиться к API: {error}</p>
              <p>Убедитесь, что:</p>
              <ul>
                <li>Django сервер запущен на http://localhost:8000</li>
                <li>Установлен и настроен django-cors-headers</li>
                <li>CORS_ALLOW_ALL_ORIGINS = True (для разработки)</li>
              </ul>
            </div>
          );
        }

        return (
          <div style={{ padding: '10px', background: '#e8f5e9', color: '#2e7d32',
borderRadius: '4px' }}>
            <strong>✓ CORS настроен правильно!</strong>
            <p>Статус API: {message}</p>
          </div>
        );
      };

      export default CORSStatus;
```

Файл src/components/NewsList.jsx:

```
import { useState, useEffect } from 'react';
import { fetchNews, deleteNews } from '../api/newsApi';

const NewsList = ({ onRefresh, refreshTrigger }) => {
```

```
const [news, setNews] = useState([]);
const [loading, setLoading] = useState(true);
const [error, setError] = useState(null);

useEffect(() => {
  loadNews();
}, [refreshTrigger]);

const loadNews = async () => {
  try {
    setLoading(true);
    const data = await fetchNews();
    setNews(data);
    setError(null);
  } catch (err) {
    setError(err.message);
  } finally {
    setLoading(false);
  }
};

const handleDelete = async (id) => {
  if (!confirm('Удалить новость?')) return;

  try {
    await deleteNews(id);
    await loadNews();
    if (onRefresh) onRefresh();
  } catch (err) {
```

```
      alert('Ошибка удаления: ' + err.message);
    }
  };

  if (loading) return <div>🔄 Загрузка новостей...</div>;
  if (error) return <div style={{ color: 'red' }}>✕ Ошибка: {error}</div>;

  if (news.length === 0) {
    return <div>Нет новостей. Добавьте первую!</div>;
  }

  return (
    <div>
      {news.map(item => (
        <div key={item.id} style={{
          border: '1px solid #ddd',
          borderRadius: '8px',
          padding: '15px',
          marginBottom: '10px'
        }}>
          <h3>{item.title}</h3>
          <p>{item.content}</p>
          <small style={{ color: '#666' }}>
            Создано: {new Date(item.created_at).toLocaleString()}
          </small>
          <button
            onClick={() => handleDelete(item.id)}
            style={{ marginLeft: '10px', background: '#dc3545', color: 'white',
border: 'none', padding: '5px 10px' }}
```

```
          >
            Удалить
          </button>
        </div>
      ))}
    </div>
  );
};

export default NewsList;
```

Файл src/components/AddNewsForm.jsx:

```
import { useState } from 'react';
import { createNews } from '../api/newsApi';

const AddNewsForm = ({ onSuccess }) => {
  const [title, setTitle] = useState('');
  const [content, setContent] = useState('');
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState('');

  const handleSubmit = async (e) => {
    e.preventDefault();

    if (!title.trim() || !content.trim()) {
      setError('Заполните все поля');
      return;
    }

    setLoading(true);
```

```
    setError('');

    try {
      await createNews({ title, content });
      setTitle('');
      setContent('');
      if (onSuccess) onSuccess();
    } catch (err) {
      setError(err.message);
    } finally {
      setLoading(false);
    }
  };

  return (
    <form onSubmit={handleSubmit} style={{
      border: '1px solid #ddd',
      borderRadius: '8px',
      padding: '20px',
      marginBottom: '20px'
    }}>
      <h3>+ Добавить новость</h3>

      {error && (
        <div style={{ color: 'red', marginBottom: '10px' }}>✕ {error}</div>
      )}

      <div style={{ marginBottom: '10px' }}>
        <input
```

```
    type="text"
    placeholder="Заголовок"
    value={title}
    onChange={(e) => setTitle(e.target.value)}
    style={{ width: '100%', padding: '8px' }}
    disabled={loading}
  />
</div>

<div style={{ marginBottom: '10px' }}>
  <textarea
    placeholder="Содержание"
    value={content}
    onChange={(e) => setContent(e.target.value)}
    rows={4}
    style={{ width: '100%', padding: '8px' }}
    disabled={loading}
  />
</div>

<button
  type="submit"
  disabled={loading}
  style={{
    padding: '8px 16px',
    backgroundColor: '#28a745',
    color: 'white',
    border: 'none',
    borderRadius: '4px',
```

```
          cursor: 'pointer'
        }}
      >
        {loading ? 'Отправка...' : 'Добавить'}
      </button>
    </form>
  );
};

export default AddNewsForm;
```

Файл src/App.jsx (финальный):

```
import { useState } from 'react';
import CORSStatus from './components/CORSStatus';
import AddNewsForm from './components/AddNewsForm';
import NewsList from './components/NewsList';

function App() {
  const [refreshKey, setRefreshKey] = useState(0);

  const handleRefresh = () => {
    setRefreshKey(prev => prev + 1);
  };

  return (
    <div style={{ maxWidth: '800px', margin: '0 auto', padding: '20px' }}>
      <h1>Интеграция React + Django API</h1>

      <CORSStatus />
```

```
        <h2>Тестирование CORS</h2>
        <div style={{ marginBottom: '20px', padding: '10px', background: '#e3f2fd', borderRadius: '4px' }}>
          <p>📌 React работает на порту <strong>5173</strong></p>
          <p>📌 Django API работает на порту <strong>8000</strong></p>
          <p>🔧 CORS настроен через django-cors-headers</p>
        </div>

        <AddNewsForm onSuccess={handleRefresh} />
        <NewsList refreshTrigger={refreshKey} onRefresh={handleRefresh} />
      </div>
    );
  }

export default App;
```

Шаг 3. Запуск и тестирование

```
# Терминал 1: Запуск Django
cd myproject
python manage.py runserver

# Терминал 2: Запуск React
cd frontend
npm run dev
```

Откройте http://localhost:5173 и проверьте:

Статус CORS должен показывать успешное подключение

Можно добавлять новости

Можно удалять новости

В консоли браузера (F12 → Network) видны запросы к API

Шаг 4. Обработка CORS-ошибок (демонстрация проблемы)

Временно отключите CORS в Django (закомментируйте 'corsheaders' и CorsMiddleware в settings.py). Обновите React-приложение - вы увидите ошибку в консоли:

Access to fetch at 'http://localhost:8000/api/health/' from origin 'http://localhost:5173'

has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present.

Это демонстрирует, почему CORS необходим и как правильно его настраивать.

4. Настройка прокси в Vite (альтернатива CORS)

Вместо настройки CORS на бэкенде, можно настроить прокси в Vite для разработки.

Файл vite.config.js:

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'

export default defineConfig({
  plugins: [react()],
  server: {
    proxy: {
      '/api': {
        target: 'http://localhost:8000',
        changeOrigin: true,
        rewrite: (path) => path,
      },
```

```
    },
  },
})
```

После этого API-запросы будут отправляться на тот же порт, что и React, а Vite будет проксировать их на Django.

5. Индивидуальные задания

5.1. Базовый уровень (15 вариантов)

Требования: Настроить CORS, реализовать GET и POST запросы к API.

Вариант	Тип данных	Эндпоинты
1	Товары	GET /products/, POST /products/
2	Пользователи	GET /users/, POST /users/
3	Комментарии	GET /comments/, POST /comments/
4	Заказы	GET /orders/, POST /orders/
5	Категории	GET /categories/, POST /categories/
6	Отзывы	GET /reviews/, POST /reviews/
7	События	GET /events/, POST /events/
8	Курсы	GET /courses/, POST /courses/
9	Задачи	GET /tasks/, POST /tasks/

Вариант	Тип данных	Эндпоинты
10	Книги	GET /books/, POST /books/
11	Фильмы	GET /movies/, POST /movies/
12	Рецепты	GET /recipes/, POST /recipes/
13	Сотрудники	GET /employees/, POST /employees/
14	Клиенты	GET /clients/, POST /clients/
15	Проекты	GET /projects/, POST /projects/

5.2. Средний уровень (15 вариантов)

Требования: Добавить PUT/DELETE запросы, обработку ошибок, индикаторы загрузки.

Вариант	Дополнительный функционал
1	Редактирование и удаление товаров
2	Фильтрация списка на клиенте
3	Поиск по названию
4	Сортировка по полям
5	Пагинация
6	Загрузка файлов (изображения)

Вариант	Дополнительный функционал
7	Аутентификация (JWT)
8	Кэширование запросов
9	Офлайн-режим (Service Worker)
10	WebSocket соединение
11	Экспорт данных в CSV
12	Импорт данных из CSV
13	История действий
14	Отмена запросов (AbortController)
15	Retry-логика при ошибках

5.3. Продвинутый уровень (15 вариантов)

Требования: Реализовать полную интеграцию с аутентификацией, хранением токенов, обработкой 401 ошибок.

Вариант	Тип приложения	Функционал
1	Блог	Посты, комментарии, авторизация
2	Интернет-магазин	Товары, корзина, заказы

Вариант	Тип приложения	Функционал
3	Социальная сеть	Посты, лайки, подписки
4	Чат	Сообщения, комнаты, пользователи
5	CRM	Контакты, сделки, задачи
6	Доска задач	Проекты, задачи, исполнители
7	Календарь	События, приглашения, напоминания
8	Форум	Темы, сообщения, теги
9	База знаний	Статьи, категории, поиск
10	Система тикетов	Тикеты, ответы, статусы
11	Образовательная платформа	Курсы, уроки, тесты
12	Медицинская	Пациенты, записи,

Вариант	Тип приложения	Функционал
	система	рецепты
13	Недвижимость	Объекты, бронирования, отзывы
14	Финансовый трекер	Транзакции, категории, бюджет
15	Система бронирования	Услуги, время, клиенты

6. Контрольные вопросы

1. Что такое CORS и для чего он нужен?
2. Когда возникает CORS-ошибка?
3. Как установить django-cors-headers?
4. Как настроить CORS для разработки (CORS_ALLOW_ALL_ORIGINS)?
5. Как проверить, что CORS настроен правильно?
6. Что такое preflight-запрос (OPTIONS) и когда он отправляется?
7. В чём разница между CORS_ALLOW_ALL_ORIGINS и CORS_ALLOWED_ORIGINS?
8. Как настроить прокси в Vite для обхода CORS?
9. Как настроить отправку кук и заголовков авторизации?
10. Какие заголовки CORS должен возвращать сервер?
11. Как настроить CORS для production с несколькими доменами?
12. Как обрабатывать CORS-ошибки на клиенте?

13. Как настроить CORS для WebSocket соединений?
14. Как настроить CORS для использования с CloudFront/CDN?
15. Как протестировать CORS-настройки автоматически?

7. Требования к отчёту

Титульный лист (название работы, ФИО, группа, вариант)

Цель работы

Теоретическое обоснование (своими словами)

Выполнение практического примера (код, скриншоты)

Выполнение индивидуального задания (код, скриншоты)

Ответы на контрольные вопросы (не менее 5 по своему уровню)

Вывод

8. Критерии оценивания

Критерий	Базовый (max 35)	Средний (max 50)	Продвинутый (max 65)
Настройка django-cors-headers	10	5	5
GET-запросы к API	10	10	5
POST-запросы к API	10	10	5
PUT/DELETE запросы	—	10	5
Обработка	5	10	10

Критерий	Базовый (max 35)	Средний (max 50)	Продвинутый (max 65)
ошибок и загрузки			
Аутентификация (JWT)	—	—	15
Качество кода	—	5	10
Итого	35	50	65

Шкала перевода баллов в оценку:

Оценка	Базовый (35)	Средний (50)	Продвинутый (65)
5 (отлично)	32-35	46-50	59-65
4 (хорошо)	28-31	40-45	51-58
3 (удовлетворительно)	22-27	32-39	42-50
2 (неудовлетворительно)	0-21	0-31	0-41

Лабораторная работа №26

Тестирование бэкенда с pytest

1. Цель работы

Освоить написание тестов для Django-приложения с использованием pytest. Научиться устанавливать и настраивать pytest-django, писать тесты для

моделей, тестировать API с помощью APITestCase, использовать фикстуры для подготовки тестовых данных и библиотеку factory_boy для создания тестовых объектов.

2. Теоретическое обоснование

2.1. Что такое тестирование бэкенда?

Тестирование бэкенда - это процесс проверки корректности работы серверной части приложения: моделей, представлений, API-эндпоинтов, бизнес-логики и взаимодействия с базой данных.

Почему важно тестировать:

Причина	Объяснение
Обнаружение багов	Тесты находят ошибки до того, как они попадут в продакшн
Рефакторинг	Можно безопасно изменять код, зная, что тесты его не сломают
Документация	Тесты показывают, как должны работать модели и API
Регрессия	Тесты предотвращают повторное появление исправленных ошибок

2.2. Виды тестов в Django

Вид теста	Что проверяет	Инструменты
Unit-тесты	Отдельные функции,	pytest,

Вид теста	Что проверяет	Инструменты
	методы моделей	unittest
Интеграционные тесты	Взаимодействие компонентов	pytest, APITestCase
API-тесты	Корректность работы эндпоинтов	APITestCase, APIClient
Тесты моделей	Валидацию, методы, связи	pytest, ModelTestCase

2.3. Сравнение unittest и pytest

Характеристика	unittest (встроенный)	pytest
Синтаксис	Классы и методы assert*	Функции и простые assert
Фикстуры	setUp(), tearDown()	@pytest.fixture (гибче)
Плагины	Ограниченно	Много (pytest-django, pytest-cov)
Подробность ошибок	Базовая	Расширенная, с diff

Характеристика	unittest (встроенный)	pytest
Скорость	Средняя	Высокая

2.4. Структура тестов в Django с pytest

```
myproject/
├── tests/
│   ├── __init__.py
│   ├── conftest.py          # Фикстуры, общие для всех тестов
│   ├── test_models.py       # Тесты моделей
│   ├── test_views.py        # Тесты представлений
│   ├── test_api.py          # Тесты API
│   └── test_forms.py        # Тесты форм
```

2.5. Пирамида тестирования

```
      /\
     /  \
    /    \    E2E тесты (мало, медленно)
   /______\
  /        \
 /          \  Интеграционные тесты (средне)
/____________\
/            \
/              \  Unit-тесты (много, быстро)
```

3. Практический пример

Что мы будем делать: Создадим тесты для новостного API (модели Category, News, Comment) с использованием pytest, pytest-django и factory_boy.

Шаг 1. Установка и настройка

1.1. Установка пакетов

```
# Установка pytest и плагинов
pip install pytest pytest-django pytest-cov

# Установка factory_boy для создания тестовых данных
pip install factory-boy

# Установка faker для генерации случайных данных (опционально)
pip install faker
```

1.2. Настройка pytest в settings.py

Файл pytest.ini (в корне проекта):

```
[pytest]
DJANGO_SETTINGS_MODULE = myproject.settings
python_files = test_*.py *_test.py tests.py
testpaths = tests
addopts =
    --cov=api
    --cov-report=html
    --cov-report=term
    --nomigrations
```

1.3. Настройка базы данных для тестов

Файл myproject/settings.py (дополнение):

```
# Настройки для тестов
```

```
import sys

if 'test' in sys.argv:
    # Используем SQLite для тестов (быстрее)
    DATABASES = {
        'default': {
            'ENGINE': 'django.db.backends.sqlite3',
            'NAME': ':memory:',
        }
    }
else:
    # Продакшн база данных
    DATABASES = {
        'default': {
            'ENGINE': 'django.db.backends.postgresql',
            'NAME': 'myproject_db',
            'USER': 'myproject_user',
            'PASSWORD': 'password',
            'HOST': 'localhost',
            'PORT': '5432',
        }
    }
```

Шаг 2. Создание моделей

Файл api/models.py:

```
from django.db import models
from django.core.validators import MinValueValidator, MaxValueValidator
from django.contrib.auth.models import User

class Category(models.Model):
```

```
    name = models.CharField(max_length=100, unique=True)
    slug = models.SlugField(max_length=100, unique=True)
    description = models.TextField(blank=True)
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.name

    def get_absolute_url(self):
        from django.urls import reverse
        return reverse('category-detail', args=[self.slug])

class News(models.Model):
    STATUS_CHOICES = [
        ('draft', 'Черновик'),
        ('published', 'Опубликовано'),
        ('archived', 'В архиве'),
    ]

    title = models.CharField(max_length=200)
    slug = models.SlugField(max_length=200, unique=True)
    content = models.TextField()
    category = models.ForeignKey(Category, on_delete=models.CASCADE, related_name='news')
    author = models.ForeignKey(User, on_delete=models.CASCADE, related_name='news')
    status = models.CharField(max_length=10, choices=STATUS_CHOICES, default='draft')
    views = models.IntegerField(default=0)
```

```
        created_at = models.DateTimeField(auto_now_add=True)
        updated_at = models.DateTimeField(auto_now=True)

        def __str__(self):
            return self.title

        def publish(self):
            self.status = 'published'
            self.save()

        def archive(self):
            self.status = 'archived'
            self.save()

    class Comment(models.Model):
        news = models.ForeignKey(News, on_delete=models.CASCADE, related_name='comments')
        author = models.ForeignKey(User, on_delete=models.CASCADE)
        text = models.TextField()
        rating = models.IntegerField(
            validators=[MinValueValidator(1), MaxValueValidator(5)],
            default=5
        )
        is_approved = models.BooleanField(default=False)
        created_at = models.DateTimeField(auto_now_add=True)

        def __str__(self):
            return f'Comment by {self.author} on {self.news}'
```

```
    def approve(self):
        self.is_approved = True
        self.save()
```

Шаг 3. Создание фабрик для тестовых данных

Файл tests/factories.py:

```
import factory
from factory.django import DjangoModelFactory
from django.contrib.auth.models import User
from faker import Faker
from api.models import Category, News, Comment

fake = Faker()

class UserFactory(DjangoModelFactory):
    class Meta:
        model = User

    username = factory.Sequence(lambda n: f'user_{n}')
    email = factory.LazyAttribute(lambda obj: f'{obj.username}@example.com')
    first_name = factory.Faker('first_name')
    last_name = factory.Faker('last_name')

    @classmethod
    def _create(cls, model_class, *args, **kwargs):
        password = kwargs.pop('password', 'testpass123')
        user = super()._create(model_class, *args, **kwargs)
        user.set_password(password)
        user.save()
```

```
        return user

class CategoryFactory(DjangoModelFactory):
    class Meta:
        model = Category

    name = factory.Sequence(lambda n: f'Категория {n}')
    slug = factory.Sequence(lambda n: f'category-{n}')
    description = factory.Faker('text', max_nb_chars=200)

class NewsFactory(DjangoModelFactory):
    class Meta:
        model = News

    title = factory.Sequence(lambda n: f'Новость {n}')
    slug = factory.Sequence(lambda n: f'news-{n}')
    content = factory.Faker('paragraph', nb_sentences=10)
    category = factory.SubFactory(CategoryFactory)
    author = factory.SubFactory(UserFactory)
    status = 'published'
    views = factory.Faker('random_int', min=0, max=1000)

class CommentFactory(DjangoModelFactory):
    class Meta:
        model = Comment

    news = factory.SubFactory(NewsFactory)
    author = factory.SubFactory(UserFactory)
    text = factory.Faker('paragraph', nb_sentences=3)
```

```
    rating = factory.Faker('random_int', min=1, max=5)
    is_approved = True
```

Шаг 4. Создание фикстур (общие для всех тестов)

Файл tests/conftest.py:

```
import pytest
from django.contrib.auth.models import User
from rest_framework.test import APIClient
from .factories import UserFactory, CategoryFactory, NewsFactory, CommentFactory


@pytest.fixture
def api_client():
    """Фикстура для API-клиента"""
    return APIClient()


@pytest.fixture
def authenticated_client():
    """Фикстура для авторизованного API-клиента"""
    user = UserFactory()
    client = APIClient()
    client.force_authenticate(user=user)
    return client


@pytest.fixture
def admin_client():
    """Фикстура для API-клиента с правами администратора"""
    user = UserFactory(is_staff=True, is_superuser=True)
    client = APIClient()
    client.force_authenticate(user=user)
```

```
    return client


@pytest.fixture
def test_user():
    """Фикстура для создания тестового пользователя"""
    return UserFactory()

@pytest.fixture
def test_category():
    """Фикстура для создания тестовой категории"""
    return CategoryFactory()

@pytest.fixture
def test_news():
    """Фикстура для создания тестовой новости"""
    return NewsFactory()

@pytest.fixture
def test_comment():
    """Фикстура для создания тестового комментария"""
    return CommentFactory()

@pytest.fixture
def multiple_news():
    """Фикстура для создания нескольких новостей"""
    return NewsFactory create_batch(5)

@pytest.fixture
def sample_data():
```

```
    """Фикстура с набором связанных тестовых данных"""
    category = CategoryFactory(name='Технологии')
    author = UserFactory()
    news = NewsFactory.create_batch(3, category=category, author=author)
    comments = []
    for n in news:
        comments.extend(CommentFactory.create_batch(2, news=n, author=author))
    return {
        'category': category,
        'author': author,
        'news': news,
        'comments': comments
    }
```

Шаг 5. Тестирование моделей

Файл tests/test_models.py:

```
import pytest
from django.core.exceptions import ValidationError
from api.models import Category, News, Comment
from factories import CategoryFactory, NewsFactory, UserFactory, CommentFactory

# ========== Тесты для модели Category ==========

@pytest.mark.django_db
class TestCategoryModel:

    def test_category_creation(self):
        """Тест создания категории"""
```

```
        category = CategoryFactory(name='Наука')
        assert category.name == 'Наука'
        assert category.slug is not None
        assert isinstance(category, Category)

    def test_category_str_method(self):
        """Тест строкового представления категории"""
        category = CategoryFactory(name='Спорт')
        assert str(category) == 'Спорт'

    def test_category_unique_name(self):
        """Тест уникальности названия категории"""
        CategoryFactory(name='Уникальное')
        with pytest.raises(Exception):
            CategoryFactory(name='Уникальное')

    def test_category_get_absolute_url(self):
        """Тест метода get_absolute_url"""
        category = CategoryFactory(slug='test-category')
        url = category.get_absolute_url()
        assert '/category/test-category/' in url

# ========== Тесты для модели News ==========

@pytest.mark.django_db
class TestNewsModel:

    def test_news_creation(self):
        """Тест создания новости"""
```

```
    news = NewsFactory(title='Тестовая новость')
    assert news.title == 'Тестовая новость'
    assert news.views == 0
    assert news.status == 'published'

def test_news_str_method(self):
    """Тест строкового представления новости"""
    news = NewsFactory(title='Важная новость')
    assert str(news) == 'Важная новость'

def test_news_publish_method(self):
    """Тест метода публикации новости"""
    news = NewsFactory(status='draft')
    assert news.status == 'draft'
    news.publish()
    assert news.status == 'published'

def test_news_archive_method(self):
    """Тест метода архивации новости"""
    news = NewsFactory(status='published')
    assert news.status == 'published'
    news.archive()
    assert news.status == 'archived'

def test_news_views_increment(self):
    """Тест увеличения счётчика просмотров"""
    news = NewsFactory(views=0)
    news.views += 1
    news.save()
```

```
        assert news.views == 1

    def test_news_related_category(self):
        """Тест связи новости с категорией"""
        category = CategoryFactory(name='Политика')
        news = NewsFactory(category=category)
        assert news.category.name == 'Политика'
        assert category.news.count() == 1

# ========== Тесты для модели Comment ==========

@pytest.mark.django_db
class TestCommentModel:

    def test_comment_creation(self):
        """Тест создания комментария"""
        comment = CommentFactory(text='Отличная статья!')
        assert comment.text == 'Отличная статья!'
        assert comment.rating == 5
        assert comment.is_approved is True

    def test_comment_str_method(self):
        """Тест строкового представления комментария"""
        comment = CommentFactory()
        assert str(comment).startswith('Comment by')

    def test_comment_approve_method(self):
        """Тест метода одобрения комментария"""
        comment = CommentFactory(is_approved=False)
```

```
        assert comment.is_approved is False
        comment.approve()
        assert comment.is_approved is True

    def test_comment_rating_validation(self):
        """Тест валидации рейтинга комментария"""
        with pytest.raises(Exception):
            CommentFactory(rating=6)  # Рейтинг не может быть больше 5
        with pytest.raises(Exception):
            CommentFactory(rating=0)  # Рейтинг не может быть меньше 1

    def test_comment_relation_to_news(self):
        """Тест связи комментария с новостью"""
        news = NewsFactory()
        comment = CommentFactory(news=news)
        assert comment.news.id == news.id
        assert news.comments.count() == 1

    def test_comment_relation_to_user(self):
        """Тест связи комментария с пользователем"""
        user = UserFactory()
        comment = CommentFactory(author=user)
        assert comment.author.id == user.id
```

Шаг 6. Тестирование API с APITestCase

Файл tests/test_api.py:

```
import pytest
from rest_framework import status
from django.urls import reverse
```

```
from factories import CategoryFactory, NewsFactory, UserFactory, CommentFactory
from api.models import News, Comment

# ========== Тесты для API категорий ==========

@pytest.mark.django_db
class TestCategoryAPI:

    def test_list_categories(self, api_client, multiple_news):
        """Тест получения списка категорий"""
        url = reverse('category-list')
        response = api_client.get(url)

        assert response.status_code == status.HTTP_200_OK
        assert isinstance(response.data, list)

    def test_create_category_without_auth(self, api_client):
        """Тест создания категории без авторизации (должно быть запрещено)"""
        url = reverse('category-list')
        data = {'name': 'Новая категория', 'slug': 'new-category'}
        response = api_client.post(url, data)

        assert response.status_code == status.HTTP_401_UNAUTHORIZED

    def test_create_category_with_auth(self, authenticated_client):
        """Тест создания категории с авторизацией"""
        url = reverse('category-list')
```

```
        data = {'name': 'Новая категория', 'slug': 'new-category'}
        response = authenticated_client post(url, data)

        assert response status_code == status HTTP_201_CREATED
        assert response data['name'] == 'Новая категория'

    def test_get_category_detail(self, api_client, test_category):
        """Тест получения деталей категории"""
        url = reverse('category-detail', args=[test_category slug])
        response = api_client get(url)

        assert response status_code == status HTTP_200_OK
        assert response data['name'] == test_category name

# ========== Тесты для API новостей ==========

@pytest.mark.django_db
class TestNewsAPI:

    def test_list_news(self, api_client, multiple_news):
        """Тест получения списка новостей"""
        url = reverse('news-list')
        response = api_client get(url)

        assert response status_code == status HTTP_200_OK
        assert len(response data) == 5

    def test_list_news_with_pagination(self, api_client):
        """Тест пагинации списка новостей"""
```

```
    NewsFactory create_batch(25)

    url = reverse('news-list')
    response = api_client get(url, {'page': 1, 'page_size': 10})

    assert response status_code == status HTTP_200_OK
    assert len(response data['results']) == 10
    assert response data['count'] == 25

def test_create_news_without_auth(self, api_client, test_category):
    """Тест создания новости без авторизации"""
    url = reverse('news-list')
    data = {
        'title': 'Новая новость',
        'slug': 'new-news',
        'content': 'Текст новости',
        'category': test_category id
    }
    response = api_client post(url, data)

    assert response status_code == status HTTP_401_UNAUTHORIZED

def test_create_news_with_auth(self, authenticated_client, test_category):
    """Тест создания новости с авторизацией"""
    url = reverse('news-list')
    data = {
        'title': 'Новая новость',
        'slug': 'new-news',
        'content': 'Текст новости',
```

```
            'category': test_category.id
        }
        response = authenticated_client.post(url, data)

        assert response.status_code == status.HTTP_201_CREATED
        assert response.data['title'] == 'Новая новость'
        assert response.data['author'] == authenticated_client.handler._force_user.id

    def test_get_news_detail(self, api_client, test_news):
        """Тест получения деталей новости"""
        url = reverse('news-detail', args=[test_news.id])
        response = api_client.get(url)

        assert response.status_code == status.HTTP_200_OK
        assert response.data['id'] == test_news.id

    def test_update_news_without_auth(self, api_client, test_news):
        """Тест обновления новости без авторизации"""
        url = reverse('news-detail', args=[test_news.id])
        data = {'title': 'Обновлённое название'}
        response = api_client.patch(url, data)

        assert response.status_code == status.HTTP_401_UNAUTHORIZED

    def test_update_news_by_owner(self, authenticated_client, test_news):
        """Тест обновления новости владельцем"""
        # Назначаем автора новости текущим пользователем
        test_news.author = authenticated_client.handler._force_user
```

```
        test_news.save()

        url = reverse('news-detail', args=[test_news.id])
        data = {'title': 'Обновлённое название'}
        response = authenticated_client.patch(url, data)

        assert response.status_code == status.HTTP_200_OK
        assert response.data['title'] == 'Обновлённое название'

    def test_update_news_by_not_owner(self, authenticated_client, test_news):
        """Тест обновления новости не владельцем (должно быть запрещено)"""
        # Создаём другого пользователя
        from .factories import UserFactory
        other_user = UserFactory()
        test_news.author = other_user
        test_news.save()

        url = reverse('news-detail', args=[test_news.id])
        data = {'title': 'Обновлённое название'}
        response = authenticated_client.patch(url, data)

        assert response.status_code == status.HTTP_403_FORBIDDEN

    def test_delete_news_by_owner(self, authenticated_client, test_news):
        """Тест удаления новости владельцем"""
        test_news.author = authenticated_client.handler._force_user
        test_news.save()
```

```
    url = reverse('news-detail', args=[test_news.id])
    response = authenticated_client.delete(url)

    assert response.status_code == status.HTTP_204_NO_CONTENT
    assert not News.objects.filter(id=test_news.id).exists()

def test_filter_news_by_category(self, api_client):
    """Тест фильтрации новостей по категории"""
    category1 = CategoryFactory(name='Категория 1')
    category2 = CategoryFactory(name='Категория 2')

    NewsFactory.create_batch(3, category=category1)
    NewsFactory.create_batch(5, category=category2)

    url = reverse('news-list')
    response = api_client.get(url, {'category': category1.id})

    assert response.status_code == status.HTTP_200_OK
    assert len(response.data) == 3

def test_search_news(self, api_client):
    """Тест поиска новостей"""
    NewsFactory(title='Python разработка')
    NewsFactory(title='Django фреймворк')
    NewsFactory(title='Java программирование')

    url = reverse('news-list')
    response = api_client.get(url, {'search': 'Python'})
```

```
        assert response status_code == status HTTP_200_OK
        assert len(response data) == 1
        assert 'Python' in response data[0]['title']

# ========== Тесты для API комментариев ==========

@pytest.mark.django_db
class TestCommentAPI:

    def test_list_comments_by_news(self, api_client, test_news):
        """Тест получения комментариев к новости"""
        CommentFactory create_batch(3, news=test_news)

        url = reverse('news-comments', args=[test_news id])
        response = api_client get(url)

        assert response status_code == status HTTP_200_OK
        assert len(response data) == 3

    def test_create_comment_without_auth(self, api_client, test_news):
        """Тест создания комментария без авторизации"""
        url = reverse('news-comments', args=[test_news id])
        data = {'text': 'Хорошая статья!', 'rating': 5}
        response = api_client post(url, data)

        assert response status_code == status HTTP_401_UNAUTHORIZED

    def test_create_comment_with_auth(self, authenticated_client, test_news):
```

```
        """Тест создания комментария с авторизацией"""
        url = reverse('news-comments', args=[test_news.id])
        data = {'text': 'Хорошая статья!', 'rating': 5}
        response = authenticated_client.post(url, data)

        assert response.status_code == status.HTTP_201_CREATED
        assert response.data['text'] == 'Хорошая статья!'
        assert response.data['rating'] == 5

    def test_create_comment_with_invalid_rating(self, authenticated_client, test_news):
        """Тест создания комментария с неверным рейтингом"""
        url = reverse('news-comments', args=[test_news.id])
        data = {'text': 'Текст', 'rating': 10}
        response = authenticated_client.post(url, data)

        assert response.status_code == status.HTTP_400_BAD_REQUEST

    def test_delete_comment_by_owner(self, authenticated_client, test_comment):
        """Тест удаления комментария владельцем"""
        test_comment.author = authenticated_client.handler._force_user
        test_comment.save()

        url = reverse('comment-detail', args=[test_comment.id])
        response = authenticated_client.delete(url)

        assert response.status_code == status.HTTP_204_NO_CONTENT
        assert not Comment.objects.filter(id=test_comment.id).exists()
```

Шаг 7. Тестирование с использованием фикстур sample_data

Файл tests/test_integration.py:

python

```
import pytest
from rest_framework import status
from django urls import reverse
from  factories import UserFactory


@pytest.mark.django_db
class TestIntegration:


    def test_full_workflow(self, api_client, sample_data):
        """Интеграционный тест полного сценария работы с API"""

        # 1. Пользователь регистрируется
        user_data = {
            'username': 'testuser',
            'email': 'test@example.com',
            'password': 'securepass123'
        }
        url = reverse('register')
        response = api_client post(url, user_data)
        assert response status_code == status HTTP_201_CREATED

        # 2. Пользователь входит в систему
        login_data = {
            'username': 'testuser',
            'password': 'securepass123'
        }
```

```
url = reverse('login')
response = api_client post(url, login_data)
assert response status_code == status HTTP_200_OK
token = response data['access']

# 3. Пользователь создаёт новость
api_client credentials(HTTP_AUTHORIZATION=f'Bearer {token}')
news_data = {
    'title': 'Интеграционная новость',
    'slug': 'integration-news',
    'content': 'Содержание интеграционной новости',
    'category': sample_data['category'].id
}
url = reverse('news-list')
response = api_client post(url, news_data)
assert response status_code == status HTTP_201_CREATED
news_id = response data['id']

# 4. Пользователь добавляет комментарий
comment_data = {
    'text': 'Отличная новость!',
    'rating': 5
}
url = reverse('news-comments', args=[news_id])
response = api_client post(url, comment_data)
assert response status_code == status HTTP_201_CREATED

# 5. Пользователь получает свои новости
url = reverse('my-news')
```

```
        response = api_client.get(url)
        assert response.status_code == status.HTTP_200_OK
        assert len(response.data) == 1

        # 6. Пользователь удаляет новость
        url = reverse('news-detail', args=[news_id])
        response = api_client.delete(url)
        assert response.status_code == status.HTTP_204_NO_CONTENT

    def test_category_with_news_statistics(self, api_client, sample_data):
        """Тест получения статистики по категориям"""
        url = reverse('category-stats')
        response = api_client.get(url)

        assert response.status_code == status.HTTP_200_OK
        for item in response.data:
            if item['name'] == sample_data['category'].name:
                assert item['news_count'] == len(sample_data['news'])
                assert item['comments_count'] == len(sample_data['comments'])

    def test_news_popularity_ranking(self, api_client):
        """Тест ранжирования новостей по популярности"""
        # Создаём новости с разным количеством просмотров
        NewsFactory(views=100, title='Популярная новость')
        NewsFactory(views=50, title='Средняя новость')
        NewsFactory(views=10, title='Непопулярная новость')

        url = reverse('popular-news')
        response = api_client.get(url)
```

```
        assert response.status_code == status.HTTP_200_OK
        assert response.data[0]['views'] == 100
        assert response.data[1]['views'] == 50
        assert response.data[2]['views'] == 10
```

Шаг 8. Запуск тестов

```
# Запуск всех тестов
pytest

# Запуск с подробным выводом
pytest -v

# Запуск конкретного файла
pytest tests/test_models.py

# Запуск конкретного теста
pytest tests/test_models.py::TestNewsModel::test_news_creation -v

# Запуск с отчётом о покрытии кода
pytest --cov=api --cov-report=html

# Запуск с остановкой при первой ошибке
pytest -x

# Запуск без миграций (быстрее)
pytest --nomigrations
```

4. Полезные приёмы тестирования

4.1. Маркеры для группировки тестов

```python
import pytest

@pytest.mark.slow
def test_slow_operation():
    """Медленный тест"""
    pass

@pytest.mark.skip(reason="Not implemented yet")
def test_not_implemented():
    pass

@pytest.mark.skipif(condition, reason="Skip condition")
def test_conditional_skip():
    pass
```

Запуск по маркеру:

```
pytest -m "not slow"   # все кроме медленных
pytest -m "api"        # только API тесты
```

4.2. Параметризация тестов

```python
import pytest

@pytest.mark.parametrize('title,expected', [
    ('Короткое', True),
    ('Очень длинное название новости которое превышает максимальную длину', False),
])
def test_title_validation(title, expected):
    """Тест валидации заголовка с разными значениями"""
    is_valid = len(title) <= 200
    assert is_valid == expected
```

```
@pytest.mark.parametrize('rating,should_pass', [
    (5, True),   # Максимальный рейтинг - проходит
    (3, True),   # Средний рейтинг - проходит
    (1, True),   # Минимальный рейтинг - проходит
    (0, False),  # Меньше минимума - ошибка
    (6, False),  # Больше максимума - ошибка
])
def test_comment_rating(rating, should_pass):
    """Тест валидации рейтинга с разными значениями"""
    comment = CommentFactory build(rating=rating)
    if should_pass:
        assert comment full_clean() is None
    else:
        with pytest raises(ValidationError):
            comment full_clean()
```

4.3. Мокирование внешних зависимостей

```
from unittest mock import patch, Mock

@pytest.mark.django_db
def test_send_notification_on_comment():
    with patch('api.services.send_email') as mock_send_email:
        comment = CommentFactory()
        mock_send_email assert_called_once()
        args, kwargs = mock_send_email call_args
        assert kwargs['subject'] == 'Новый комментарий'
```

5. Индивидуальные задания

5.1. Базовый уровень (15 вариантов)

Требования: Написать тесты для моделей, используя factory_boy.

Вариант	Модель	Поля для тестирования
1	Category	name, slug, unique, str
2	News	title, slug, content, status
3	Comment	text, rating, is_approved
4	User	username, email, password
5	Product	name, price, stock
6	Order	total, status, created_at
7	Review	text, rating, is_published
8	Tag	name, slug, created_at
9	Event	title, date, location
10	Ticket	code, price, is_used
11	Payment	amount, method, status
12	Message	text, is_read, created_at
13	Like	user, post, created_at
14	Follow	follower, followed, created_at

Вариант	Модель	Поля для тестирования
15	Notification	text, is_read, user

5.2. Средний уровень (15 вариантов)

Требования: Написать тесты для API (CRUD операции, фильтрация, пагинация).

Вариант	API эндпоинты	Дополнительные тесты
1	/api/categories/	Фильтрация по name, поиск
2	/api/news/	Фильтрация по status, пагинация
3	/api/comments/	Фильтрация по rating, сортировка
4	/api/users/	Регистрация, профиль, обновление
5	/api/products/	Фильтрация по price, сортировка
6	/api/orders/	Фильтрация по status, дате
7	/api/reviews/	Фильтрация по rating, moderation
8	/api/tags/	Поиск по name

Вариант	API эндпоинты	Дополнительные тесты
9	/api/events/	Фильтрация по date
10	/api/tickets/	Проверка использования
11	/api/payments/	Фильтрация по status
12	/api/messages/	Отметка о прочтении
13	/api/likes/	Уникальность like
14	/api/follows/	Уникальность подписки
15	/api/notifications/	Отметка о прочтении

5.3. Продвинутый уровень (15 вариантов)

Требования: Написать интеграционные тесты для полных сценариев (регистрация → логин → создание → модерация).

Вариант	Сценарий
1	Пользователь → создание новости → получение списка → удаление
2	Пользователь → создание комментария → модерация → публикация
3	Админ → создание категории → назначение модератора
4	Пользователь → заказ товара → оплата → получение статуса
5	Пользователь → отзыв → обновление → удаление

Вариант	Сценарий
6	Пользователь → подписка на категорию → уведомление
7	Пользователь → создание события → регистрация участников
8	Админ → создание промокода → применение пользователем
9	Пользователь → создание тикета → получение ответа
10	Пользователь → создание поста → лайк → комментирование
11	Пользователь → создание чата → отправка сообщений
12	Пользователь → создание задачи → назначение → выполнение
13	Пользователь → регистрация → подтверждение email
14	Админ → создание роли → назначение прав
15	Пользователь → импорт данных → валидация → сохранение

6. Контрольные вопросы

1. Что такое тестирование и зачем оно нужно в Django?
2. Как установить pytest-django?
3. Как написать простой тест для модели?
4. Что такое фикстуры в pytest и как их использовать?

5. Как проверить, что объект был создан в базе данных?
6. В чём разница между @pytest.mark.django_db и @pytest.mark.django_db(transaction=True)?
7. Как тестировать API с помощью APIClient?
8. Что такое factory_boy и зачем он нужен?
9. Как протестировать аутентификацию (JWT)?
10. Как протестировать обработку ошибок и валидацию?
11. Как протестировать сигналы Django?
12. Как протестировать кастомные менеджеры моделей?
13. Как измерить покрытие кода тестами?
14. Как тестировать асинхронный код (Django Channels)?
15. Как настроить CI/CD для автоматического запуска тестов?

7. Требования к отчёту

Титульный лист (название работы, ФИО, группа, вариант)

Цель работы

Теоретическое обоснование (своими словами)

Выполнение практического примера (код, скриншоты)

Выполнение индивидуального задания (код, результаты тестов, покрытие)

Ответы на контрольные вопросы (не менее 5 по своему уровню)

Вывод

8. Критерии оценивания

Критерий	Базовый (max 35)	Средний (max 50)	Продвинутый (max 65)
Установка и настройка pytest	5	5	5
Тесты моделей	15	10	5
Фабрики (factory_boy)	10	10	5
Фикстуры (conftest.py)	5	5	5
Тесты API (GET, POST)	—	15	10
Тесты API (PUT, DELETE)	—	5	10
Интеграционные тесты	—	—	15
Качество кода и покрытие	—	—	10
Итого	35	50	65

Шкала перевода баллов в оценку:

Оценка	Базовый (35)	Средний (50)	Продвинутый (65)
5 (отлично)	32-35	46-50	59-65
4 (хорошо)	28-31	40-45	51-58
3 (удовлетворительно)	22-27	32-39	42-50
2 (неудовлетворительно)	0-21	0-31	0-41

Лабораторная работа №27

E2E-тестирование с Cypress

1. Цель работы

Научиться писать сквозные (end-to-end) тесты для веб-приложений с использованием Cypress. Освоить установку и настройку Cypress, написание тестов авторизации, тестов создания постов, тестирование основного сценария пользователя ("счастливый путь"), а также методы отладки и поиска ошибок в тестах.

2. Теоретическое обоснование

2.1. Что такое E2E-тестирование?

E2E (End-to-End) тестирование - это метод тестирования, который проверяет работу приложения целиком, от начала до конца, имитируя реальные действия пользователя.

Вид теста	Что проверяет	Скорость	Сложность
Unit-тесты	Отдельные функции	Очень быстро	Низкая
Интеграционные тесты	Взаимодействие компонентов	Быстро	Средняя
E2E-тесты	Полный пользовательский сценарий	Медленно	Высокая

Аналогия: E2E-тест - это как контрольная закупка в магазине. Проверяющий приходит как обычный покупатель, проходит весь путь: выбирает товары, оплачивает, получает чек. Если всё работает - тест пройден.

2.2. Что такое Cypress?

Cypress - это современный инструмент для E2E-тестирования веб-приложений. Он запускает тесты в реальном браузере и позволяет взаимодействовать с приложением как настоящий пользователь.

Преимущества Cypress:

Преимущество	Описание
Работа в реальном браузере	Тесты выполняются в Chrome, Firefox, Edge
Автоматическое ожидание	Cypress ждёт загрузки элементов без sleep

Преимущество	Описание
Time-travel	Можно навести на шаг в логе и увидеть состояние приложения
Отладка	Интеграция с DevTools браузера
Скриншоты и видео	Автоматически создаются при падении тестов

2.3. Сравнение Cypress с альтернативами

Инструмент	Язык	Подход	Лучше для
Cypress	JavaScript	Запуск в браузере	Веб-приложений
Playwright	JS/TS, Python, C#	Запуск в браузере	Кросс-браузерных тестов
Selenium	Многие языки	Удалённое управление браузером	Комплексных сценариев
Puppeteer	JavaScript	Управление Chrome/Chromium	Тестирования Chrome

2.4. Структура теста в Cypress

```
describe('Название группы тестов', () => {
```

```
  beforeEach(() => {
    // Выполняется перед каждым тестом
    cy.visit('/');
  });

  it('Название конкретного теста', () => {
    // Поиск элемента
    cy.get('button').click();

    // Проверка
    cy.url().should('include', '/dashboard');
    cy.contains('Добро пожаловать').should('be.visible');
  });
});
```

3. Практический пример

Что мы будем делать: Создадим E2E-тесты для новостного приложения: авторизация, создание новости, проверка списка новостей.

Шаг 1. Установка и настройка Cypress

1.1. Установка Cypress

```
# Перейдите в папку с React-приложением
cd frontend

# Установка Cypress
npm install -D cypress

# Или для глобальной установки
npm install -g cypress
```

1.2. Запуск Cypress

```
# Запуск интерфейса Cypress
npx cypress open

# Запуск тестов в headless-режиме
npx cypress run
```

1.3. Структура папок после установки

```
frontend/
├── cypress/
│   ├── e2e/          # Тесты (создаём сами)
│   ├── fixtures/     # Тестовые данные
│   ├── support/      # Вспомогательные функции
│   │   ├── commands.js # Кастомные команды
│   │   └── e2e.js    # Конфигурация
│   └── screenshots/  # Скриншоты при падении
├── cypress.config.js  # Конфигурационный файл
└── package.json
```

Шаг 2. Настройка конфигурации

Файл cypress.config.js:

```
const { defineConfig } = require('cypress');

module exports = defineConfig({
  e2e: {
    baseUrl: 'http://localhost:5173',  // URL вашего React-приложения
    supportFile: 'cypress/support/e2e.js',
    specPattern: 'cypress/e2e/**/*.cy.{js,jsx,ts,tsx}',
    viewportWidth: 1280,
    viewportHeight: 720,
    defaultCommandTimeout: 10000,      // Таймаут для команд (10 сек)
    requestTimeout: 10000,
```

```
    responseTimeout: 30000,
    screenshotOnRunFailure: true,
    video: true,
    videoCompression: 32,
    retries: {
      runMode: 2,       // Количество повторных попыток при запуске через
CLI
      openMode: 0,      // Без повторов в интерактивном режиме
    },
  },
});
```

Шаг 3. Создание кастомных команд

Файл cypress/support/commands.js:

```
// 1. Команда для логина
Cypress Commands add('login', (username, password) => {
  cy.visit('/login');
  cy.get('[data-testid="username-input"]').type(username);
  cy.get('[data-testid="password-input"]').type(password);
  cy.get('[data-testid="login-button"]').click();

  // Ждём перенаправления на главную
  cy.url().should('include', '/');
});

// 2. Команда для логина через API (быстрее)
Cypress Commands add('loginByApi', (username, password) => {
  cy.request({
    method: 'POST',
    url: 'http://localhost:8000/api/token/',
```

```
    body: { username, password },
  }).then((response) => {
    expect(response.status).to.eq(200);
    window.localStorage.setItem('access_token', response.body.access);
    window.localStorage.setItem('refresh_token', response.body.refresh);
  });
  cy.visit('/');
});

// 3. Команда для создания тестовой новости
Cypress.Commands.add('createNews', (title, content) => {
  cy.get('[data-testid="add-news-button"]').click();
  cy.get('[data-testid="news-title"]').type(title);
  cy.get('[data-testid="news-content"]').type(content);
  cy.get('[data-testid="submit-news"]').click();
});

// 4. Команда для очистки тестовых данных
Cypress.Commands.add('cleanupTestData', () => {
  cy.request({
    method: 'DELETE',
    url: 'http://localhost:8000/api/test-cleanup/',
    headers: {
      Authorization: `Bearer ${window.localStorage.getItem('access_token')}`,
    },
  });
});
```

Шаг 4. Создание тестовых данных (fixtures)

Файл cypress/fixtures/users.json:

```
{
  "validUser": {
    "username": "testuser",
    "password": "TestPass123!",
    "email": "test@example.com"
  },
  "invalidUser": {
    "username": "wronguser",
    "password": "wrongpass"
  },
  "admin": {
    "username": "admin",
    "password": "AdminPass123!"
  }
}
```

Файл cypress/fixtures/news.json:

```
{
  "validNews": {
    "title": "Тестовая новость Cypress",
    "content": "Это содержание тестовой новости, созданной с помощью Cypress E2E тестов."
  },
  "longTitle": {
    "title": "Очень длинный заголовок новости, который превышает максимальную длину в 200 символов и должен вызвать ошибку валидации при попытке сохранения такой новости в базе данных",
    "content": "Содержание"
  },
  "emptyTitle": {
```

```
  "title": "",
  "content": "Содержание новости без заголовка"
 }
}
```

Шаг 5. Тестирование авторизации

Файл cypress/e2e/auth.cy.js:

```
describe('Тестирование авторизации', () => {

 beforeEach(() => {
  cy.visit('/login');
 });

 describe('Позитивные сценарии', () => {

  it('Должен успешно войти с валидными данными', () => {
   cy.fixture('users').then((users) => {
    const user = users.validUser;

    cy.get('[data-testid="username-input"]').type(user.username);
    cy.get('[data-testid="password-input"]').type(user.password);
    cy.get('[data-testid="login-button"]').click();

    // Проверки
    cy.url().should('include', '/');
    cy.contains('Добро пожаловать').should('be.visible');
    cy.get('[data-testid="user-menu"]').should('contain', user.username);
   });
  });
```

```
  it('Должен сохранять сессию после перезагрузки страницы', () => {
    cy.fixture('users').then((users) => {
      cy.login(users.validUser.username, users.validUser.password);
      cy.reload();

      cy.get('[data-testid="user-menu"]').should('be.visible');
      cy.url().should('include', '/');
    });
  });

  it('Должен корректно выходить из системы', () => {
    cy.fixture('users').then((users) => {
      cy.login(users.validUser.username, users.validUser.password);
      cy.get('[data-testid="logout-button"]').click();

      cy.url().should('include', '/login');
      cy.get('[data-testid="login-button"]').should('be.visible');
    });
  });
});

describe('Негативные сценарии', () => {

  it('Должен показать ошибку при пустом имени пользователя', () => {
    cy.get('[data-testid="password-input"]').type('password123');
    cy.get('[data-testid="login-button"]').click();

    cy.get('[data-testid="error-message"]').should('be.visible');
```

```
    cy.get('[data-testid="error-message"]').should('contain', 'Имя пользователя обязательно');
  });

  it('Должен показать ошибку при пустом пароле', () => {
    cy.get('[data-testid="username-input"]').type('testuser');
    cy.get('[data-testid="login-button"]').click();

    cy.get('[data-testid="error-message"]').should('be.visible');
  });

  it('Должен показать ошибку при неверных учётных данных', () => {
    cy.fixture('users').then((users) => {
      const user = users.invalidUser;

      cy.get('[data-testid="username-input"]').type(user.username);
      cy.get('[data-testid="password-input"]').type(user.password);
      cy.get('[data-testid="login-button"]').click();

      cy.get('[data-testid="error-message"]').should('be.visible');
      cy.get('[data-testid="error-message"]').should('contain', 'Неверное имя пользователя или пароль');
    });
  });

  it('Должен показывать индикатор загрузки во время отправки', () => {
    cy.fixture('users').then((users) => {
      const user = users.validUser;
```

```
    cy.get('[data-testid="username-input"]').type(user.username);
    cy.get('[data-testid="password-input"]').type(user.password);
    cy.get('[data-testid="login-button"]').click();

    cy.get('[data-testid="loading-spinner"]').should('be.visible');
    cy.get('[data-testid="loading-spinner"]').should('not.exist', { timeout: 5000 });
  });
 });
});

describe('Регистрация', () => {

 it('Должен успешно зарегистрировать нового пользователя', () => {
  const uniqueUsername = `testuser_${Date.now()}`;

  cy.visit('/register');
  cy.get('[data-testid="username-input"]').type(uniqueUsername);
  cy.get('[data-testid="email-input"]').type(`${uniqueUsername}@example.com`);
  cy.get('[data-testid="password-input"]').type('TestPass123!');
  cy.get('[data-testid="confirm-password-input"]').type('TestPass123!');
  cy.get('[data-testid="register-button"]').click();

  cy.url().should('include', '/login');
  cy.contains('Регистрация прошла успешно').should('be.visible');
 });

 it('Должен показать ошибку при несовпадении паролей', () => {
```

```
    cy.visit('/register');
    cy.get('[data-testid="username-input"]').type('newuser');
    cy.get('[data-testid="email-input"]').type('newuser@example.com');
    cy.get('[data-testid="password-input"]').type('TestPass123!');
    cy.get('[data-testid="confirm-password-input"]').type('DifferentPass123!');
    cy.get('[data-testid="register-button"]').click();

    cy.get('[data-testid="error-message"]').should('contain', 'Пароли не совпадают');
  });

  it('Должен показать ошибку при коротком пароле', () => {
    cy.visit('/register');
    cy.get('[data-testid="username-input"]').type('newuser');
    cy.get('[data-testid="email-input"]').type('newuser@example.com');
    cy.get('[data-testid="password-input"]').type('123');
    cy.get('[data-testid="confirm-password-input"]').type('123');
    cy.get('[data-testid="register-button"]').click();

    cy.get('[data-testid="error-message"]').should('contain', 'минимум 8 символов');
  });
 });
});
```

Шаг 6. Тестирование создания новости

Файл cypress/e2e/news.cy.js:

```
describe('Тестирование работы с новостями', () => {
```

```
  beforeEach(() => {
    cy.fixture('users').then((users) => {
      cy.login(users.validUser.username, users.validUser.password);
    });
  });

  describe('Создание новости', () => {

    it('Должен создать новость с валидными данными', () => {
      cy.fixture('news').then((news) => {
        const testNews = news.validNews;

        cy.get('[data-testid="add-news-button"]').click();
        cy.get('[data-testid="news-title"]').type(testNews.title);
        cy.get('[data-testid="news-content"]').type(testNews.content);
        cy.get('[data-testid="submit-news"]').click();

        // Проверка успешного добавления
        cy.contains(testNews.title).should('be.visible');
        cy.contains(testNews.content.substring(0, 50)).should('be.visible');
        cy.get('[data-testid="success-toast"]').should('be.visible');
        cy.get('[data-testid="success-toast"]').should('contain', 'Новость
добавлена');
      });
    });

    it('Должен показать ошибку при пустом заголовке', () => {
      cy.fixture('news').then((news) => {
        const testNews = news.emptyTitle;
```

```
        cy.get('[data-testid="add-news-button"]').click();
        cy.get('[data-testid="news-title"]').clear();
        cy.get('[data-testid="news-content"]').type(testNews.content);
        cy.get('[data-testid="submit-news"]').click();

        cy.get('[data-testid="title-error"]').should('be.visible');
        cy.get('[data-testid="title-error"]').should('contain', 'Заголовок обязателен');
      });
    });

    it('Должен показать ошибку при слишком длинном заголовке', () => {
      cy.fixture('news').then((news) => {
        const testNews = news.longTitle;

        cy.get('[data-testid="add-news-button"]').click();
        cy.get('[data-testid="news-title"]').type(testNews.title);
        cy.get('[data-testid="news-content"]').type(testNews.content);
        cy.get('[data-testid="submit-news"]').click();

        cy.get('[data-testid="title-error"]').should('be.visible');
        cy.get('[data-testid="title-error"]').should('contain', 'Максимум 200 символов');
      });
    });

    it('Должен закрыть форму без сохранения при нажатии "Отмена"', () => {
```

```
    cy.get('[data-testid="add-news-button"]').click();
    cy.get('[data-testid="news-title"]').type('Тестовый заголовок');
    cy.get('[data-testid="cancel-button"]').click();

    // Форма должна закрыться
    cy.get('[data-testid="news-modal"]').should('not.exist');
  });
});

describe('Редактирование новости', () => {

  beforeEach(() => {
    // Создаём тестовую новость перед редактированием
    cy.fixture('news').then((news) => {
      cy.createNews(news.validNews.title, news.validNews.content);
    });
  });

  it('Должен отредактировать существующую новость', () => {
    cy.get('[data-testid="edit-news-button"]').first().click();
    cy.get('[data-testid="news-title"]').clear().type('Обновлённый
заголовок');
    cy.get('[data-testid="news-content"]').clear().type('Обновлённое
содержание');
    cy.get('[data-testid="submit-news"]').click();

    cy.contains('Обновлённый заголовок').should('be.visible');
    cy.contains('Обновлённое содержание').should('be.visible');
  });
```

```
});

describe('Удаление новости', () => {

  beforeEach(() => {
    cy.fixture('news').then((news) => {
      cy.createNews(news.validNews.title, news.validNews.content);
    });
  });

  it('Должен удалить новость после подтверждения', () => {
    const newsTitle = 'Тестовая новость Cypress';

    cy.contains(newsTitle).should('be.visible');

    cy.get('[data-testid="delete-news-button"]').first().click();
    cy.get('[data-testid="confirm-delete"]').click();

    cy.contains(newsTitle).should('not.exist');
    cy.get('[data-testid="success-toast"]').should('contain', 'Новость
удалена');
  });

  it('Не должен удалять новость при отмене подтверждения', () => {
    const newsTitle = 'Тестовая новость Cypress';

    cy.contains(newsTitle).should('be.visible');

    cy.get('[data-testid="delete-news-button"]').first().click();
```

```
    cy.get('[data-testid="cancel-delete"]').click();

    cy.contains(newsTitle).should('be.visible');
  });
});

describe('Просмотр новостей', () => {

  it('Должен открыть детальную страницу новости', () => {
    cy.get('[data-testid="news-link"]').first().click();

    cy.url().should('include', '/news/');
    cy.get('[data-testid="news-title"]').should('be.visible');
    cy.get('[data-testid="news-content"]').should('be.visible');
    cy.get('[data-testid="back-button"]').click();

    cy.url().should('include', '/');
  });
});
});
```

Шаг 7. Тестирование основного сценария ("Счастливый путь")

Файл cypress/e2e/happy-path.cy.js:

```
describe('Основной сценарий пользователя (Счастливый путь)', () => {

  it('Полный цикл: регистрация → логин → создание новости → просмотр → удаление', () => {
    const uniqueUsername = `user_${Date.now()}`;
    const newsTitle = `Новость ${Date.now()}`;
```

```
        const newsContent = 'Содержание тестовой новости для проверки полного цикла';

        // ===== ШАГ 1: Регистрация =====
        cy.visit('/register');

        cy.get('[data-testid="username-input"]').type(uniqueUsername);
        cy.get('[data-testid="email-input"]').type(`${uniqueUsername}@example.com`);
        cy.get('[data-testid="password-input"]').type('TestPass123!');
        cy.get('[data-testid="confirm-password-input"]').type('TestPass123!');
        cy.get('[data-testid="register-button"]').click();

        cy.url().should('include', '/login');
        cy.contains('Регистрация прошла успешно').should('be.visible');

        // ===== ШАГ 2: Логин =====
        cy.get('[data-testid="username-input"]').type(uniqueUsername);
        cy.get('[data-testid="password-input"]').type('TestPass123!');
        cy.get('[data-testid="login-button"]').click();

        cy.url().should('include', '/');
        cy.contains(`Добро пожаловать, ${uniqueUsername}`).should('be.visible');

        // ===== ШАГ 3: Создание новости =====
        cy.get('[data-testid="add-news-button"]').click();
        cy.get('[data-testid="news-title"]').type(newsTitle);
        cy.get('[data-testid="news-content"]').type(newsContent);
```

```
        cy.get('[data-testid="submit-news"]').click();

        cy.contains(newsTitle).should('be.visible');
        cy.contains(newsContent.substring(0, 50)).should('be.visible');

        // ===== ШАГ 4: Просмотр деталей новости =====
        cy.contains(newsTitle).click();
        cy.url().should('include', '/news/');
        cy.get('[data-testid="news-title"]').should('contain', newsTitle);
        cy.get('[data-testid="news-content"]').should('contain', newsContent);

        // ===== ШАГ 5: Возврат на главную =====
        cy.get('[data-testid="back-button"]').click();
        cy.url().should('include', '/');

        // ===== ШАГ 6: Удаление новости =====
        cy.get('[data-testid="delete-news-button"]').first().click();
        cy.get('[data-testid="confirm-delete"]').click();

        cy.contains(newsTitle).should('not.exist');
        cy.get('[data-testid="success-toast"]').should('contain', 'Новость
удалена');

        // ===== ШАГ 7: Выход из системы =====
        cy.get('[data-testid="user-menu"]').click();
        cy.get('[data-testid="logout-button"]').click();

        cy.url().should('include', '/login');
        cy.get('[data-testid="login-button"]').should('be.visible');
```

```
  });
});
```

Шаг 8. Отладка тестов

Файл cypress/e2e/debug.cy.js (примеры отладки):

```
describe('Отладка тестов', () => {

  it('Пример использования pause()', () => {
    cy.visit('/');

    // Тест остановится здесь, можно инспектировать DOM
    cy.pause();

    cy.get('[data-testid="add-news-button"]').click();
  });

  it('Пример использования debug()', () => {
    cy.visit('/');

    cy.get('[data-testid="add-news-button"]').debug().click();
  });

  it('Пример логирования', () => {
    cy.visit('/');

    cy.log('Начало теста');

    cy.get('[data-testid="add-news-button"]').then(($btn) => {
      cy.log(`Кнопка найдена: ${$btn.text()}`);
    });
```

```
    cy.wrap('Дополнительная информация').then((msg) => {
      cy.log(msg);
    });
  });

  it('Пример проверки видимости и состояния элементов', () => {
    cy.visit('/');

    // Проверка, что элемент существует
    cy.get('[data-testid="news-list"]').should('exist');

    // Проверка, что элемент видим
    cy.get('[data-testid="news-list"]').should('be.visible');

    // Проверка, что кнопка не disabled
    cy.get('[data-testid="add-news-button"]').should('not.be.disabled');

    // Проверка атрибута
    cy.get('[data-testid="add-news-button"]').should('have.attr', 'type',
'button');

    // Проверка CSS-свойства
    cy.get('[data-testid="add-news-button"]').should('have.css', 'background-
color', 'rgb(0, 123, 255)');
  });

  it('Пример работы с несколькими вкладками', () => {
    cy.visit('/');
```

```
    cy.window().then((win) => {
      cy.stub(win, 'open').as('windowOpen');
    });

    cy.get('[data-testid="external-link"]').click();
    cy.get('@windowOpen').should('be.called');
  });
});
```

Шаг 9. Запуск тестов

```
# Запуск в интерактивном режиме (откроется окно Cypress)
npx cypress open

# Запуск всех тестов в headless-режиме
npx cypress run

# Запуск конкретного теста
npx cypress run --spec "cypress/e2e/auth.cy.js"

# Запуск с определённым браузером
npx cypress run --browser chrome
npx cypress run --browser firefox
npx cypress run --browser edge

# Запуск в режиме отладки
npx cypress run --headed --no-exit

# Генерация отчёта о покрытии
npx cypress run --env coverage=true
```

4. Лучшие практики Cypress

Практика	Плохо	Хорошо
Селекторы	cy.get('.btn-primary')	cy.get('[data-testid="button"]')
Ожидания	cy.wait(3000)	cy.get('.loading').should('t')
Зависимости	Тесты зависят от порядка	Каждый тест независим
Утверждения	Одно утверждение на тест	Несколько св утверждений
Очистка	Тесты оставляют данные	afterEach очищает данные

5. Индивидуальные задания

5.1. Базовый уровень (15 вариантов)

Требования: Написать тесты авторизации (логин, регистрация, выход).

Вариант	Дополнительные тесты
1	Валидация email при регистрации
2	Восстановление пароля
3	Смена пароля

Вариант	Дополнительные тесты
4	Двухфакторная аутентификация
5	Капча на форме входа
6	Блокировка после 3 неудачных попыток
7	Ограничение времени сессии
8	Вход через соцсети (мок)
9	Запоминание пользователя ("Remember me")
10	Подтверждение email
11	Ограничение по IP
12	Логирование попыток входа
13	Политика паролей
14	Очистка формы после ошибки
15	Показ/скрытие пароля

5.2. Средний уровень (15 вариантов)

Требования: Написать CRUD-тесты для управления постами/новостями.

Вариант	CRUD-операции	Дополнительно
1	Создание, чтение, обновление, удаление	Пагинация
2	Создание, чтение, обновление, удаление	Поиск
3	Создание, чтение, обновление, удаление	Фильтрация
4	Создание, чтение, обновление, удаление	Сортировка
5	Создание, чтение, обновление, удаление	Теги
6	Создание, чтение, обновление, удаление	Изображения
7	Создание, чтение, обновление, удаление	Комментарии
8	Создание, чтение, обновление, удаление	Лайки
9	Создание, чтение, обновление, удаление	Избранное
10	Создание, чтение, обновление,	Черновики

Вариант	CRUD-операции	Дополнительно
	удаление	
11	Создание, чтение, обновление, удаление	Планирование публикации
12	Создание, чтение, обновление, удаление	Экспорт в PDF
13	Создание, чтение, обновление, удаление	История версий
14	Создание, чтение, обновление, удаление	Категории
15	Создание, чтение, обновление, удаление	Подписки

5.3. Продвинутый уровень (15 вариантов)

Требования: Написать комплексный E2E-сценарий с несколькими пользователями и ролями.

Вариант	Сценарий
1	Пользователь → создание новости → модератор → одобрение → публикация
2	Автор → черновик → редактор → правки →

Вариант	Сценарий
	публикация
3	Пользователь → комментарий → модератор → одобрение/отклонение
4	Клиент → заказ → оплата → менеджер → обработка → доставка
5	Студент → курс → тест → преподаватель → оценка → сертификат
6	Кандидат → резюме → HR → собеседование → оффер
7	Покупатель → корзина → промокод → заказ → отзыв
8	Пользователь → тикет → поддержка → ответ → закрытие
9	Подписчик → рассылка → переход по ссылке → регистрация
10	Организатор → мероприятие → продажа билетов → регистрация
11	Продавец → товар → покупатель → оплата → доставка → возврат
12	Пользователь → чат → агент → решение проблемы → опрос

Вариант	Сценарий
13	Преподаватель → задание → студент → сдача → проверка → оценка
14	Админ → создание роли → назначение прав → тестирование доступа
15	Пользователь → импорт данных → валидация → сохранение → экспорт

6. Контрольные вопросы

1. Что такое E2E-тестирование и чем оно отличается от unit-тестов?
2. Как установить и настроить Cypress?
3. Как написать простой тест для проверки заголовка страницы?
4. Как найти элемент на странице в Cypress?
5. Как проверить, что элемент видим на странице?
6. Как обрабатывать асинхронные операции в Cypress?
7. Как создать кастомную команду в Cypress?
8. Как использовать фикстуры для тестовых данных?
9. Как имитировать действия пользователя (клик, ввод текста)?
10. Как проверить, что произошло перенаправление на другой URL?
11. Как перехватывать и мокировать API-запросы?
12. Как тестировать загрузку файлов?
13. Как запускать тесты в разных браузерах?
14. Как интегрировать Cypress в CI/CD?
15. Как отлаживать падающие тесты (скриншоты, видео, time-travel)?

7. Требования к отчёту

Титульный лист (название работы, ФИО, группа, вариант)

Цель работы

Теоретическое обоснование (своими словами)

Выполнение практического примера (код, скриншоты)

Выполнение индивидуального задания (код, скриншоты прохождения тестов)

Ответы на контрольные вопросы (не менее 5 по своему уровню)

Вывод

8. Критерии оценивания

Критерий	Базовый (max 35)	Средний (max 50)	Продвинутый (max 65)
Установка и настройка Cypress	5	5	5
Тесты авторизации (логин, регистрация, выход)	20	10	5
CRUD-тесты (создание, редактирование, удаление)	—	20	10
Тесты основного сценария (happy path)	10	15	10
Комплексные сценарии (несколько	—	—	25

Критерий	Базовый (max 35)	Средний (max 50)	Продвинутый (max 65)
пользователей/ролей)			
Отладка и качество кода	—	—	10
Итого	35	50	65

Шкала перевода баллов в оценку:

Оценка	Базовый (35)	Средний (50)	Продвинутый (65)
5 (отлично)	32-35	46-50	59-65
4 (хорошо)	28-31	40-45	51-58
3 (удовлетворительно)	22-27	32-39	42-50
2 (неудовлетворительно)	0-21	0-31	0-41

Лабораторная работа №28

Контейнеризация Django-приложения с Docker

1. Цель работы

Освоить процесс контейнеризации Django-приложения с использованием Docker. Научиться создавать Dockerfile, настраивать

зависимости, использовать переменные окружения для конфигурации, собирать Docker-образ и запускать контейнер с Django-приложением.

2. Теоретическое обоснование

2.1. Что такое Docker и контейнеризация?

Docker - это платформа для разработки, доставки и запуска приложений в изолированных контейнерах. Контейнер - это легковесный, переносимый пакет, который включает приложение и все его зависимости.

Аналогия: Представьте, что вы переезжаете в новую квартиру. Вместо того чтобы каждый раз собирать мебель, подключать технику и настраивать интернет, вы можете взять с собой готовую комнату (контейнер) - со стенами, мебелью, розетками. При переезде вы просто ставите эту комнату в новом месте, и всё работает.

Контейнер vs Виртуальная машина:

Характеристика	Контейнер (Docker)	Виртуальная машина
Размер	Мегабайты	Гигабайты
Запуск	Секунды	Минуты
Изоляция	На уровне ОС	Полная (гипервизор)
Ресурсы	Меньше накладных расходов	Больше
Переносимость	Высокая	Средняя

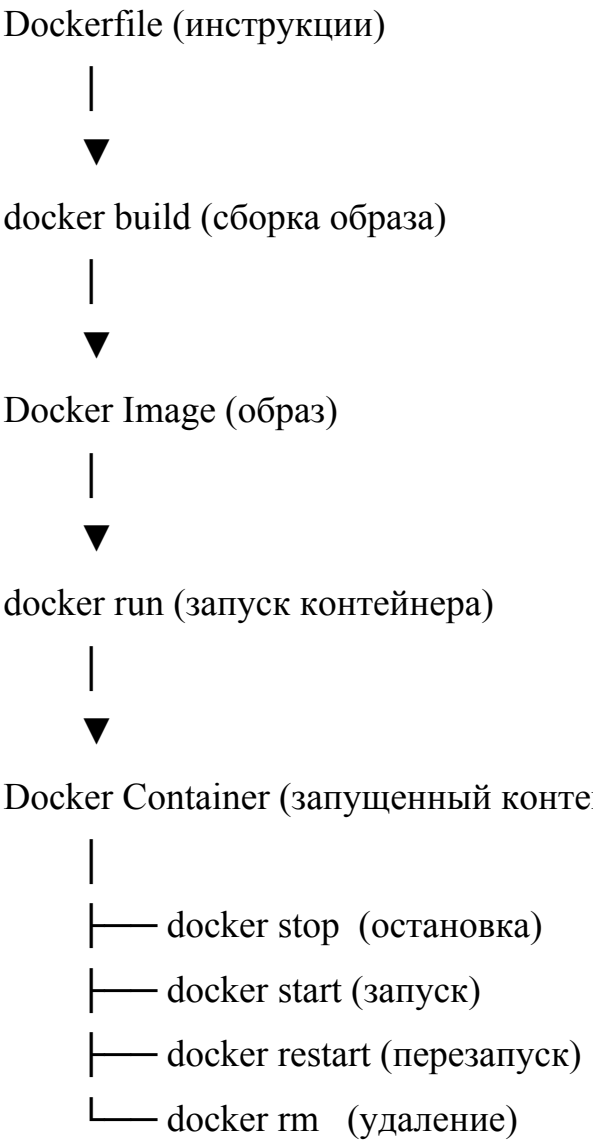
2.2. Почему Docker нужен для Django-приложений?

Проблема	Решение Docker
"У меня работает, а на сервере - нет"	Одинаковое окружение везде
Сложная настройка окружения	Всё описано в Dockerfile
Конфликты версий Python	Каждый проект в своём контейнере
Трудно масштабировать	Легко запустить несколько копий

2.3. Основные компоненты Docker

Компонент	Что это	Аналогия
Dockerfile	Инструкция по сборке образа	Рецепт блюда
Docker Image (образ)	Шаблон для создания контейнеров	Замороженная пицца
Docker Container (контейнер)	Запущенный экземпляр образа	Приготовленная пицца
Docker Registry	Хранилище образов (например, Docker Hub)	Магазин замороженных пицц

2.4. Жизненный цикл Docker-контейнера



3. Практический пример

Что мы будем делать: Создадим Dockerfile для Django-приложения, настроим переменные окружения, соберём образ и запустим контейнер.

Шаг 1. Подготовка Django-приложения

1.1. Структура проекта

```
myproject/
├── manage.py
├── requirements.txt
├── Dockerfile
├── .dockerignore
├── .env.example
├── myproject/
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
└── api/
    ├── __init__.py
    ├── models.py
    ├── views.py
    └── ...
```

1.2. Файл requirements.txt

```
Django==4.2.7
djangorestframework==3.14.0
django-cors-headers==4.3.0
psycopg2-binary==2.9.9
gunicorn==21.2.0
python-dotenv==1.0.0
```

Шаг 2. Создание Dockerfile

Файл Dockerfile:

```
# 1. Базовый образ (используем официальный образ Python)
FROM python:3.11-slim

# 2. Устанавливаем переменные окружения
ENV PYTHONDONTWRITEBYTECODE=1
ENV PYTHONUNBUFFERED=1

# 3. Устанавливаем рабочую директорию внутри контейнера
WORKDIR /app

# 4. Устанавливаем системные зависимости
RUN apt-get update && apt-get install -y \
    gcc \
    libpq-dev \
    && rm -rf /var/lib/apt/lists/*

# 5. Копируем файл с зависимостями
COPY requirements.txt .

# 6. Устанавливаем Python-зависимости
RUN pip install --no-cache-dir -r requirements.txt

# 7. Копируем код приложения в контейнер
COPY . .

# 8. Создаём пользователя для запуска приложения (без прав root)
RUN adduser --disabled-password --gecos '' appuser
RUN chown -R appuser:appuser /app
USER appuser
```

9. Открываем порт (8000 стандартный для Django)
EXPOSE 8000

10. Команда запуска приложения
CMD ["gunicorn", "--bind", "0.0.0.0:8000", "myproject.wsgi:application"]

2.1. Пояснение инструкций Dockerfile

Инструкция	Что делает	Почему важно
FROM python:3.11-slim	Базовый образ с Python	Начинаем с чистой основы
ENV PYTHONDONTWRITEBYTECODE=1	Отключает создание .pyc файлов	Экономит место
ENV PYTHONUNBUFFERED=1	Отключает буферизацию вывода	Логи видны сразу
WORKDIR /app	Создаёт папку /app и переходит в неё	Все команды будут выполняться там
COPY requirements.txt .	Копирует файл из проекта в контейнер	Кеширование зависимостей
RUN pip install ...	Устанавливает зависимости	Слой кешируется

Инструкция	Что делает	Почему важно
COPY . .	Копирует весь код	Последний слой (меняется часто)
USER appuser	Переключается на непривилегированного пользователя	Безопасность
EXPOSE 8000	Указывает, какой порт слушает приложение	Документация
CMD [...]	Команда запуска	Что выполнить при старте

Шаг 3. Создание .dockerignore

Файл .dockerignore:

```
# Виртуальное окружение
venv/
env/
.venv/

# Файлы IDE
.vscode/
.idea/
*.pyc
__pycache__/
```

```
# Git
.git/
.gitignore

# Базы данных
*.db
*.sqlite3

# Логи
*.log

# Переменные окружения
.env
.env.local

# Тесты
tests/
coverage/
htmlcov/

# Docker
Dockerfile
.dockerignore
docker-compose.yml

# Документация
README.md
docs/
```

Шаг 4. Настройка переменных окружения

Файл .env.example (шаблон для переменных):

```
# Django
DJANGO_SECRET_KEY=django-insecure-key-change-in-production
DJANGO_DEBUG=True
DJANGO_ALLOWED_HOSTS=localhost,127.0.0.1

# База данных
DB_NAME=myproject_db
DB_USER=myproject_user
DB_PASSWORD=myproject_password
DB_HOST=db
DB_PORT=5432

# Настройки приложения
API_TITLE=MyProject API
API_VERSION=v1

# Настройки email
EMAIL_HOST=smtp.gmail.com
EMAIL_PORT=587
EMAIL_USE_TLS=True
EMAIL_HOST_USER=your-email@gmail.com
EMAIL_HOST_PASSWORD=your-password
```

Шаг 5. Настройка settings.py для использования переменных окружения

Файл myproject/settings.py (фрагменты):

```
import os
from pathlib import Path
from dotenv import load_dotenv

# Загружаем переменные окружения из .env файла
load_dotenv()

BASE_DIR = Path(__file__).resolve().parent.parent

# Безопасность
SECRET_KEY = os.getenv('DJANGO_SECRET_KEY', 'django-insecure-default-key')
DEBUG = os.getenv('DJANGO_DEBUG', 'True') == 'True'

ALLOWED_HOSTS = os.getenv('DJANGO_ALLOWED_HOSTS', 'localhost,127.0.0.1').split(',')

# Приложения
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    # Third-party
    'rest_framework',
    'corsheaders',
```

```
    # Local
    'api',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'corsheaders.middleware.CorsMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'myproject.urls'

# Шаблоны
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
```

```
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'myproject.wsgi.application'

# База данных (из переменных окружения)
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': os.getenv('DB_NAME', 'myproject_db'),
        'USER': os.getenv('DB_USER', 'myproject_user'),
        'PASSWORD': os.getenv('DB_PASSWORD', ''),
        'HOST': os.getenv('DB_HOST', 'localhost'),
        'PORT': os.getenv('DB_PORT', '5432'),
    }
}

# Валидация паролей
AUTH_PASSWORD_VALIDATORS = [
    {'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator'},
    {'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator'},
    {'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator'},
```

```
        {'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator'},
    ]

    # Локализация
    LANGUAGE_CODE = 'ru-ru'
    TIME_ZONE = 'Europe/Moscow'
    USE_I18N = True
    USE_TZ = True

    # Статические файлы
    STATIC_URL = 'static/'
    STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')

    # Медиа файлы
    MEDIA_URL = '/media/'
    MEDIA_ROOT = os.path.join(BASE_DIR, 'media')

    # Настройки CORS
    CORS_ALLOWED_ORIGINS = [
        "http://localhost:3000",
        "http://localhost:5173",
    ]

    # Настройки REST Framework
    REST_FRAMEWORK = {
        'DEFAULT_PERMISSION_CLASSES': [
            'rest_framework.permissions.AllowAny',
        ],
```

```
    'DEFAULT_PAGINATION_CLASS':
'rest_framework.pagination.PageNumberPagination',
    'PAGE_SIZE': 10,
}

# Первичный ключ по умолчанию
DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

Шаг 6. Создание скрипта для запуска (entrypoint.sh)

Файл entrypoint.sh:

```
#!/bin/sh

# Ожидание запуска базы данных (если используем PostgreSQL)
# echo "Waiting for database..."
# while ! nc -z $DB_HOST $DB_PORT; do
#   sleep 0.1
# done
# echo "Database started"

# Применение миграций
echo "Applying database migrations..."
python manage.py migrate --noinput

# Сбор статических файлов
echo "Collecting static files..."
python manage.py collectstatic --noinput

# Создание суперпользователя (если не существует)
echo "Creating superuser..."
```

```
python manage.py shell << END
from django.contrib.auth import get_user_model
User = get_user_model()
if not User.objects.filter(username='admin').exists():
    User.objects.create_superuser('admin', 'admin@example.com', 'admin123')
END

# Запуск Gunicorn
echo "Starting Gunicorn..."
exec gunicorn --bind 0.0.0.0:8000 --workers 4 --threads 2 myproject.wsgi:application
```

Добавляем в Dockerfile:

```
# ... предыдущие инструкции ...

# Копируем скрипт entrypoint и делаем его исполняемым
COPY entrypoint.sh /entrypoint.sh
RUN chmod +x /entrypoint.sh

# ... остальные инструкции ...

# Запускаем entrypoint
ENTRYPOINT ["/entrypoint.sh"]
```

Шаг 7. Сборка Docker-образа

```
# Сборка образа с тегом myproject:latest
docker build -t myproject:latest .
```

```
# Сборка с указанием платформы (для совместимости с Apple Silicon)
docker build --platform linux/amd64 -t myproject:latest .

# Сборка с отображением прогресса
docker build --progress=plain -t myproject:latest .
```

Что происходит при сборке:

```
Step 1/15 : FROM python:3.11-slim
 ---> 1d3f3b5b8c9a
Step 2/15 : WORKDIR /app
 ---> Running in 4a3f2b1c8d9e
 ---> 2e4f5a6b7c8d
Step 3/15 : COPY requirements.txt .
 ---> 8d9e0a1b2c3d
Step 4/15 : RUN pip install --no-cache-dir -r requirements.txt
 ---> 3c4d5e6f7a8b
...
Successfully built 9a8b7c6d5e4f
Successfully tagged myproject:latest
```

Шаг 8. Просмотр образов

```
# Список всех образов
docker images

# Вывод:
# REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
# myproject     latest    9a8b7c6d5e4f   2 minutes ago    350MB
# python        3.11-slim 1d3f3b5b8c9a   2 weeks ago      120MB
```

Шаг 9. Запуск контейнера

```
# Базовый запуск
docker run -d -p 8000:8000 --name myproject-app myproject:latest

# Запуск с переменными окружения
docker run -d \
  -p 8000:8000 \
  --name myproject-app \
  -e DJANGO_SECRET_KEY="my-secret-key" \
  -e DJANGO_DEBUG=False \
  -e DJANGO_ALLOWED_HOSTS=localhost,example.com \
  myproject:latest

# Запуск с файлом переменных окружения
docker run -d \
  -p 8000:8000 \
  --name myproject-app \
  --env-file .env \
  myproject:latest

# Запуск с привязкой локальной папки (для разработки)
docker run -d \
  -p 8000:8000 \
  --name myproject-app \
  -v $(pwd):/app \
  myproject:latest
```

Шаг 10. Управление контейнером

```
# Просмотр запущенных контейнеров
docker ps

# Просмотр всех контейнеров (включая остановленные)
docker ps -a

# Просмотр логов контейнера
docker logs myproject-app
docker logs -f myproject-app  # следить в реальном времени

# Выполнение команды внутри контейнера
docker exec -it myproject-app python manage.py shell
docker exec -it myproject-app python manage.py createsuperuser

# Копирование файлов в/из контейнера
docker cp myproject-app:/app/media ./media
docker cp ./backup.sql myproject-app:/backup.sql

# Остановка контейнера
docker stop myproject-app

# Запуск остановленного контейнера
docker start myproject-app

# Перезапуск контейнера
docker restart myproject-app

# Удаление контейнера
docker rm myproject-app
```

```
# Удаление контейнера с принудительной остановкой
docker rm -f myproject-app
```

Шаг 11. Проверка работы приложения

```
# Проверка через curl
curl http://localhost:8000/api/health/

# Должен вернуть:
# {"status": "ok", "message": "API работает"}

# Проверка в браузере
# Открыть http://localhost:8000/admin/
```

Шаг 12. Отладка проблем

```
# Проверить логи (если контейнер не запускается)
docker logs myproject-app

# Запустить контейнер в интерактивном режиме (для отладки)
docker run -it --rm myproject:latest /bin/bash

# Проверить, что порт не занят
lsof -i :8000

# Удалить все остановленные контейнеры
docker container prune

# Удалить все неиспользуемые образы
docker image prune -a
```

4. Оптимизация Docker-образа

4.1. Многоступенчатая сборка (Multi-stage build)

```
# Этап 1: сборка зависимостей
FROM python:3.11-slim AS builder

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Этап 2: финальный образ
FROM python:3.11-slim

WORKDIR /app

# Копируем только установленные пакеты
COPY --from=builder /usr/local/lib/python3.11/site-packages /usr/local/lib/python3.11/site-packages
COPY --from=builder /usr/local/bin /usr/local/bin

COPY . .

CMD ["gunicorn", "--bind", "0.0.0.0:8000", "myproject.wsgi:application"]
```

4.2. Использование alpine (более лёгкий образ)

```
FROM python:3.11-alpine
```

```
# Устанавливаем системные зависимости Alpine
RUN apk add --no-cache \
    gcc \
    musl-dev \
    postgresql-dev \
    linux-headers

# ... остальные инструкции
```

4.3. Оптимизация слоёв

```
# ПЛОХО: каждый RUN создаёт новый слой
RUN apt-get update
RUN apt-get install -y gcc
RUN apt-get install -y libpq-dev
RUN rm -rf /var/lib/apt/lists/*

# ХОРОШО: объединяем команды
RUN apt-get update && apt-get install -y \
    gcc \
    libpq-dev \
    && rm -rf /var/lib/apt/lists/*
```

5. Индивидуальные задания

5.1. Базовый уровень (15 вариантов)

Требования: Создать Dockerfile для Django-приложения, собрать образ и запустить контейнер.

Вариант	Дополнительная настройка
1	Добавить переменные окружения для базы данных
2	Настроить сбор статических файлов при запуске
3	Добавить автоматическое создание суперпользователя
4	Настроить Gunicorn с 4 воркерами
5	Добавить Healthcheck в Dockerfile
6	Настроить логирование в stdout/stderr
7	Добавить обработку сигналов (graceful shutdown)
8	Настроить таймауты Gunicorn
9	Добавить ограничение памяти для контейнера
10	Настроить ограничение CPU для контейнера
11	Добавить volume для медиа-файлов
12	Настроить volume для статических файлов
13	Добавить логирование запросов к базе данных
14	Настроить Prometheus метрики
15	Добавить поддержку HTTPS

5.2. Средний уровень (15 вариантов)

Требования: Настроить многоступенчатую сборку (multi-stage build) и оптимизировать размер образа.

Вариант	Дополнительная оптимизация
1	Использовать Alpine Linux
2	Удалить временные файлы после установки пакетов
3	Использовать --no-cache-dir для pip
4	Оптимизировать порядок слоёв для кеширования
5	Удалить системные пакеты после сборки
6	Использовать slim образ вместо slim-buster
7	Настроить сжатие слоёв
8	Использовать --link для копирования
9	Удалить документацию и заголовочные файлы
10	Использовать distroless образ
11	Настроить сборку с --platform
12	Использовать кеш Docker BuildKit
13	Настроить параллельную сборку
14	Использовать внешний кеш слоёв

Вариант	Дополнительная оптимизация
15	Настроить автоматическую очистку кеша

5.3. Продвинутый уровень (15 вариантов)

Требования: Настроить многослойную архитектуру с Docker Compose (будет в следующей лабораторной) или настроить CI/CD для автоматической сборки образов.

Вариант	Дополнительный функционал
1	Настроить сборку в GitLab CI
2	Настроить сборку в GitHub Actions
3	Настроить автоматический push в Docker Hub
4	Настроить тегирование образов (latest, git-hash)
5	Настроить сканирование уязвимостей (Trivy)
6	Настроить подпись образов (cosign)
7	Настроить автоматическое обновление (renovate)
8	Настроить мониторинг размера образа
9	Настроить кэширование слоёв в CI
10	Настроить параллельную сборку для нескольких платформ
11	Настроить SBOM (Software Bill of Materials)

Вариант	Дополнительный функционал
12	Настроить проверку лицензий
13	Настроить автоматический откат при ошибках
14	Настроить A/B тестирование образов
15	Настроить канареечный деплой

6. Контрольные вопросы

1. Что такое Docker и контейнеризация?
2. Чем отличается контейнер от виртуальной машины?
3. Что такое Dockerfile и какие инструкции в нём используются?
4. Как собрать Docker-образ из Dockerfile?
5. Как запустить контейнер из образа?

6. Что такое многоступенчатая сборка (multi-stage build) и зачем она нужна?
7. Зачем нужен файл .dockerignore?
8. Как передать переменные окружения в контейнер?
9. Как посмотреть логи запущенного контейнера?
10. Как выполнить команду внутри работающего контейнера?

11. Как оптимизировать размер Docker-образа?
12. Как настроить healthcheck в Dockerfile?
13. Как работает кеширование слоёв при сборке?
14. Как настроить сборку для разных платформ?
15. Как настроить CI/CD для автоматической сборки образов?

7. Требования к отчёту

Титульный лист (название работы, ФИО, группа, вариант)

Цель работы

Теоретическое обоснование (своими словами)

Выполнение практического примера (Dockerfile, команды, скриншоты)

Выполнение индивидуального задания (Dockerfile, скриншоты)

Ответы на контрольные вопросы (не менее 5 по своему уровню)

Вывод

8. Критерии оценивания

Критерий	Базовый (max 35)	Средний (max 50)	Продвинутый (max 65)
Создание Dockerfile	15	10	10
Настройка .dockerignore	5	5	5
Использование переменных окружения	10	10	10
Многоступенчатая сборка	—	15	10
Оптимизация размера образа	—	10	10

Критерий	Базовый (max 35)	Средний (max 50)	Продвинутый (max 65)
CI/CD автоматизация	—	—	15
Качество кода и документация	5	—	5
Итого	35	50	65

Шкала перевода баллов в оценку:

Оценка	Базовый (35)	Средний (50)	Продвинутый (65)
5 (отлично)	32-35	46-50	59-65
4 (хорошо)	28-31	40-45	51-58
3 (удовлетворительно)	22-27	32-39	42-50
2 (неудовлетворительно)	0-21	0-31	0-41

Лабораторная работа №29

Оркестрация с Docker Compose

1. Цель работы

Научиться запускать многоконтейнерные приложения с использованием Docker Compose. Освоить создание файла docker-

compose.yml, настройку сервисов (Django, PostgreSQL, Redis, Nginx), конфигурацию сетей и volumes, а также управление всеми сервисами (запуск, остановка, просмотр логов).

2. Теоретическое обоснование

2.1. Что такое Docker Compose?

Docker Compose - это инструмент для определения и запуска многоконтейнерных Docker-приложений. С помощью одного YAML-файла вы описываете все сервисы, их зависимости, сети и тома хранения данных.

Аналогия: Представьте, что вам нужно запустить несколько служб одновременно: веб-сервер (Django), базу данных (PostgreSQL), кэш (Redis) и прокси-сервер (Nginx). Вместо того чтобы запускать каждый контейнер отдельной командой (и помнить все параметры), вы пишете один "рецепт" (docker-compose.yml), и Docker Compose запускает всё одной командой.

2.2. Зачем нужна оркестрация?

Проблема	Решение Docker Compose
Много контейнеров, трудно управлять	Описываем все сервисы в одном файле
Контейнеры должны общаться друг с другом	Автоматически создаёт сеть
Данные теряются при перезапуске контейнера	Persistent volumes
Порядок запуска важен (БД → Django)	depends_on

Проблема	Решение Docker Compose
Разные переменные окружения для сервисов	env_file, environment

2.3. Сравнение: docker run vs docker-compose

docker run	docker-compose
docker run -d --name postgres -p 5432:5432 ... postgres	docker-compose up -d
docker run -d --name redis -p 6379:6379 ... redis	Все сервисы запускаются вместе
docker run -d --name django -p 8000:8000 --link postgres ... django	Автоматическая сеть
Нужно запоминать все команды	Всё описано в YAML

2.4. Основные компоненты docker-compose.yml

```yaml
version: '3.8'          # Версия формата Compose

services:               # Определение сервисов
  web:                  # Имя сервиса (Django)
    build: .            # Сборка из Dockerfile
    ports:              # Проброс портов
      - "8000:8000"
    environment:        # Переменные окружения
```

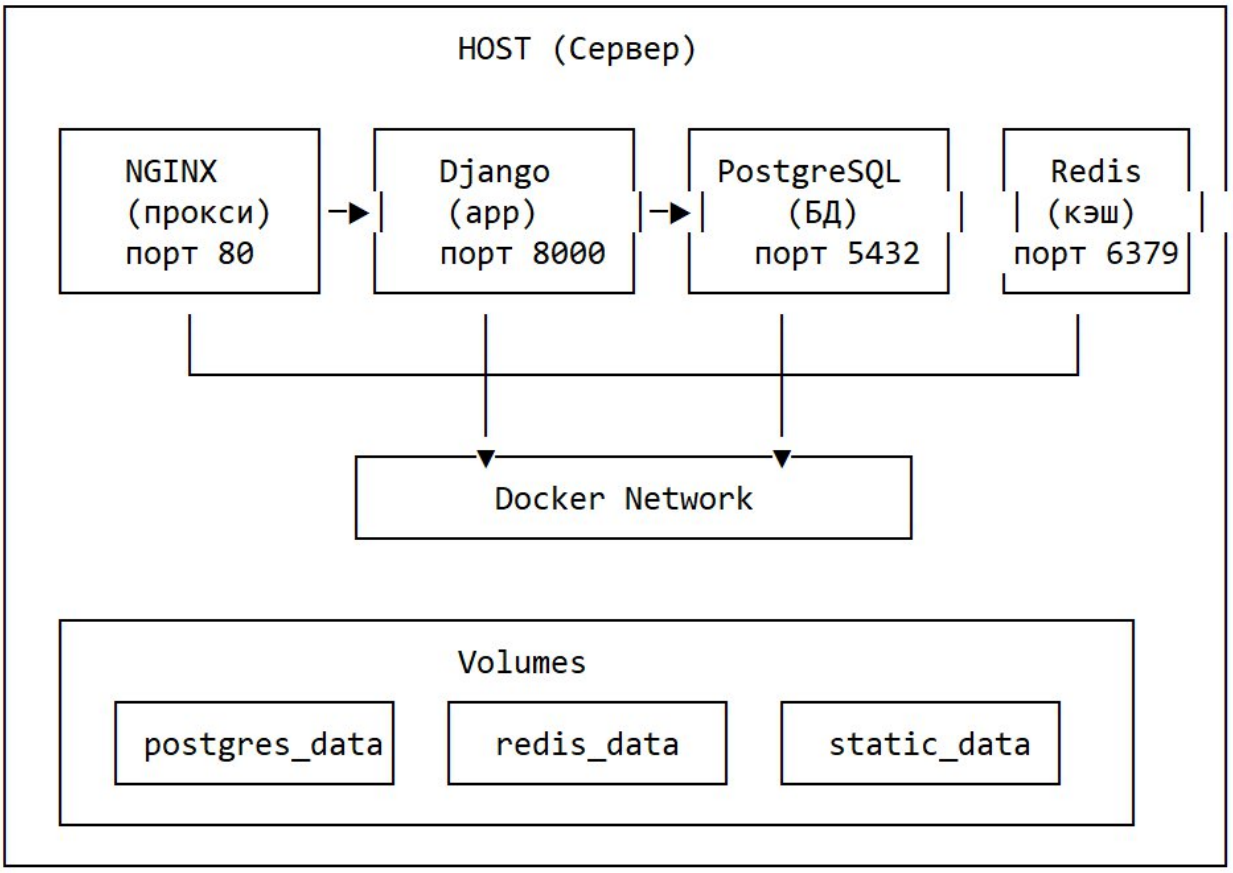
```
      - DEBUG=1
    depends_on:         # Зависимости (порядок запуска)
      - db
      - redis
    volumes:            # Тома для хранения данных
      - ./app:/app

  db:                   # Сервис базы данных
    image: postgres:15  # Готовый образ из Docker Hub
    environment:
      - POSTGRES_DB=myproject
    volumes:
      - postgres_data:/var/lib/postgresql/data

volumes:                # Определение томов
  postgres_data:        # Именованный том

networks:               # (опционально) Сети
  app-network:          # Кастомная сеть
    driver: bridge
```

2.5. Архитектура приложения с Docker Compose



3. Практический пример

Что мы будем делать: Создадим Docker Compose файл для запуска Django-приложения с PostgreSQL, Redis и Nginx.

Шаг 1. Подготовка проекта

1.1. Структура проекта

myproject/

├── docker-compose.yml

├── .env

├── backend/

│ ├── Dockerfile

│ ├── requirements.txt

│ ├── manage.py

│ ├── entrypoint.sh

│ ├── myproject/

```
│   │   ├── settings.py
│   │   └── ...
│   └── api/
│       └── ...
├── nginx/
│   ├── Dockerfile
│   └── nginx.conf
└── scripts/
    └── wait-for-it.sh
```

Шаг 2. Создание Dockerfile для Django (backend/Dockerfile)

Файл backend/Dockerfile:

```
FROM python:3.11-slim

ENV PYTHONDONTWRITEBYTECODE=1
ENV PYTHONUNBUFFERED=1

WORKDIR /app

# Установка системных зависимостей
RUN apt-get update && apt-get install -y \
    gcc \
    libpq-dev \
    netcat-openbsd \
    && rm -rf /var/lib/apt/lists/*

# Копирование зависимостей
COPY requirements.txt .
```

```
RUN pip install --no-cache-dir -r requirements.txt

# Копирование кода приложения
COPY . .

# Копирование скрипта ожидания
COPY entrypoint.sh /entrypoint.sh
RUN chmod +x /entrypoint.sh

ENTRYPOINT ["/entrypoint.sh"]
```

Файл backend/entrypoint.sh:

```
#!/bin/sh

echo "Waiting for PostgreSQL..."
while ! nc -z $DB_HOST $DB_PORT; do
  sleep 0.1
done
echo "PostgreSQL started"

echo "Waiting for Redis..."
while ! nc -z $REDIS_HOST $REDIS_PORT; do
  sleep 0.1
done
echo "Redis started"

echo "Applying database migrations..."
python manage.py migrate --noinput
```

```
echo "Collecting static files..."
python manage.py collectstatic --noinput

echo "Creating superuser..."
python manage.py shell << END
from django.contrib.auth import get_user_model
User = get_user_model()
if not User.objects.filter(username='admin').exists():
    User.objects.create_superuser('admin', 'admin@example.com', 'admin123')
END

echo "Starting Gunicorn..."
exec gunicorn --bind 0.0.0.0:8000 --workers 4 --threads 2 myproject.wsgi:application
```

Шаг 3. Настройка Nginx

Файл nginx/Dockerfile:

```
FROM nginx:alpine

# Удаляем конфигурацию по умолчанию
RUN rm /etc/nginx/conf.d/default.conf

# Копируем нашу конфигурацию
COPY nginx.conf /etc/nginx/conf.d/
```

Файл nginx/nginx.conf:

```
upstream django {
```

```
    server backend:8000;
}

server {
    listen 80;
    server_name localhost;
    client_max_body_size 100M;

    # Статические файлы
    location /static/ {
        alias /static/;
    }

    # Медиа файлы
    location /media/ {
        alias /media/;
    }

    # Проксирование на Django
    location / {
        proxy_pass http://django;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;

        # Таймауты для долгих запросов
        proxy_connect_timeout 75s;
        proxy_send_timeout 300s;
```

```
        proxy_read_timeout 300s;
    }

    # Healthcheck для Docker
    location /health/ {
        return 200 "healthy\n";
        add_header Content-Type text/plain;
    }
}
```

Шаг 4. Настройка переменных окружения

Файл .env:

```
# Django
DJANGO_SECRET_KEY=your-secret-key-change-in-production
DJANGO_DEBUG=False
DJANGO_ALLOWED_HOSTS=localhost,127.0.0.1

# База данных
DB_NAME=myproject_db
DB_USER=myproject_user
DB_PASSWORD=myproject_password
DB_HOST=postgres
DB_PORT=5432

# Redis
REDIS_HOST=redis
REDIS_PORT=6379

# Настройки Nginx
```

```
NGINX_HOST=localhost
```

Шаг 5. Настройка settings.py для Docker

Файл backend/myproject/settings.py (фрагменты):

```
import os
from pathlib import Path

BASE_DIR = Path(__file__).resolve().parent.parent

SECRET_KEY = os.getenv('DJANGO_SECRET_KEY', 'django-insecure-default')

DEBUG = os.getenv('DJANGO_DEBUG', 'True') == 'True'

ALLOWED_HOSTS = os.getenv('DJANGO_ALLOWED_HOSTS', 'localhost,127.0.0.1').split(',')

# База данных (из переменных окружения)
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': os.getenv('DB_NAME', 'myproject_db'),
        'USER': os.getenv('DB_USER', 'myproject_user'),
        'PASSWORD': os.getenv('DB_PASSWORD', ''),
        'HOST': os.getenv('DB_HOST', 'localhost'),
        'PORT': os.getenv('DB_PORT', '5432'),
    }
}
```

```
# Redis (для кэша и сессий)
CACHES = {
    'default': {
        'BACKEND': 'django_redis.cache.RedisCache',
        'LOCATION': f'redis://{os.getenv("REDIS_HOST", "localhost")}:{os.getenv("REDIS_PORT", 6379)}/1',
        'OPTIONS': {
            'CLIENT_CLASS': 'django_redis.client.DefaultClient',
        }
    }
}

# Сессии в Redis
SESSION_ENGINE = 'django.contrib.sessions.backends.cache'
SESSION_CACHE_ALIAS = 'default'

# Статические файлы
STATIC_URL = '/static/'
STATIC_ROOT = '/static/'

# Медиа файлы
MEDIA_URL = '/media/'
MEDIA_ROOT = '/media/'

# Настройки CORS (для React)
CORS_ALLOWED_ORIGINS = [
    "http://localhost:3000",
    "http://localhost:5173",
    "http://localhost:80",
```

]

Шаг 6. Создание docker-compose.yml

Файл docker-compose.yml:

```
version: '3.8'

services:
  # 1. Сервис базы данных PostgreSQL
  postgres:
    image: postgres:15-alpine
    container_name: myproject-postgres
    restart: unless-stopped
    environment:
      POSTGRES_DB: ${DB_NAME:-myproject_db}
      POSTGRES_USER: ${DB_USER:-myproject_user}
      POSTGRES_PASSWORD: ${DB_PASSWORD:-myproject_password}
      PGDATA: /var/lib/postgresql/data/pgdata
    volumes:
      - postgres_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U ${DB_USER:-myproject_user} -d
${DB_NAME:-myproject_db}"]
      interval: 10s
      timeout: 5s
      retries: 5
    networks:
      - app-network
```

```
    # 2. Сервис Redis (кэш и брокер сообщений)
    redis:
      image: redis:7-alpine
      container_name: myproject-redis
      restart: unless-stopped
      command: redis-server --appendonly yes
      volumes:
        - redis_data:/data
      ports:
        - "6379:6379"
      healthcheck:
        test: ["CMD", "redis-cli", "ping"]
        interval: 10s
        timeout: 5s
        retries: 5
      networks:
        - app-network

    # 3. Основной сервис Django
    backend:
      build: ./backend
      container_name: myproject-backend
      restart: unless-stopped
      environment:
        DJANGO_SECRET_KEY: ${DJANGO_SECRET_KEY}
        DJANGO_DEBUG: ${DJANGO_DEBUG:-False}
        DJANGO_ALLOWED_HOSTS:      ${DJANGO_ALLOWED_HOSTS:-
localhost}
```

```
    DB_NAME: ${DB_NAME:-myproject_db}
    DB_USER: ${DB_USER:-myproject_user}
    DB_PASSWORD: ${DB_PASSWORD:-myproject_password}
    DB_HOST: postgres
    DB_PORT: ${DB_PORT:-5432}
    REDIS_HOST: redis
    REDIS_PORT: ${REDIS_PORT:-6379}
  volumes:
    - ./backend:/app
    - static_volume:/static
    - media_volume:/media
  depends_on:
    postgres:
      condition: service_healthy
    redis:
      condition: service_healthy
  networks:
    - app_network

# 4. Сервис Nginx (веб-сервер и прокси)
nginx:
  build: ./nginx
  container_name: myproject_nginx
  restart: unless-stopped
  ports:
    - "80:80"
    - "443:443"  # HTTPS (если настроен SSL)
  volumes:
    - static_volume:/static
```

```
      - media_volume:/media
      - ./nginx/nginx.conf:/etc/nginx/conf.d/nginx.conf
    depends_on:
      - backend
    networks:
      - app_network

# 5. Определение томов (volumes) для сохранения данных
volumes:
  postgres_data:
    name: myproject_postgres_data
  redis_data:
    name: myproject_redis_data
  static_volume:
    name: myproject_static_volume
  media_volume:
    name: myproject_media_volume

# 6. Определение сети
networks:
  app-network:
    name: myproject_app_network
    driver: bridge
```

Шаг 7. Запуск сервисов

bash

```
# Запуск всех сервисов в фоновом режиме
docker-compose up -d

# Запуск с пересборкой образов
```

```
docker-compose up -d --build

# Запуск с отображением логов в реальном времени
docker-compose up

# Запуск только определённых сервисов
docker-compose up -d postgres redis
```

Шаг 8. Управление сервисами

```
# Просмотр запущенных сервисов
docker-compose ps

# Вывод:
# NAME                  STATE     PORTS
# myproject-postgres    Up        0.0.0.0:5432->5432/tcp
# myproject-redis       Up        0.0.0.0:6379->6379/tcp
# myproject-backend     Up        0.0.0.0:8000->8000/tcp
# myproject-nginx       Up        0.0.0.0:80->80/tcp

# Просмотр логов всех сервисов
docker-compose logs

# Просмотр логов конкретного сервиса
docker-compose logs backend
docker-compose logs postgres
docker-compose logs nginx

# Просмотр логов в реальном времени
docker-compose logs -f
```

```
docker-compose logs -f backend redis

# Выполнение команды в контейнере
docker-compose exec backend python manage.py shell
docker-compose exec backend python manage.py createsuperuser
docker-compose exec postgres psql -U myproject_user -d myproject_db

# Копирование файлов
docker-compose cp backend:/app/media ./backup/
docker-compose cp ./backup.sql postgres:/backup.sql

# Остановка всех сервисов
docker-compose down

# Остановка с удалением томов (потеря данных!)
docker-compose down -v

# Перезапуск сервисов
docker-compose restart
docker-compose restart backend
```

Шаг 9. Полезные команды Docker Compose

```
# Просмотр используемых портов
docker-compose port backend 8000
docker-compose port postgres 5432

# Просмотр используемых томов
docker volume ls | grep myproject
```

```
# Просмотр используемых сетей
docker network ls | grep myproject

# Выполнение миграций вручную
docker-compose run --rm backend python manage.py makemigrations
docker-compose run --rm backend python manage.py migrate

# Создание дампа базы данных
docker-compose exec postgres pg_dump -U myproject_user myproject_db > backup.sql

# Восстановление базы данных из дампа
cat backup.sql | docker-compose exec -T postgres psql -U myproject_user myproject_db

# Очистка неиспользуемых контейнеров, сетей и образов
docker-compose down --rmi all --volumes --remove-orphans
```

Шаг 10. Проверка работоспособности

```
# Проверка через curl
curl http://localhost/health/      # Healthcheck Nginx
curl http://localhost/api/health/  # Healthcheck Django

# Проверка через браузер
# Открыть http://localhost/admin/

# Проверка Redis
docker-compose exec redis redis-cli ping
# Должен вернуть: PONG
```

```
# Проверка PostgreSQL
docker-compose exec postgres pg_isready -U myproject_user
```

Шаг 11. Профили для разных окружений

Файл docker-compose.override.yml (переопределение для разработки):

```
version: '3.8'

services:
  backend:
    environment:
      DJANGO_DEBUG: "True"
    command: python manage.py runserver 0.0.0.0:8000
    volumes:
      - ./backend /app

  nginx:
    ports:
      - "8080:80"
```

Запуск с переопределением:

bash

```
# Автоматически использует docker-compose.yml и docker-compose.override.yml
docker-compose up -d

# Использование другого файла
docker-compose -f docker-compose.yml -f docker-compose.prod.yml up -d
```

4. Расширенная конфигурация

4.1. Добавление Celery worker

```
services:
  # ... другие сервисы ...

  celery:
    build: ./backend
    container_name: myproject-celery
    restart: unless-stopped
    command: celery -A myproject worker -l info
    environment:
      DB_NAME: ${DB_NAME:-myproject_db}
      DB_USER: ${DB_USER:-myproject_user}
      DB_PASSWORD: ${DB_PASSWORD:-myproject_password}
      DB_HOST: postgres
      REDIS_HOST: redis
      REDIS_PORT: ${REDIS_PORT:-6379}
    volumes:
      - ./backend:/app
    depends_on:
      postgres:
        condition: service_healthy
      redis:
        condition: service_healthy
    networks:
      - app-network

  celery-beat:
    build: ./backend
```

```
  container_name: myproject-celery-beat
  restart: unless-stopped
  command: celery -A myproject beat -l info
  environment:
    REDIS_HOST: redis
    REDIS_PORT: ${REDIS_PORT:-6379}
  volumes:
    - ./backend:/app
  depends_on:
    redis:
      condition: service_healthy
  networks:
    - app-network
```

4.2. Добавление Flower (мониторинг Celery)

```
services:
  flower:
    image: mher/flower
    container_name: myproject-flower
    restart: unless-stopped
    command: celery --broker=redis://redis:6379/0 flower --port=5555
    ports:
      - "5555:5555"
    depends_on:
      - redis
    networks:
      - app-network
```

4.3. Добавление PgAdmin

```
services:
  pgadmin:
    image: dpage/pgadmin4
    container_name: myproject_pgadmin
    restart: unless-stopped
    environment:
      PGADMIN_DEFAULT_EMAIL: admin@example.com
      PGADMIN_DEFAULT_PASSWORD: admin123
    ports:
      - "5050:80"
    depends_on:
      - postgres
    networks:
      - app_network
```

5. Индивидуальные задания

5.1. Базовый уровень (15 вариантов)

Требования: Создать docker-compose.yml с сервисами Django, PostgreSQL и Redis.

Вариант	Дополнительный сервис
1	Nginx
2	PgAdmin
3	phpMyAdmin (для MySQL)

Вариант	Дополнительный сервис
4	Adminer
5	Redis Commander
6	Mongo Express (для MongoDB)
7	Portainer
8	Traefik
9	Caddy
10	Apache
11	Lighttpd
12	HAProxy
13	Varnish
14	Memcached
15	Elasticsearch

5.2. Средний уровень (15 вариантов)

Требования: Настроить volumes для постоянного хранения данных и healthcheck для сервисов.

Вариант	Дополнительная настройка
1	Healthcheck для всех сервисов

Вариант	Дополнительная настройка
2	Именованные volumes для статики и медиа
3	Переменные окружения через .env файл
4	depends_on с condition: service_healthy
5	Профили для dev/prod
6	Docker Compose override для разработки
7	Лимиты ресурсов (CPU, память)
8	Restart policies
9	Логирование в файл
10	Кастомная сеть с указанием подсети
11	Использование secrets
12	Настройка tmpfs для временных файлов
13	Кастомный entrypoint
14	Init контейнер для миграций
15	Sidecar контейнер для бэкапов

5.3. Продвинутый уровень (15 вариантов)

Требования: Добавить сервисы Celery, Flower, мониторинг (Prometheus + Grafana).

Вариант	Дополнительные сервисы
1	Celery worker + Celery beat
2	Flower (мониторинг Celery)
3	Prometheus + Grafana
4	Loki + Promtail
5	Elasticsearch + Kibana
6	Jaeger (трассировка)
7	Sentry
8	MinIO (S3-совместимое хранилище)
9	MailHog (тестирование email)
10	RabbitMQ
11	Kafka
12	ZooKeeper
13	ClickHouse
14	TimescaleDB
15	Keycloak

6. Контрольные вопросы

1. Что такое Docker Compose и зачем он нужен?
2. Какая структура файла docker-compose.yml?
3. Как запустить все сервисы, описанные в docker-compose.yml?
4. Как остановить все сервисы?
5. Что такое depends_on и зачем он нужен?
6. В чём разница между depends_on и condition: service_healthy?
7. Что такое volumes в Docker Compose и зачем они нужны?
8. Как передать переменные окружения в сервисы?
9. Как посмотреть логи конкретного сервиса?
10. Как выполнить команду внутри контейнера?
11. Как настроить разные профили для разработки и продакшена?
12. Как использовать docker-compose.override.yml?
13. Как настроить healthcheck для кастомного сервиса?
14. Как настроить лимиты ресурсов для сервисов?
15. Как настроить бэкап томов с данными?

7. Требования к отчёту

Титульный лист (название работы, ФИО, группа, вариант)

Цель работы

Теоретическое обоснование (своими словами)

Выполнение практического примера (docker-compose.yml, команды, скриншоты)

Выполнение индивидуального задания (docker-compose.yml, скриншоты)

Ответы на контрольные вопросы (не менее 5 по своему уровню)

Вывод

8. Критерии оценивания

Критерий	Базовый (max 35)	Средний (max 50)	Продвинутый (max 65)
Создание docker-compose.yml	15	10	10
Настройка сервисов (Django, Postgres, Redis)	15	10	10
Настройка Nginx	—	15	10
Volumes и healthcheck	—	10	10
Celery, Flower, мониторинг	—	—	15
Качество кода и документация	5	5	10
Итого	35	50	65

Шкала перевода баллов в оценку:

Оценка	Базовый (35)	Средний (50)	Продвинутый (65)
5 (отлично)	32-35	46-50	59-65
4 (хорошо)	28-31	40-45	51-58

Оценка	Базовый (35)	Средний (50)	Продвинутый (65)
3 (удовлетворительно)	22-27	32-39	42-50
2 (неудовлетворительно)	0-21	0-31	0-41

Лабораторная работа №30

Настройка Nginx и Gunicorn для продакшна

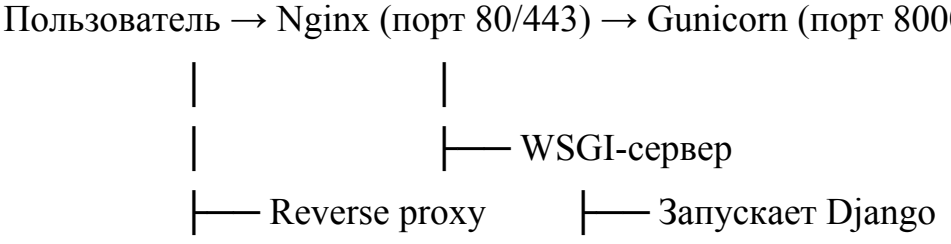
1. Цель работы

Освоить production-развертывание Django-приложений с использованием Gunicorn в качестве WSGI-сервера и Nginx в качестве reverse proxy. Научиться настраивать раздачу статических и медиа-файлов, а также подключать SSL-сертификаты через Let's Encrypt для обеспечения безопасного HTTPS-соединения.

2. Теоретическое обоснование

2.1. Архитектура production-развертывания Django

В production-среде Django не используется встроенный сервер разработки (runserver), так как он небезопасен и неэффективен. Вместо этого используется связка:



├── Раздача статики └── Управляет процессами
└── SSL termination

Аналогия: Представьте ресторан:

Nginx - это администратор на входе. Он встречает гостей (пользователей), принимает заказы, отправляет их на кухню и подаёт готовые блюда.

Gunicorn - это повара на кухне. Они готовят блюда (обрабатывают запросы), но не общаются напрямую с гостями.

Django - это рецепты и продукты.

2.2. Сравнение веб-серверов и WSGI-серверов

Компонент	Назначение	Примеры
Web Server	Принимает HTTP-запросы, отдаёт статику, проксирует динамические запросы	Nginx, Apache, Caddy
WSGI Server	Запускает Python-приложение, управляет процессами	Gunicorn, uWSGI, Daphne
Application	Бизнес-логика	Django, Flask, FastAPI

2.3. Почему Gunicorn?

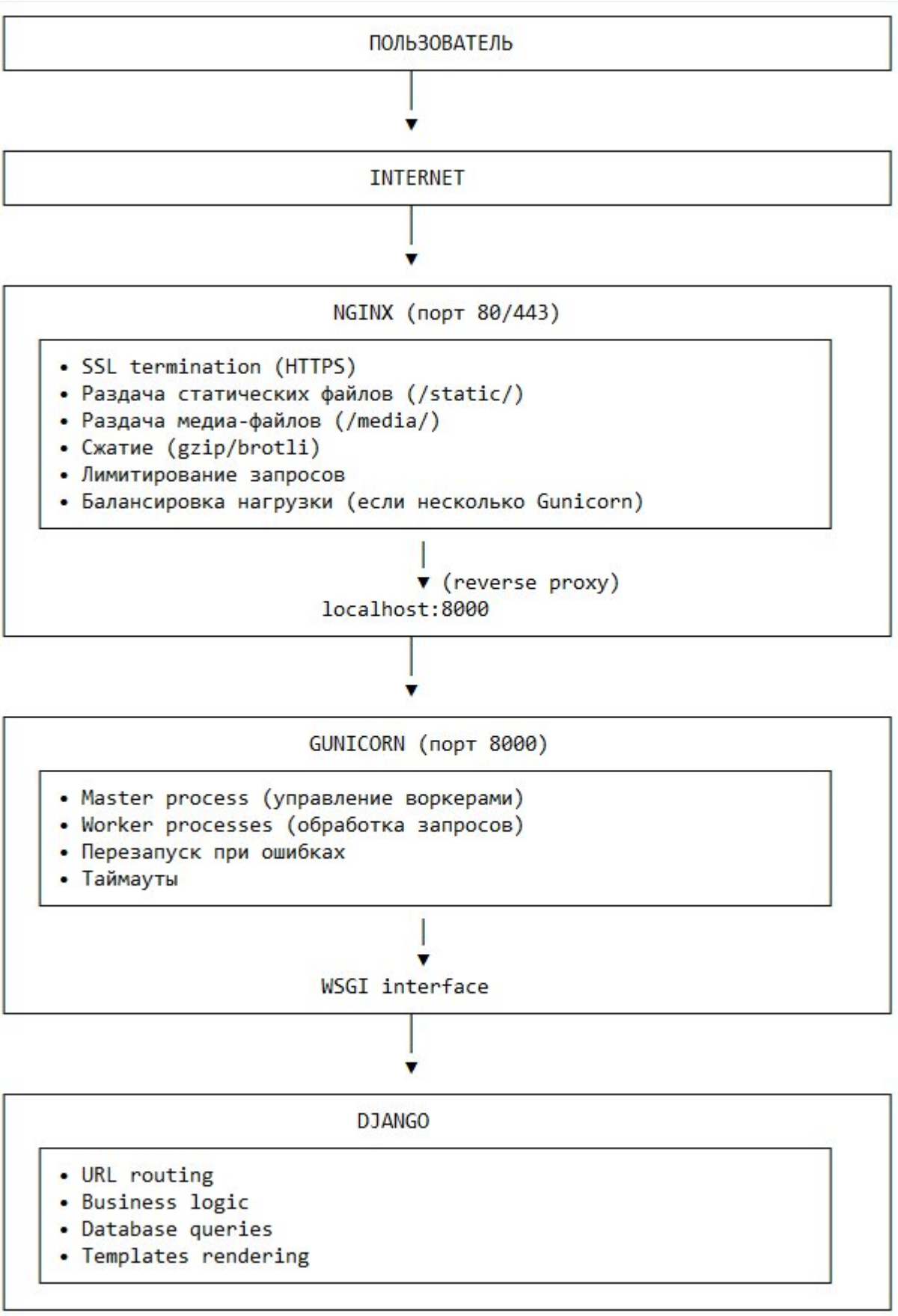
Характеристика	Gunicorn	uWSGI	Daphne
Сложность настройки	Простая	Сложная	Средняя
Производительность	Высокая	Очень высокая	Высокая
Поддержка ASGI	✕ Нет (только WSGI)	✕ Нет	✓ Да
Документация	Отличная	Хорошая	Хорошая
Рекомендация	Для большинства проектов	Для высоконагруженных	Для WebSocket

2.4. Зачем нужен Nginx перед Gunicorn?

Задача	**Почему не Gunicorn**	**Почему Nginx**
Раздача статики	Gunicorn плохо отдаёт файлы	Nginx оптимизирован для статики
Сжатие	Не поддерживает	Поддерживает gzip/brotli

Задача	Почему не Gunicorn	Почему Nginx
SSL	Не поддерживает	Поддерживает (Let's Encrypt)
Балансировка	Ограниченно	Отлично
Ограничение запросов	Нет	Да
Кэширование	Нет	Да

2.5. Полная архитектура



3. Практический пример

Что мы будем делать: Настроим production-окружение для Django-приложения с использованием Gunicorn и Nginx.

Шаг 1. Подготовка Django-приложения

1.1. Настройка settings.py для production

Файл myproject/settings.py (фрагменты):

```
import os
from pathlib import Path

BASE_DIR = Path(__file__) resolve() parent parent

# Безопасность
SECRET_KEY = os environ get('DJANGO_SECRET_KEY')
DEBUG = os environ get('DJANGO_DEBUG', 'False') == 'True'

ALLOWED_HOSTS = os environ get('DJANGO_ALLOWED_HOSTS', 'localhost,127.0.0.1') split(',')

# Security headers
SECURE_BROWSER_XSS_FILTER = True
SECURE_CONTENT_TYPE_NOSNIFF = True
X_FRAME_OPTIONS = 'DENY'

# HTTPS
if not DEBUG
    SECURE_SSL_REDIRECT = True
    SESSION_COOKIE_SECURE = True
    CSRF_COOKIE_SECURE = True
```

```
    SECURE_HSTS_SECONDS = 31536000  # 1 год
    SECURE_HSTS_INCLUDE_SUBDOMAINS = True
    SECURE_HSTS_PRELOAD = True

# Статические файлы
STATIC_URL = '/static/'
STATIC_ROOT = '/var/www/myproject/static/'
STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static')]

# Медиа файлы (загружаемые пользователями)
MEDIA_URL = '/media/'
MEDIA_ROOT = '/var/www/myproject/media/'

# Логирование
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'formatters': {
        'verbose': {
            'format': '{levelname} {asctime} {module} {process:d} {thread:d} {message}',
            'style': '{',
        },
        'simple': {
            'format': '{levelname} {message}',
            'style': '{',
        },
    },
    'handlers': {
```

```
        'file': {
            'level': 'INFO',
            'class': 'logging.handlers.RotatingFileHandler',
            'filename': '/var/log/myproject/django.log',
            'maxBytes': 1024 * 1024 * 10,  # 10 MB
            'backupCount': 5,
            'formatter': 'verbose',
        },
        'console': {
            'level': 'INFO',
            'class': 'logging.StreamHandler',
            'formatter': 'simple',
        },
    },
    'root': {
        'handlers': ['console', 'file'],
        'level': 'INFO',
    },
}
```

Шаг 2. Установка и настройка Gunicorn

2.1. Установка Gunicorn

```
# В виртуальном окружении
pip install gunicorn

# Добавление в requirements.txt
echo "gunicorn==21.2.0" >> requirements.txt
```

2.2. Создание конфигурации Gunicorn

Файл gunicorn.conf.py (в корне проекта):

```
"""
Gunicorn configuration file
"""

import multiprocessing
import os

# Базовые настройки
bind = '127.0.0.1:8000'          # Адрес и порт
backlog = 2048                   # Размер очереди ожидания

# Настройки воркеров
workers = multiprocessing.cpu_count() * 2 + 1  # Формула: (2 * CPU) + 1
worker_class = 'sync'            # Тип воркеров (sync, gevent, eventlet)
threads = 2                      # Количество потоков на воркер
worker_connections = 1000        # Макс. соединений на воркер

# Таймауты
timeout = 30                     # Таймаут обработки запроса (сек)
graceful_timeout = 30            # Таймаут graceful shutdown
keepalive = 5                    # Keep-alive соединения

# Настройки процессов
max_requests = 1000              # Перезапускать воркер после N запросов
max_requests_jitter = 50         # Случайное отклонение

# Логирование
```

```
    accesslog = '/var/log/myproject/gunicorn_access.log'
    errorlog = '/var/log/myproject/gunicorn_error.log'
    loglevel = 'info'
    access_log_format = '%(h)s %(l)s %(u)s %(t)s "%(r)s" %(s)s %(b)s "%(f)s" "%(a)s"'

    # Безопасность
    limit_request_line = 4094        # Макс. длина URL
    limit_request_fields = 100       # Макс. количество заголовков
    limit_request_field_size = 8190  # Макс. размер заголовка

    # Пользователь и группа (должны существовать)
    user = 'www-data'
    group = 'www-data'

    # Proc name
    proc_name = 'myproject'

    # Даем возможность перезагружать конфигурацию без остановки
    reload = False
    reload_extra_files = []

    # Настройки для работы с SSL (если Gunicorn сам обрабатывает SSL, но лучше через Nginx)
    # keyfile = '/etc/ssl/private/myproject.key'
    # certfile = '/etc/ssl/certs/myproject.crt'

    # Daemon mode (лучше использовать systemd)
    daemon = False
```

```
pidfile = '/var/run/gunicorn/myproject.pid'

# Настройки памяти
worker_tmp_dir = '/dev/shm'          # Использовать tmpfs для временных файлов

# Настройки для работы с Nginx
forwarded_allow_ips = '127.0.0.1' # Доверенные прокси
proxy_allow_ips = '127.0.0.1'     # Разрешённые прокси-серверы

# Preload application (экономит память, но усложняет перезагрузку)
preload_app = False

# Кастомные параметры (можно использовать в приложении)
def post_fork(server, worker):
    """Действия после создания воркера"""
    import logging
    logging.info(f"Worker {worker.pid} spawned")

def pre_fork(server, worker):
    """Действия перед созданием воркера"""
    pass

def on_starting(server):
    """Действия при запуске Gunicorn"""
    import logging
    logging.info("Gunicorn starting...")

def on_reload(server):
```

```
    """Действия при перезагрузке"""
    import logging
    logging.info('Gunicorn reloading...')

def worker_exit(server, worker):
    """Действия при выходе воркера"""
    import logging
    logging.info(f'Worker {worker.pid} exited')
```

2.3. Запуск Gunicorn

```
# Базовый запуск
gunicorn myproject.wsgi:application

# Запуск с конфигурационным файлом
gunicorn --config gunicorn.conf.py myproject.wsgi:application

# Запуск с привязкой к unix socket (рекомендуется)
gunicorn --bind unix:/tmp/myproject.sock myproject.wsgi:application

# Запуск с настройками в командной строке
gunicorn --workers 4 --threads 2 --bind 127.0.0.1:8000 myproject.wsgi:application

# Запуск с логами в консоль
gunicorn --log-level debug myproject.wsgi:application
```

Шаг 3. Настройка Nginx

3.1. Установка Nginx

```
# Ubuntu/Debian
sudo apt update
sudo apt install nginx

# Проверка установки
nginx -v
sudo systemctl status nginx
```

3.2. Создание конфигурации Nginx

Файл /etc/nginx/sites-available/myproject:

```
# Upstream для Gunicorn (если используется TCP)
upstream django {
    server 127.0.0.1:8000;
    # Если используется unix socket:
    # server unix:/tmp/myproject.sock;

    keepalive 32;
}

# Основной сервер (HTTP)
server {
    listen 80;
    listen [::]:80;
    server_name myproject.com www.myproject.com;

    # Перенаправление на HTTPS
    return 301 https://$server_name$request_uri;
}
```

```
    # Сервер HTTPS (после настройки SSL)
    server {
        listen 443 ssl http2;
        listen [::]:443 ssl http2;
        server_name myproject.com www.myproject.com;

        # Пути к SSL-сертификатам (будет настроено позже)
        ssl_certificate /etc/letsencrypt/live/myproject.com/fullchain.pem;
        ssl_certificate_key /etc/letsencrypt/live/myproject.com/privkey.pem;

        # Настройки SSL
        ssl_protocols TLSv1.2 TLSv1.3;
        ssl_ciphers HIGH:!aNULL:!MD5;
        ssl_prefer_server_ciphers on;
        ssl_session_cache shared:SSL:10m;
        ssl_session_timeout 10m;

        # Security headers
        add_header Strict-Transport-Security "max-age=31536000;
includeSubDomains; preload" always;
        add_header X-Frame-Options "DENY" always;
        add_header X-Content-Type-Options "nosniff" always;
        add_header X-XSS-Protection "1; mode=block" always;
        add_header Referrer-Policy "strict-origin-when-cross-origin" always;

        # Лимиты
        client_max_body_size 100M;
        client_body_timeout 60s;
        send_timeout 60s;
```

```
    # Gzip сжатие
    gzip on;
    gzip_vary on;
    gzip_proxied any;
    gzip_comp_level 6;
    gzip_types text/plain text/css text/xml text/javascript application/json application/javascript application/xml+rss application/rss+xml image/svg+xml;

    # Логи
    access_log /var/log/nginx/myproject_access.log;
    error_log /var/log/nginx/myproject_error.log;

    # Статические файлы
    location /static/ {
        alias /var/www/myproject/static/;
        expires 30d;
        add_header Cache-Control "public, immutable";
        access_log off;
    }

    # Медиа файлы (загружаемые пользователями)
    location /media/ {
        alias /var/www/myproject/media/;
        expires 7d;
        add_header Cache-Control "public";
        access_log off;
    }
```

```
    # Favicon
    location /favicon.ico {
        alias /var/www/myproject/static/favicon.ico;
        access_log off;
        log_not_found off;
    }

    # Robots.txt
    location /robots.txt {
        alias /var/www/myproject/static/robots.txt;
        access_log off;
        log_not_found off;
    }

    # Healthcheck для Docker/мониторинга
    location /health/ {
        proxy_pass http://django;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        access_log off;
    }

    # Проксирование всех остальных запросов на Django
    location / {
        proxy_pass http://django;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
```

```
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;

        # Таймауты
        proxy_connect_timeout 75s;
        proxy_send_timeout 300s;
        proxy_read_timeout 300s;

        # Буферизация
        proxy_buffering off;
        proxy_buffer_size 4k;
        proxy_buffers 8 4k;
        proxy_busy_buffers_size 8k;
    }

    # Запрет доступа к скрытым файлам
    location ~ /\. {
        deny all;
        access_log off;
        log_not_found off;
    }

    # Запрет доступа к файлам с расширениями, которые не должны быть доступны
    location ~* \.(py|sql|db|sh|ini|conf|bak|swp)$ {
        deny all;
        access_log off;
```

```
        log_not_found off;
    }
}
```

3.3. Активация конфигурации

```
# Создание символьной ссылки в sites-enabled
sudo ln -s /etc/nginx/sites-available/myproject /etc/nginx/sites-enabled/

# Проверка конфигурации
sudo nginx -t

# Перезагрузка Nginx
sudo systemctl reload nginx
```

Шаг 4. Настройка systemd для Gunicorn

Файл /etc/systemd/system/gunicorn.service:

```
[Unit]
Description=Gunicorn service for myproject
After=network.target postgresql.service redis.service

[Service]
Type=notify
User=www-data
Group=www-data

# Рабочая директория
WorkingDirectory=/var/www/myproject
```

```
# Переменные окружения
Environment="PATH=/var/www/myproject/venv/bin"
Environment="DJANGO_SETTINGS_MODULE=myproject.settings"
EnvironmentFile=/var/www/myproject/.env

# Команда запуска
ExecStart=/var/www/myproject/venv/bin/gunicorn \
    --config /var/www/myproject/gunicorn.conf.py \
    myproject.wsgi:application

# Перезапуск при падении
Restart=on-failure
RestartSec=10

# Ограничения ресурсов
LimitNOFILE=65535
LimitNPROC=65535
MemoryMax=512M
CPUQuota=200%

# Логирование
StandardOutput=journal
StandardError=journal
SyslogIdentifier=gunicorn

[Install]
WantedBy=multi-user.target
```

Управление сервисом:

```
# Перезагрузка systemd
sudo systemctl daemon-reload

# Запуск сервиса
sudo systemctl start gunicorn

# Остановка
sudo systemctl stop gunicorn

# Перезапуск
sudo systemctl restart gunicorn

# Проверка статуса
sudo systemctl status gunicorn

# Включение автозапуска
sudo systemctl enable gunicorn

# Просмотр логов
sudo journalctl -u gunicorn -f
```

Шаг 5. Настройка SSL с Let's Encrypt

5.1. Установка Certbot

```
# Установка Certbot и плагина для Nginx
sudo apt update
sudo apt install certbot python3-certbot-nginx

# Проверка установки
certbot --version
```

5.2. Получение SSL-сертификата

```
# Автоматическая настройка (рекомендуется)
sudo certbot --nginx -d myproject.com -d www.myproject.com

# Только получение сертификата (без автоматической настройки)
sudo certbot certonly --nginx -d myproject.com -d www.myproject.com

# Получение wildcard-сертификата (для *.myproject.com)
sudo certbot certonly --manual --preferred-challenges dns -d myproject.com
-d *.myproject.com
```

5.3. Настройка автоматического обновления

```
# Проверка продления (dry-run)
sudo certbot renew --dry-run

# Автоматическое обновление (встроено в cron)
sudo systemctl status certbot.timer

# Ручное обновление
sudo certbot renew
```

Шаг 6. Настройка firewall

```
# Разрешение SSH
sudo ufw allow 22

# Разрешение HTTP и HTTPS
```

```
sudo ufw allow 80/tcp
sudo ufw allow 443/tcp

# Включение firewall
sudo ufw enable

# Проверка статуса
sudo ufw status verbose
```

Шаг 7. Настройка мониторинга

7.1. Настройка логирования

Файл /etc/logrotate.d/myproject:

```
/var/log/myproject/*.log {
    daily
    missingok
    rotate 30
    compress
    delaycompress
    notifempty
    create 0640 www-data www-data
    sharedscripts
    postrotate
        systemctl reload gunicorn > /dev/null 2>&1 || true
        systemctl reload nginx > /dev/null 2>&1 || true
    endscript
}
```

7.2. Healthcheck эндпоинт

Файл api/views.py:

```
from django.http import JsonResponse
from django.db import connections
from django.db.utils import OperationalError
from django.core.cache import cache

def health_check(request):
    status = {
        'status': 'ok',
        'timestamp': timezone.now().isoformat(),
    }

    # Проверка базы данных
    try:
        connections['default'].cursor()
        status['database'] = 'ok'
    except OperationalError:
        status['database'] = 'error'
        status['status'] = 'degraded'

    # Проверка Redis (если используется)
    try:
        cache.set('health_check', 'ok', timeout=5)
        if cache.get('health_check') == 'ok':
            status['redis'] = 'ok'
        else:
            status['redis'] = 'error'
            status['status'] = 'degraded'
```

```
    except Exception:
        status['redis'] = 'error'
        status['status'] = 'degraded'

    http_status = 200 if status['status'] == 'ok' else 503
    return JsonResponse(status, status=http_status)
```

4. Полный процесс развертывания (чеклист)

```
# 1. Подготовка сервера
sudo apt update && sudo apt upgrade -y
sudo apt install python3-pip python3-venv nginx postgresql redis

# 2. Клонирование репозитория
git clone https://github.com/username/myproject.git /var/www/myproject
cd /var/www/myproject

# 3. Настройка виртуального окружения
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt

# 4. Настройка переменных окружения
cp .env.example .env
nano .env

# 5. Миграции и сбор статики
python manage.py migrate
python manage.py collectstatic --noinput
```

```
# 6. Настройка Gunicorn
cp gunicorn.conf.py /etc/gunicorn/
sudo cp systemd/gunicorn.service /etc/systemd/system/
sudo systemctl daemon-reload
sudo systemctl enable gunicorn
sudo systemctl start gunicorn

# 7. Настройка Nginx
sudo cp nginx/myproject.conf /etc/nginx/sites-available/
sudo ln -s /etc/nginx/sites-available/myproject /etc/nginx/sites-enabled/
sudo nginx -t
sudo systemctl reload nginx

# 8. Настройка SSL
sudo certbot --nginx -d myproject.com -d www.myproject.com

# 9. Настройка firewall
sudo ufw allow 22
sudo ufw allow 80/tcp
sudo ufw allow 443/tcp
sudo ufw enable

# 10. Проверка
curl https://myproject.com/health/
```

5. Индивидуальные задания

5.1. Базовый уровень (15 вариантов)

Требования: Настроить Gunicorn и Nginx для Django-приложения, настроить раздачу статики.

Вариант	Дополнительная настройка
1	Настройка Gunicorn с 4 воркерами
2	Настройка Gunicorn с unix socket
3	Настройка Nginx с раздачей статики
4	Настройка Nginx с раздачей медиа
5	Настройка логов Gunicorn
6	Настройка логов Nginx
7	Настройка лимитов (client_max_body_size)
8	Настройка таймаутов Gunicorn
9	Настройка keepalive в Nginx
10	Настройка gzip сжатия
11	Настройка rate limiting
12	Настройка healthcheck эндпоинта
13	Настройка redirect с HTTP на HTTPS
14	Настройка security headers
15	Настройка раздачи favicon и robots.txt

5.2. Средний уровень (15 вариантов)

Требования: Настроить SSL-сертификаты через Let's Encrypt, настроить systemd для Gunicorn.

Вариант	Дополнительная настройка
1	Получение SSL-сертификата через Certbot
2	Настройка автоматического обновления сертификатов
3	Настройка HSTS (Strict-Transport-Security)
4	Настройка CSP (Content-Security-Policy)
5	Настройка OCSP stapling
6	Настройка SSL session cache
7	Настройка TLS 1.3
8	Настройка CAA записи в DNS
9	Настройка wildcard-сертификата
10	Настройка мультидоменного сертификата
11	Настройка rate limiting в Nginx
12	Настройка блэклинга IP
13	Настройка allow/deny по IP
14	Настройка basic auth для staging

Вариант	Дополнительная настройка
15	Настройка geoip для разных регионов

5.3. Продвинутый уровень (15 вариантов)

Требования: Настроить балансировку нагрузки, кэширование, мониторинг.

Вариант	Дополнительная настройка
1	Настройка балансировки нагрузки между несколькими Gunicorn
2	Настройка кэширования ответов в Nginx
3	Настройка кэширования статики с ETag
4	Настройка CDN (CloudFront, CloudFlare)
5	Настройка WebSocket через Nginx (Django Channels)
6	Настройка Prometheus метрик для Nginx
7	Настройка Grafana дашборда
8	Настройка канареечного деплоя
9	Настройка сине-зелёного деплоя
10	Настройка автоматического масштабирования
11	Настройка гео-балансировки

Вариант	Дополнительная настройка
12	Настройка защиты от DDoS (fail2ban)
13	Настройка WAF (ModSecurity)
14	Настройка автогенерации сертификатов
15	Настройка ACME DNS challenge

6. Контрольные вопросы

1. Почему нельзя использовать runserver в production?
2. Что такое Gunicorn и зачем он нужен?
3. Что такое reverse proxy и почему используется Nginx?
4. Как настроить Gunicorn для запуска Django?
5. Как настроить Nginx для раздачи статических файлов?

6. Как рассчитать оптимальное количество воркеров Gunicorn?
7. В чём разница между workers и threads в Gunicorn?
8. Как настроить systemd для управления Gunicorn?
9. Как получить SSL-сертификат через Let's Encrypt?
10. Как настроить автоматическое обновление сертификатов?
11. Как настроить балансировку нагрузки между несколькими Gunicorn?
12. Как настроить кэширование ответов в Nginx?
13. Как настроить WebSocket через Nginx (Django Channels)?
14. Как настроить мониторинг и алертинг?
15. Как настроить канареечный деплой?

7. Требования к отчёту

Титульный лист (название работы, ФИО, группа, вариант)

Цель работы

Теоретическое обоснование (своими словами)

Выполнение практического примера (конфигурации, команды, скриншоты)

Выполнение индивидуального задания (конфигурации, скриншоты)

Ответы на контрольные вопросы (не менее 5 по своему уровню)

Вывод

8. Критерии оценивания

Критерий	Базовый (max 35)	Средний (max 50)	Продвинутый (max 65)
Настройка Gunicorn	15	10	10
Настройка Nginx (статика, прокси)	15	10	10
Настройка systemd	—	10	10
SSL (Let's Encrypt)	—	15	10
Балансировка, кэширование, мониторинг	—	—	15
Качество кода и документация	5	5	10

Критерий	Базовый (max 35)	Средний (max 50)	Продвинутый (max 65)
Итого	35	50	65

Шкала перевода баллов в оценку:

Оценка	Базовый (35)	Средний (50)	Продвинутый (65)
5 (отлично)	32-35	46-50	59-65
4 (хорошо)	28-31	40-45	51-58
3 (удовлетворительно)	22-27	32-39	42-50
2 (неудовлетворительно)	0-21	0-31	0-41

Лабораторная работа №31

Развертывание на VPS

1. Цель работы

Научиться развертывать веб-приложение на удаленном сервере (VPS). Освоить процесс аренды VPS, настройку SSH-доступа, установку Docker и Docker Compose, клонирование репозитория, запуск приложения и настройку фаервола для обеспечения безопасности.

2. Теоретическое обоснование

2.1. Что такое VPS?

VPS (Virtual Private Server) - это виртуальный выделенный сервер, который работает в облачной инфраструктуре. По сути, вы арендуете часть физического сервера, которая ведёт себя как полноценный独立ный сервер с собственными ресурсами, операционной системой и доступом по SSH.

Аналогия: Представьте многоквартирный дом. Обычный хостинг - это общага (ресурсы общие, соседи могут влиять на скорость). VPS - это собственная квартира (у вас есть своё пространство, никто не мешает, но дом общий). Выделенный сервер - это собственный дом (всё только ваше).

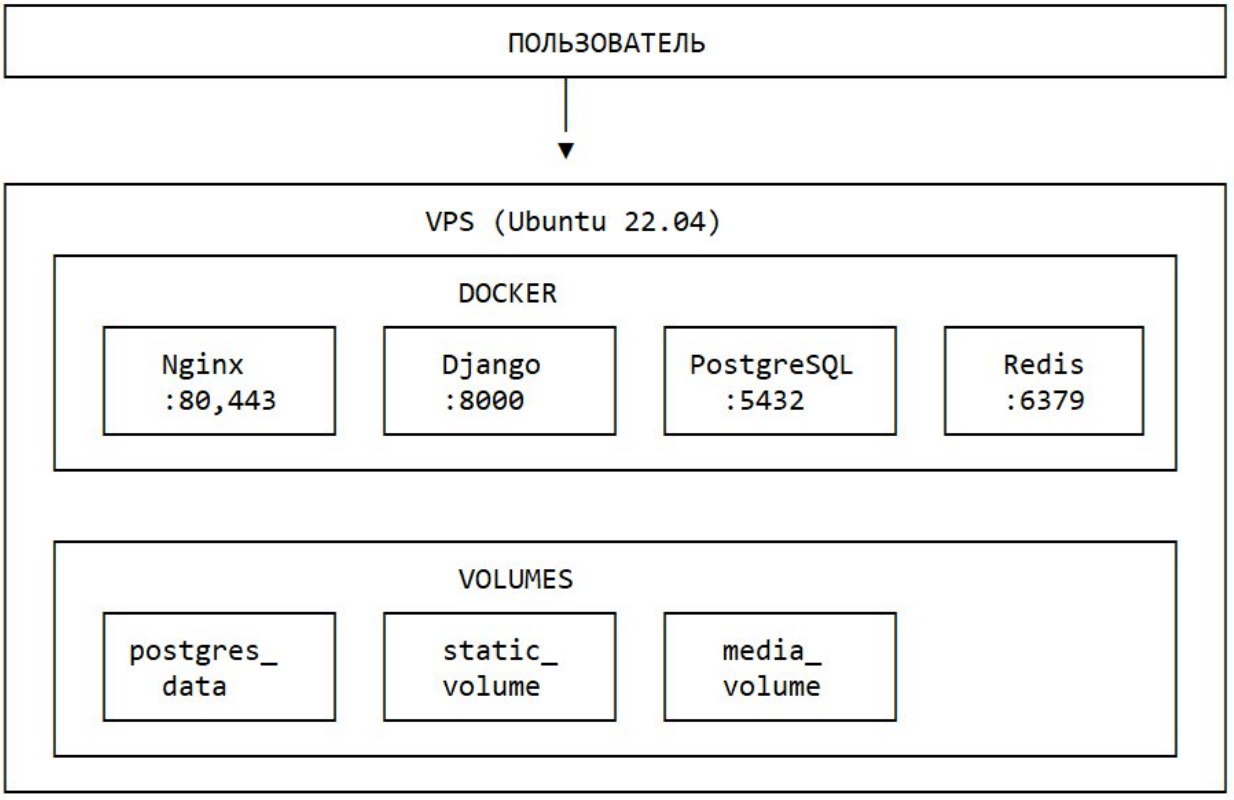
2.2. Сравнение типов хостинга

Тип хостинга	Плюсы	Минусы	Для каких проектов
Shared Hosting	Дешёвый, простой	Ограниченные ресурсы, соседи	Блоги, лендинги
VPS	Полный контроль, масштабируемый	Требует администрирования	Большинство проектов
Dedicated Server	Максимальная производительность	Дорогой, сложный	Крупные проекты
Cloud (AWS/GCP)	Эластичность, глобальная инфраструктура	Сложный биллинг	Enterprise

2.3. Почему Docker на VPS?

Без Docker	С Docker
Нужно вручную устанавливать зависимости	Всё описано в Dockerfile
Разные версии Python конфликтуют	Каждый сервис в своём контейнере
Трудно переехать на другой сервер	Один образ - везде одинаково
Сложно масштабировать	Легко запустить несколько копий

2.4. Архитектура развертывания



3. Практический пример

Что мы будем делать: Арендуем VPS, настроим SSH-доступ, установим Docker и Docker Compose, развернём Django-приложение с PostgreSQL и Nginx.

Шаг 1. Выбор и аренда VPS

1.1. Популярные провайдеры

Провайдер	Минимальная цена	Особенности
DigitalOcean	~$4/мес	Простой интерфейс, отличная документация
Vultr	~$2.5/мес	Много локаций
Hetzner	~€3.5/мес	Отличное соотношение цена/качество
TimeWeb	~₽250/мес	Российский провайдер
Reg.ru VPS	~₽300/мес	Российский провайдер
AWS Lightsail	~$3.5/мес	Экосистема AWS

1.2. Создание сервера (на примере DigitalOcean)

```
# 1. Зарегистрироваться на DigitalOcean
# 2. Создать проект
# 3. Нажать "Create" → "Droplets"
```

```
# Параметры:
# - Distribution: Ubuntu 22.04 LTS
# - Plan: Basic ($4-6/мес)
# - CPU: 1 GB RAM / 1 CPU (минимально)
# - Add block storage: не нужно (для старта)
# - Choose datacenter region: ближайший к пользователям
# - Authentication: SSH Key (рекомендуется) или пароль

# 4. Добавить SSH ключ (показано ниже)
# 5. Выбрать имя: myproject-vps
# 6. Нажать "Create Droplet"
```

Шаг 2. Настройка SSH-доступа

2.1. Генерация SSH-ключа (локально)

```
# Генерация SSH-ключа (если ещё нет)
ssh-keygen -t ed25519 -C "your-email@example.com"

# По умолчанию ключи сохраняются в ~/.ssh/id_ed25519 (приватный) и ~/.ssh/id_ed25519.pub (публичный)

# Просмотр публичного ключа
cat ~/.ssh/id_ed25519.pub

# Вывод (пример):
# ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAI... your-email@example.com
```

2.2. Добавление SSH-ключа в панель провайдера

```
# DigitalOcean:
# Settings → Security → SSH Keys → Add SSH Key
# Вставить содержимое ~/.ssh/id_ed25519.pub

# Другие провайдеры:
# Аналогично, ищите "SSH Keys" в настройках
```

2.3. Подключение к серверу

```
# Подключение по SSH (IP из панели)
ssh root@<IP_вашего_сервера>

# Или с указанием пользователя (Ubuntu)
ssh ubuntu@<IP_вашего_сервера>

# Пример:
ssh root@159.89.123.45

# При первом подключении будет запрос на подтверждение fingerprint
# Введите "yes" и нажмите Enter
```

2.4. Создание пользователя с правами sudo (безопасность)

```
# Войдите на сервер
ssh root@<IP_вашего_сервера>

# Создание нового пользователя
adduser deploy
```

```
# Добавление пользователя в группу sudo
usermod -aG sudo deploy

# Копирование SSH-ключа root для нового пользователя
rsync --archive --chown=deploy:deploy ~/.ssh /home/deploy/

# Проверка
su - deploy
ssh -T git@github.com  # для проверки ключа
```

2.5. Настройка SSH для безопасности

```
# Редактирование конфигурации SSH
sudo nano /etc/ssh/sshd_config

# Изменить (или добавить):
# PermitRootLogin no
# PasswordAuthentication no
# PubkeyAuthentication yes
# Port 2222 (опционально, для смены порта)

# Перезапуск SSH
sudo systemctl restart sshd

# Теперь подключаемся как deploy:
ssh deploy@<IP> -p 2222  # если меняли порт
```

Шаг 3. Установка Docker и Docker Compose

3.1. Установка Docker

```
# Обновление пакетов
sudo apt update && sudo apt upgrade -y

# Установка зависимостей
sudo apt install -y apt-transport-https ca-certificates curl software-properties-common

# Добавление официального GPG-ключа Docker
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

# Добавление репозитория Docker
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

# Установка Docker
sudo apt update
sudo apt install -y docker-ce docker-ce-cli containerd.io docker-compose-plugin

# Добавление пользователя в группу docker (чтобы не использовать sudo)
sudo usermod -aG docker $USER

# Перезагрузка (или выход и вход)
newgrp docker
```

```
# Проверка установки
docker --version
docker compose version
```

3.2. Установка Docker Compose V2 (альтернативный способ)

```
# Скачивание Docker Compose
sudo curl -L "https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

# Установка прав на выполнение
sudo chmod +x /usr/local/bin/docker-compose

# Проверка
docker-compose --version
```

Шаг 4. Настройка фаервола (UFW)

bash

```
# Установка UFW (если не установлен)
sudo apt install -y ufw

# Разрешение SSH (ВАЖНО: сделать до включения!)
sudo ufw allow 22/tcp
# Если меняли порт SSH: sudo ufw allow 2222/tcp

# Разрешение HTTP и HTTPS
sudo ufw allow 80/tcp
sudo ufw allow 443/tcp
```

```
# Разрешение порта приложения (если нужно для отладки)
sudo ufw allow 8000/tcp

# Включение фаервола
sudo ufw enable

# Проверка статуса
sudo ufw status verbose

# Вывод:
# Status: active
# To                         Action      From
# --                         ------      ----
# 22/tcp                     ALLOW       Anywhere
# 80/tcp                     ALLOW       Anywhere
# 443/tcp                    ALLOW       Anywhere
```

Шаг 5. Подготовка проекта для деплоя

5.1. Структура репозитория

```
myproject/
├── .github/
│   └── workflows/
│       └── deploy.yml       # CI/CD (опционально)
├── backend/
│   ├── Dockerfile
│   ├── requirements.txt
│   ├── entrypoint.sh
```

```
│   ├── manage.py
│   └── myproject/
│       └── settings.py
├── nginx/
│   ├── Dockerfile
│   └── nginx.conf
├── docker-compose.yml
├── .env.prod
├── .env.example
└── README.md
```

5.2. Файл .env.prod (переменные окружения)

```
# Django
DJANGO_SECRET_KEY=very-secret-key-change-in-production
DJANGO_DEBUG=False
DJANGO_ALLOWED_HOSTS=your-domain.com,www.your-domain.com,<IP_сервера>

# База данных
DB_NAME=myproject_db
DB_USER=myproject_user
DB_PASSWORD=strong-password-here
DB_HOST=postgres
DB_PORT=5432

# Redis
REDIS_HOST=redis
REDIS_PORT=6379
```

```
# Настройки email
EMAIL_HOST=smtp.gmail.com
EMAIL_PORT=587
EMAIL_USE_TLS=True
EMAIL_HOST_USER=your-email@gmail.com
EMAIL_HOST_PASSWORD=your-app-password
```

5.3. docker-compose.prod.yml (production версия)

```
version: '3.8'

services:
  postgres:
    image: postgres:15-alpine
    container_name: myproject_postgres
    restart: always
    environment:
      POSTGRES_DB: ${DB_NAME}
      POSTGRES_USER: ${DB_USER}
      POSTGRES_PASSWORD: ${DB_PASSWORD}
    volumes:
      - postgres_data:/var/lib/postgresql/data
    networks:
      - myproject_network
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U ${DB_USER}"]
      interval: 10s
      timeout: 5s
```

```
    retries: 5

  redis:
    image: redis:7-alpine
    container_name: myproject-redis
    restart: always
    command: redis-server --appendonly yes
    volumes:
      - redis_data:/data
    networks:
      - myproject-network
    healthcheck:
      test: ["CMD", "redis-cli", "ping"]
      interval: 10s
      timeout: 5s
      retries: 5

  backend:
    build: ./backend
    container_name: myproject-backend
    restart: always
    environment:
      DJANGO_SECRET_KEY: ${DJANGO_SECRET_KEY}
      DJANGO_DEBUG: ${DJANGO_DEBUG}
      DJANGO_ALLOWED_HOSTS: ${DJANGO_ALLOWED_HOSTS}
      DB_NAME: ${DB_NAME}
      DB_USER: ${DB_USER}
      DB_PASSWORD: ${DB_PASSWORD}
      DB_HOST: postgres
```

```
    DB_PORT: ${DB_PORT}
    REDIS_HOST: redis
    REDIS_PORT: ${REDIS_PORT}
  volumes:
    - static_volume /app/static
    - media_volume /app/media
  depends_on:
    postgres:
      condition: service_healthy
    redis:
      condition: service_healthy
  networks:
    - myproject-network

nginx:
  build: ./nginx
  container_name: myproject-nginx
  restart: always
  ports:
    - "80:80"
    - "443:443"
  volumes:
    - static_volume /static
    - media_volume /media
    - ./ssl /etc/nginx/ssl
  depends_on:
    - backend
  networks:
    - myproject-network
```

```
volumes:
  postgres_data:
    name: myproject_postgres_data
  redis_data:
    name: myproject_redis_data
  static_volume:
    name: myproject_static_volume
  media_volume:
    name: myproject_media_volume

networks:
  myproject-network:
    name: myproject_network
    driver: bridge
```

Шаг 6. Деплой на сервер

6.1. Клонирование репозитория

```
# Подключение к серверу
ssh deploy@<IP>

# Установка Git (если не установлен)
sudo apt install -y git

# Клонирование репозитория
git clone https://github.com/username/myproject.git /home/deploy/myproject
cd /home/deploy/myproject
```

```
# Или с использованием SSH-ключа (если приватный репозиторий)
git clone git@github.com:username/myproject.git /home/deploy/myproject
```

6.2. Настройка переменных окружения

```
# Создание .env файла из примера
cp .env.example .env

# Редактирование переменных
nano .env

# Проверка
cat .env
```

```
# Запуск в фоновом режиме
docker compose -f docker-compose.prod.yml up -d

# Просмотр логов
docker compose -f docker-compose.prod.yml logs -f

# Проверка статуса
docker compose -f docker-compose.prod.yml ps
```

6.4. Миграции и создание суперпользователя

```
# Выполнение миграций
docker compose -f docker-compose.prod.yml exec backend python manage.py migrate
```

```
# Сбор статики
docker compose -f docker-compose.prod.yml exec backend python manage.py collectstatic --noinput

# Создание суперпользователя
docker compose -f docker-compose.prod.yml exec backend python manage.py createsuperuser
```

Шаг 7. Настройка домена (опционально)

7.1. Покупка домена

```
# Регистраторы:
# - Namecheap
# - GoDaddy
# - Reg.ru
# - Nic.ru
```

7.2. Настройка DNS

```
# В панели управления доменом добавить A-запись:
# Type: A
# Name: @
# Value: <IP_вашего_сервера>
# TTL: 3600

# Для www:
# Type: A
# Name: www
```

```
# Value: <IP_вашего_сервера>
```

7.3. Настройка SSL (Certbot)

```
# Установка Certbot
sudo apt install -y certbot

# Получение сертификата
sudo certbot certonly --standalone -d your-domain.com -d www.your-domain.com

# Настройка автоматического обновления
sudo crontab -e
# Добавить: 0 2 * * * certbot renew --quiet
```

Шаг 8. Мониторинг и логи

```
# Просмотр логов Docker
docker compose -f docker-compose.prod.yml logs --tail=100

# Просмотр логов в реальном времени
docker compose -f docker-compose.prod.yml logs -f

# Просмотр использования ресурсов
docker stats

# Просмотр дискового пространства
df -h
```

```
# Просмотр использования памяти
free -h

# Просмотр загрузки CPU
top
```

Шаг 9. Обновление приложения

```
# Подключение к серверу
ssh deploy@<IP>

# Переход в папку проекта
cd /home/deploy/myproject

# Получение последних изменений
git pull origin main

# Пересборка и перезапуск контейнеров
docker compose -f docker-compose.prod.yml up -d --build

# Применение миграций (если есть)
docker compose -f docker-compose.prod.yml exec backend python manage.py migrate

# Сбор статики (если есть изменения)
docker compose -f docker-compose.prod.yml exec backend python manage.py collectstatic --noinput
```

Шаг 10. Бэкап данных

```
# Создание бэкапа базы данных
docker compose -f docker-compose.prod.yml exec postgres pg_dump -U ${DB_USER} ${DB_NAME} > backup_$(date +%Y%m%d).sql

# Копирование медиа-файлов
docker cp myproject-backend:/app/media ./backup_media_$(date +%Y%m%d)

# Автоматический бэкап через cron
sudo crontab -e
# Добавить:
# 0 3 * * * cd /home/deploy/myproject && docker compose exec -T postgres pg_dump -U myproject_user myproject_db > /home/deploy/backups/backup_$(date +\%Y\%m\%d).sql
```

4. Скрипт автоматического деплоя

Файл deploy.sh:

```
#!/bin/bash

set -e  # Остановка при ошибке

echo "🚀 Начало деплоя..."

# Переменные
PROJECT_DIR="/home/deploy/myproject"
BACKUP_DIR="/home/deploy/backups"
DATE=$(date +%Y%m%d_%H%M%S)
```

```
# 1. Переход в директорию проекта
cd $PROJECT_DIR

# 2. Получение последних изменений
echo "📦 Обновление кода..."
git pull origin main

# 3. Бэкап базы данных
echo "💾 Создание бэкапа базы данных..."
mkdir -p $BACKUP_DIR
docker compose -f docker-compose.prod.yml exec -T postgres pg_dump -U ${DB_USER} ${DB_NAME} > $BACKUP_DIR/backup_$DATE.sql

# 4. Пересборка и перезапуск контейнеров
echo "🐳 Пересборка контейнеров..."
docker compose -f docker-compose.prod.yml up -d --build

# 5. Применение миграций
echo "🏷 Применение миграций..."
docker compose -f docker-compose.prod.yml exec backend python manage.py migrate --noinput

# 6. Сбор статики
echo "📎 Сбор статических файлов..."
docker compose -f docker-compose.prod.yml exec backend python manage.py collectstatic --noinput
```

```
# 7. Очистка старых образов
echo "    Очистка неиспользуемых образов..."
docker image prune -f

echo "✅ Деплой завершён успешно!"

# Делаем скрипт исполняемым
chmod +x deploy.sh

# Запуск
./deploy.sh
```

5. Индивидуальные задания

5.1. Базовый уровень (15 вариантов)

Требования: Арендовать VPS, настроить SSH, установить Docker и Docker Compose.

Вариант	Провайдер	Регион	Дополнительная настройка
1	Digital Ocean	Frankfurt	Настройка UFW
2	Vultr	Amsterdam	Настройка fail2ban
3	Hetzner	Nuremberg	Настройка SSH на порт 2222
4	TimeW	Mosc	Настройка swap

Вариант	Провайдер	Регион	Дополнительная настройка
	eb	ow	
5	Reg.ru	Moscow	Настройка docker без sudo
6	AWS Lightsail	Frankfurt	Настройка CloudWatch
7	Google Cloud	Belgium	Настройка firewall
8	Azure	Netherlands	Настройка мониторинга
9	Linode	London	Настройка логирования
10	OVH	Warsaw	Настройка бэкапов
11	Scaleway	Paris	Настройка IPv6
12	Ionos	Berlin	Настройка hostname
13	UpCloud	Chicago	Настройка timezone

Вариант	Провайдер	Регион	Дополнительная настройка
14	Hostinger	Singapore	Настройка locale
15	GoDaddy	Arizona	Настройка cron для обновлений

5.2. Средний уровень (15 вариантов)

Требования: Развернуть Django-приложение с PostgreSQL и Nginx, настроить домен.

Вариант	Дополнительная настройка
1	Настройка SSL через Let's Encrypt
2	Настройка домена и DNS
3	Настройка автоматических бэкапов
4	Настройка мониторинга (Prometheus + Grafana)
5	Настройка логирования (Loki + Promtail)
6	Настройка CI/CD (GitHub Actions)
7	Настройка CDN (CloudFront)
8	Настройка WebSocket (Django Channels)
9	Настройка Redis для кэша

Вариант	Дополнительная настройка
10	Настройка Celery для фоновых задач
11	Настройка Sentry для ошибок
12	Настройка Rate Limiting
13	Настройка GeoIP
14	Настройка WAF (ModSecurity)
15	Настройка Docker Swarm

5.3. Продвинутый уровень (15 вариантов)

Требования: Настроить высокую доступность (HA), балансировку нагрузки, автодеплой.

Вариант	Дополнительная настройка
1	Настройка балансировки нагрузки (HAProxy)
2	Настройка репликации базы данных
3	Настройка автоматического масштабирования
4	Настройка канареечного деплоя
5	Настройка сине-зелёного деплоя
6	Настройка Kubernetes (k3s)
7	Настройка Service Mesh (Linkerd)

Вариант	Дополнительная настройка
8	Настройка CI/CD с ArgoCD
9	Настройка мониторинга в реальном времени
10	Настройка алертинга (PagerDuty)
11	Настройка инфраструктуры как код (Terraform)
12	Настройка конфигурации как код (Ansible)
13	Настройка секретов (HashiCorp Vault)
14	Настройка сервисной сетки (Istio)
15	Настройка мультирегионального развертывания

6. Контрольные вопросы

1. Что такое VPS и чем он отличается от обычного хостинга?
2. Как подключиться к серверу по SSH?
3. Как создать нового пользователя и дать ему права sudo?
4. Как установить Docker на Ubuntu?
5. Как настроить SSH для безопасности (отключение root login, password auth)?
6. Как развернуть приложение с Docker Compose?
7. Как применить миграции в контейнере?
8. Как настроить домен и DNS для сервера?

9. Как получить SSL-сертификат через Let's Encrypt?
10. Продвинутый уровень
11. Как настроить автоматическое обновление кода (CI/CD)?
12. Как настроить мониторинг и алертинг?
13. Как настроить бэкап базы данных и медиа-файлов?
14. Как настроить отказоустойчивость (High Availability)?
15. Как настроить автоматическое масштабирование?

7. Требования к отчёту

Титульный лист (название работы, ФИО, группа, вариант)

Цель работы

Теоретическое обоснование (своими словами)

Выполнение практического примера (команды, скриншоты)

Выполнение индивидуального задания (конфигурации, скриншоты)

Ответы на контрольные вопросы (не менее 5 по своему уровню)

Вывод

8. Критерии оценивания

Критерий	Базовый (max 35)	Средний (max 50)	Продвинутый (max 65)
Аренда VPS и SSH	10	10	10
Установка Docker и Compose	15	10	10
Развертывание приложения	10	10	10

Критерий	Базовый (max 35)	Средний (max 50)	Продвинутый (max 65)
Настройка домена и SSL	—	15	10
CI/CD и мониторинг	—	5	10
HA и балансировка	—	—	10
Качество документации	—	—	5
Итого	35	50	65

Шкала перевода баллов в оценку:

Оценка	Базовый (35)	Средний (50)	Продвинутый (65)
5 (отлично)	32-35	46-50	59-65
4 (хорошо)	28-31	40-45	51-58
3 (удовлетворительно)	22-27	32-39	42-50
2 (неудовлетворительно)	0-21	0-31	0-41

Лабораторная работа №32

Настройка CI/CD с GitHub Actions

1. Цель работы

Освоить настройку непрерывной интеграции и непрерывной доставки (CI/CD) для веб-приложений с использованием GitHub Actions. Научиться создавать workflow-файлы, автоматически запускать тесты при пуше в репозиторий, собирать Docker-образы, пушить их в container registry и автоматически деплоить приложение на VPS.

2. Теоретическое обоснование

2.1. Что такое CI/CD?

CI/CD (Continuous Integration / Continuous Delivery) - это набор практик и инструментов для автоматизации процессов разработки, тестирования и развертывания программного обеспечения.

Аналогия: Представьте конвейер на заводе. Вы кладёте заготовку (код) на ленту, и автоматически:

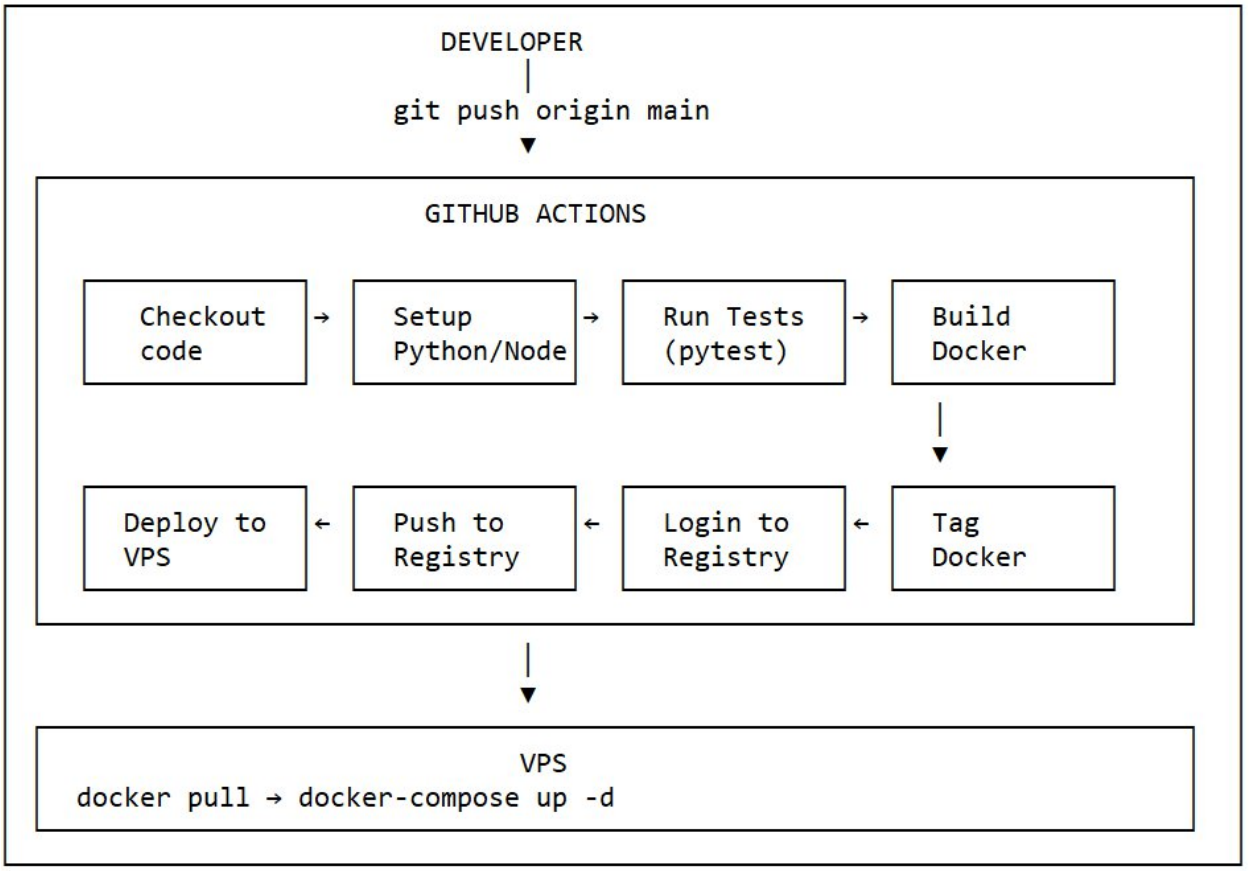
- Проверяется качество (тесты)
- Собирается продукт (сборка)
- Упаковывается (Docker-образ)
- Отправляется на склад (registry)
- Доставляется клиенту (деплой)

2.2. Компоненты CI/CD

Компонент	Что делает	В нашем случае
CI(Continuous Integration)	Автоматическая сборка и тестирование при	npm test, pytest

Компонент	Что делает	В нашем случае
	каждом пуше	
CD (Continuous Delivery)	Автоматическая подготовка к релизу	Сборка Docker-образа, пуш в registry
CD (Continuous Deployment)	Автоматический деплой на сервер	Деплой на VPS

2.3. Архитектура CI/CD с GitHub Actions



2.4. Основные понятия GitHub Actions

Понятие	Описание	Пример
Workflow	Автоматизированный процесс	deploy.yml
Event	Событие, запускающее workflow	push, pull_request
Job	Набор шагов, выполняющихся на одном runner	test, build, deploy
Step	Отдельная команда или action	uses: actions/checkout@v4
Runner	Сервер, где выполняются jobs	ubuntu-latest
Action	Переиспользуемый блок кода	actions/setup-python@v5

3. Практический пример

Что мы будем делать: Настроим CI/CD для Django-приложения с Docker: запуск тестов, сборка образа, пуш в Docker Hub и деплой на VPS.

Шаг 1. Подготовка репозитория

1.1. Структура репозитория

```
myproject/
├── .github/
```

```
│   └── workflows/
│       └── deploy.yml          # CI/CD конфигурация
├── backend/
│   ├── Dockerfile
│   ├── requirements.txt
│   ├── requirements-test.txt
│   ├── pytest.ini
│   ├── manage.py
│   └── myproject/
│       └── settings.py
├── nginx/
│   ├── Dockerfile
│   └── nginx.conf
├── docker-compose.yml
├── docker-compose.prod.yml
├── .env.example
└── README.md
```

1.2. Секреты GitHub (Settings → Secrets and variables → Actions)

Secret Name	Значение	Откуда взять
DOCKER_USERNAME	Имя пользователя Docker Hub	dockerhub.com
DOCKER_PASSWORD	Пароль или токен	Settings → Security
DOCKER_REPO	Название репозитория	username/my

Secret Name	Значение	Откуда взять
		project
VPS_HOST	IP или домен VPS	DigitalOcean / Vultr
VPS_USERNAME	Имя пользователя (deploy)	При создании
VPS_SSH_KEY	Приватный SSH-ключ	cat ~/.ssh/id_ed25519
VPS_PORT	Порт SSH (22 или другой)	Настройка SSH

Шаг 2. Создание файла тестов (pytest)

2.1. Файл requirements-test.txt

```
-r requirements.txt
pytest==7.4.3
pytest-django==4.5.2
factory-boy==3.3.0
faker==20.1.0
```

2.2. Файл pytest.ini

```
[pytest]
DJANGO_SETTINGS_MODULE = myproject.settings
```

```
python_files = test_*.py *_test.py tests.py
testpaths = tests
addopts =
    --reuse-db
    --nomigrations
    --cov=api
    --cov-report=term
    --cov-report=html
```

2.3. Тесты (пример)

Файл backend/api/tests/test_models.py:

```
import pytest
from django.contrib.auth.models import User
from api.models import Category, News, Comment

@pytest.mark.django_db
class TestCategoryModel:
    def test_category_creation(self):
        category = Category.objects.create(name='Технологии', slug='tech')
        assert category.name == 'Технологии'
        assert str(category) == 'Технологии'

@pytest.mark.django_db
class TestNewsModel:
    def test_news_creation(self):
        user = User.objects.create_user(username='testuser')
        category = Category.objects.create(name='Новости', slug='news')
        news = News.objects.create(
```

```
            title='Тестовая новость',
            slug='test-news',
            content='Содержание',
            category=category,
            author=user
        )
        assert news.title == 'Тестовая новость'
        assert news.status == 'draft'
```

Шаг 3. Создание workflow-файла

Файл .github/workflows/deploy.yml:

```
name: CI/CD Pipeline

# События, запускающие workflow
on:
  push:
    branches: [ main, develop ]
    paths-ignore:
      - 'README.md'
      - '.gitignore'
      - 'docs/**'
  pull_request:
    branches: [ main ]
  workflow_dispatch:  # Ручной запуск

# Переменные окружения для всего workflow
env:
  DOCKER_REPO: ${{ secrets.DOCKER_REPO }}
```

```
  DOCKER_TAG: ${{ github.sha }}
  BRANCH_NAME: ${{ github.ref_name }}

# Jobs выполняются параллельно, если не указаны зависимости
jobs:

  #============ JOB 1: ТЕСТИРОВАНИЕ ====================
  test:
    name:      Run Tests
    runs-on: ubuntu-latest
    timeout-minutes: 10

    services:
      # Тестовая PostgreSQL
      postgres:
        image: postgres:15-alpine
        env:
          POSTGRES_DB: test_db
          POSTGRES_USER: test_user
          POSTGRES_PASSWORD: test_pass
        ports:
          - 5432:5432
        options: >-
          --health-cmd pg_isready
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5

      # Тестовый Redis
```

```
    redis:
      image: redis:7-alpine
      ports:
        - 6379:6379
      options: >-
        --health-cmd "redis-cli ping"
        --health-interval 10s
        --health-timeout 5s
        --health-retries 5

  steps:
    # 1. Клонирование кода
    - name: 📦 Checkout code
      uses: actions/checkout@v4

    # 2. Установка Python
    - name: 🐍 Setup Python
      uses: actions/setup-python@v5
      with:
        python-version: '3.11'
        cache: 'pip'
        cache-dependency-path: backend/requirements-test.txt

    # 3. Установка зависимостей
    - name: 📦 Install dependencies
      run: |
        cd backend
        python -m pip install --upgrade pip
```

```
        pip install -r requirements-test.txt

    # 4. Настройка переменных окружения для тестов
    - name: 🔧 Setup environment
      run: |
        echo "DB_NAME=test_db" >> $GITHUB_ENV
        echo "DB_USER=test_user" >> $GITHUB_ENV
        echo "DB_PASSWORD=test_pass" >> $GITHUB_ENV
        echo "DB_HOST=localhost" >> $GITHUB_ENV
        echo "DB_PORT=5432" >> $GITHUB_ENV
        echo "REDIS_HOST=localhost" >> $GITHUB_ENV
        echo "REDIS_PORT=6379" >> $GITHUB_ENV
        echo "DJANGO_SECRET_KEY=test-secret-key" >>
$GITHUB_ENV
        echo "DJANGO_DEBUG=True" >> $GITHUB_ENV
        echo "DJANGO_ALLOWED_HOSTS=*" >> $GITHUB_ENV

    # 5. Запуск миграций
    - name: 🏷 Run migrations
      run: |
        cd backend
        python manage.py migrate

    # 6. Запуск тестов с покрытием
    - name:  Run tests with coverage
      run: |
        cd backend
        pytest --cov=api --cov-report=xml --cov-report=term
```

```yaml
    # 7. Загрузка отчёта о покрытии (опционально)
    - name: 📊 Upload coverage to Codecov
      uses: codecov/codecov-action@v3
      with:
        file: ./backend/coverage.xml
        flags: unittests
        name: codecov-umbrella
        fail_ci_if_error: false

# ========== JOB 2: СБОРКА DOCKER-ОБРАЗА ==========
build:
  name: 🐳 Build Docker Image
  runs-on: ubuntu-latest
  needs: [test]  # Зависит от успешного тестирования
  if: github.event_name == 'push' && github.ref == 'refs/heads/main'

  steps:
    - name: 📦 Checkout code
      uses: actions/checkout@v4

    # 1. Настройка Docker Buildx (для multi-platform сборки)
    - name: 🔧 Setup Docker Buildx
      uses: docker/setup-buildx-action@v3

    # 2. Логин в Docker Hub
    - name: 🔐 Login to Docker Hub
      uses: docker/login-action@v3
```

```
    with:
      username: ${{ secrets.DOCKER_USERNAME }}
      password: ${{ secrets.DOCKER_PASSWORD }}

  # 3. Сборка и пуш Docker-образа
  - name: 🐳 Build and push
    uses: docker/build-push-action@v5
    with:
      context: ./backend
      file: ./backend/Dockerfile
      push: true
      tags: |
        ${{ secrets.DOCKER_REPO }}:latest
        ${{ secrets.DOCKER_REPO }}:${{ github.sha }}
        ${{ secrets.DOCKER_REPO }}:${{ github.ref_name }}
      cache-from: type=gha
      cache-to: type=gha,mode=max
      labels: |
        org.opencontainers.image.title=MyProject
        org.opencontainers.image.description=Django application
        org.opencontainers.image.version=${{ github.sha }}
        org.opencontainers.image.revision=${{ github.sha }}
        org.opencontainers.image.created=${{ github.event.head_commit.timestamp }}

  # 4. Проверка образа
  - name: 🔍 Verify image
    run: |
      docker pull ${{ secrets.DOCKER_REPO }}:latest
```

```
        docker images

    # ==================== JOB 3: ДЕПЛОЙ НА VPS ====================
    deploy:
      name: 🚀 Deploy to VPS
      runs-on: ubuntu-latest
      needs: [build]  # Зависит от успешной сборки
      if: github.event_name == 'push' && github.ref == 'refs/heads/main'
      environment:
        name: production
        url: https://your-domain.com

      steps:
        - name: 📦 Checkout code
          uses: actions/checkout@v4

        # 1. Настройка SSH
        - name: 🔧 Setup SSH
          run: |
            mkdir -p ~/.ssh
            echo "${{ secrets.VPS_SSH_KEY }}" > ~/.ssh/id_rsa
            chmod 600 ~/.ssh/id_rsa
            ssh-keyscan -H ${{ secrets.VPS_HOST }} >> ~/.ssh/known_hosts

        # 2. Проверка подключения
        - name: 🔌 Test SSH connection
          run: |
```

```
ssh -o ConnectTimeout=10 ${{ secrets.VPS_USERNAME }}@${{ secrets.VPS_HOST }} "echo 'SSH connection successful'"

      # 3. Деплой на сервер
      - name: 🚀 Deploy to VPS
        run: |
          ssh ${{ secrets.VPS_USERNAME }}@${{ secrets.VPS_HOST }} << 'EOF'
            set -e
            echo "🚀 Starting deployment..."

            # Переход в директорию проекта
            cd /home/deploy/myproject

            # Обновление кода
            git pull origin main

            # Обновление переменных окружения
            # (предполагается, что .env уже существует на сервере)

            # Пересборка и перезапуск контейнеров
            docker compose -f docker-compose.prod.yml pull backend
            docker compose -f docker-compose.prod.yml up -d --build

            # Применение миграций
            docker compose -f docker-compose.prod.yml exec backend python manage.py migrate --noinput
```

```
            # Сбор статики
            docker compose -f docker-compose.prod.yml exec backend python manage.py collectstatic --noinput

            # Очистка неиспользуемых образов
            docker image prune -f

            # Проверка статуса
            docker compose -f docker-compose.prod.yml ps

            echo "✅ Deployment completed successfully!"
          EOF

      # 4. Проверка healthcheck эндпоинта
      - name: Health check
        run: |
          sleep 10
          curl -f https://your-domain.com/health/ || exit 1

  # ==================== JOB 4: УВЕДОМЛЕНИЯ ====================
  notify:
    name: ☎ Send Notification
    runs-on: ubuntu-latest
    needs: [test, build, deploy]
    if: always()  # Выполняется всегда (успех или ошибка)

    steps:
      # Уведомление в Telegram
```

```
    - name: 📱 Telegram notification
      uses: appleboy/telegram-action@master
      with:
        to: ${{ secrets.TELEGRAM_CHAT_ID }}
        token: ${{ secrets.TELEGRAM_BOT_TOKEN }}
        message: |
          🚀 CI/CD Pipeline
          Repository: ${{ github.repository }}
          Branch: ${{ github.ref_name }}
          Commit: ${{ github.event.head_commit.message }}
          Author: ${{ github.event.head_commit.author.name }}
          Status: ${{ job.status }}
          Workflow: ${{ github.workflow }}
          Run URL: ${{ github.server_url }}/${{ github.repository }}/actions/runs/${{ github.run_id }}
      if: ${{ env.TELEGRAM_BOT_TOKEN != '' && env.TELEGRAM_CHAT_ID != '' }}

    # Уведомление в Slack (опционально)
    - name: 💬 Slack notification
      uses: slackapi/slack-github-action@v1.24.0
      with:
        payload: |
          {
            "text": "CI/CD Pipeline completed! Status: ${{ job.status }}",
            "blocks": [
              {
```

```
                "type": "section",
                "text": {
                  "type": "mrkdwn",
                  "text": "🚀 *CI/CD Pipeline*\nStatus: ${{ job.status }}\nRepository: ${{ github.repository }}\nBranch: ${{ github.ref_name }}"
                }
              }
            ]
          }
        env:
          SLACK_WEBHOOK_URL: ${{ secrets.SLACK_WEBHOOK_URL }}
        if: ${{ env.SLACK_WEBHOOK_URL != '' }}
```

Шаг 4. Расширенная конфигурация (разные окружения)

Файл .github/workflows/deploy.yml (с окружениями):

```
name: CI/CD Pipeline

on:
  push:
    branches: [ main, staging, develop ]

jobs:

  test:
    name: Test
    runs-on: ubuntu-latest
    # ... те же шаги, что выше ...
```

```
  # Деплой на staging
  deploy-staging:
    name: Deploy to Staging
    runs-on: ubuntu-latest
    needs: [test]
    if: github.ref == 'refs/heads/staging'
    environment:
      name: staging
      url: https://staging.your-domain.com
    steps:
      - name: Deploy to Staging VPS
        # ... аналогично деплою на production, но на staging сервер

  # Деплой на production
  deploy-production:
    name: Deploy to Production
    runs-on: ubuntu-latest
    needs: [test, deploy-staging]
    if: github.ref == 'refs/heads/main'
    environment:
      name: production
      url: https://your-domain.com
    steps:
      - name: Deploy to Production VPS
        # ... шаги деплоя
```

Шаг 5. Проверка статуса CI/CD

```
# После пуша в репозиторий
```

```
git add .
git commit -m "Add CI/CD pipeline"
git push origin main

# В браузере:
# 1. Перейти на GitHub → ваш репозиторий → Actions
# 2. Увидеть запущенный workflow
# 3. Кликнуть на него для просмотра логов
```

Шаг 6. Настройка GitHub Actions для фронтенда (React)

Файл .github/workflows/frontend.yml (для React):

```
name: Frontend CI/CD

on:
  push:
    branches: [ main ]
    paths:
      - 'frontend/**'

jobs:
  test-frontend:
    name:     Test React App
    runs-on: ubuntu latest
    defaults:
      run:
        working-directory: ./frontend

    steps:
      - uses: actions/checkout@v4
```

```yaml
    - name: 🔧 Setup Node.js
      uses: actions/setup-node@v4
      with:
        node-version: '20'
        cache: 'npm'
        cache-dependency-path: frontend/package-lock.json

    - name: 📦 Install dependencies
      run: npm ci

    - name: 🧪 Run tests
      run: npm test -- --coverage --watchAll=false

    - name: 🏗️ Build application
      run: npm run build

    - name: 📊 Upload coverage
      uses: codecov/codecov-action@v3
      with:
        directory: ./frontend/coverage
        flags: frontend

  deploy-frontend:
    name: 🚀 Deploy Frontend to Vercel
    runs-on: ubuntu-latest
    needs: test-frontend
```

```yaml
steps:
  - uses: actions/checkout@v4

  - name: 🔧 Setup Node.js
    uses: actions/setup-node@v4
    with:
      node-version: '20'

  - name: 📦 Install dependencies
    run: |
      cd frontend
      npm ci

  - name: 🏗️ Build
    run: |
      cd frontend
      npm run build

  - name: 🚀 Deploy to Vercel
    uses: amondnet/vercel-action@v25
    with:
      vercel-token: ${{ secrets.VERCEL_TOKEN }}
      vercel-org-id: ${{ secrets.VERCEL_ORG_ID }}
      vercel-project-id: ${{ secrets.VERCEL_PROJECT_ID }}
      vercel-args: '--prod'
      working-directory: ./frontend
```

Шаг 7. Тегирование и версионирование

```
# В workflow добавить создание Git-тега
- name: 🏷 Create Git tag
  if: github.ref == 'refs/heads/main'
  run: |
    VERSION=$(date +'%Y.%m.%d-%H%M%S')
    git tag "v${VERSION}"
    git push origin "v${VERSION}"
```

Шаг 8. Matrix стратегия для тестирования

```
test:
  name: Test with Python ${{ matrix.python-version }} and Django ${{ matrix.django-version }}
  runs-on: ubuntu-latest
  strategy:
    fail-fast: false
    matrix:
      python-version: ['3.10', '3.11', '3.12']
      django-version: ['4.2', '5.0']

  steps:
    - name: Setup Python ${{ matrix.python-version }}
      uses: actions/setup-python@v5
      with:
        python-version: ${{ matrix.python-version }}

    - name: Install dependencies
      run: |
        pip install Django==${{ matrix.django-version }}
```

```
        pip install -r requirements-test.txt

    - name: Run tests
      run: pytest
```

4. Настройка секретов в GitHub

```
    # 1. Перейти в репозиторий → Settings → Secrets and variables → Actions
    # 2. Нажать "New repository secret"

    # Необходимые секреты:

    # Docker Hub
    DOCKER_USERNAME: your-dockerhub-username
    DOCKER_PASSWORD: your-dockerhub-password-or-token
    DOCKER_REPO: your-dockerhub-username/myproject

    # VPS
    VPS_HOST: 159.89.123.45
    VPS_USERNAME: deploy
    VPS_SSH_KEY: |
      -----BEGIN OPENSSH PRIVATE KEY-----
      ...
      -----END OPENSSH PRIVATE KEY-----
    VPS_PORT: 22

    # Telegram (опционально)
```

TELEGRAM_BOT_TOKEN: 1234567890:ABCdefGHIjklMNOpqrsTUVwxyz

TELEGRAM_CHAT_ID: 123456789

Slack (опционально)

SLACK_WEBHOOK_URL: https://hooks.slack.com/services/...

Vercel (для фронтенда)

VERCEL_TOKEN: your-vercel-token

VERCEL_ORG_ID: team_...

VERCEL_PROJECT_ID: prj_...

5. Индивидуальные задания

5.1. Базовый уровень (15 вариантов)

Требования: Настроить CI (тесты и линтинг) для бэкенда или фронтенда.

Вариант	Тип приложения	Дополнительные проверки
1	Django	flake8, black, isort
2	Django	mypy (типизация)
3	Django	bandit (безопасность)
4	Django	safety (уязвимости)
5	React	ESLint

Вариант	Тип приложения	Дополнительные проверки
6	React	Prettier
7	React	TypeScript проверка
8	Django + React	Оба с линтингом
9	Django	Тесты с coverage > 80%
10	Django	Тесты с PostgreSQL
11	Django	Тесты с Redis
12	React	Тесты с Jest
13	React	Тесты с React Testing Library
14	Django	Тесты API (DRF)
15	Django	Тесты моделей и форм

5.2. Средний уровень (15 вариантов)

Требования: Настроить сборку Docker-образа и пуш в registry.

Вариант	Registry	Дополнительная настройка
1	Docker Hub	Tag latest и git-sha
2	GitHub Container Registry	ghcr.io
3	GitLab Registry	registry.gitlab.com
4	Amazon ECR	aws ecr
5	Google GAR	gcr.io
6	Azure ACR	azurecr.io
7	DigitalOcean Registry	докер-репозиторий DO
8	Yandex Cloud CR	cr.yandex
9	Harbor (self-hosted)	Свой registry
10	Quay.io	quay.io
11	Multi-platform сборка	linux/amd64, linux/arm64
12	Кэширование слоёв	GitHub

Вариант	Registry	Дополнительная настройка
		Actions cache
13	Подпись образов	cosign
14	SBOM (Software Bill of Materials)	syft
15	Сканирование уязвимостей	Trivy

5.3. Продвинутый уровень (15 вариантов)

Требования: Настроить CD (автоматический деплой на сервер) с разными окружениями.

Вариант	Среда	Дополнительная настройка
1	VPS (staging + production)	Два окружения
2	VPS + Docker Compose	Автоматический деплой
3	VPS + Kubernetes (k3s)	kubectl apply
4	AWS ECS (Fargate)	ecs deploy
5	Google Cloud Run	gcloud run deploy

Вариант	Среда	Дополнительная настройка
6	Azure Container Apps	az containerapp deploy
7	Heroku	git push heroku
8	Railway	railway up
9	Render	render deploy
10	Fly.io	fly deploy
11	Netlify (frontend)	netlify deploy
12	Vercel (frontend)	vercel deploy
13	Cloudflare Pages	wrangler deploy
14	GitHub Pages	gh-pages
15	Multi-cloud (AWS + GCP)	Балансировка

6. Контрольные вопросы

1. Что такое CI/CD и зачем он нужен?
2. Что такое GitHub Actions?
3. Какие основные компоненты GitHub Actions вы знаете?
4. Как создать workflow-файл?
5. Как запустить тесты при пуше в репозиторий?

6. Как собрать Docker-образ в GitHub Actions?
7. Как загрузить образ в Docker Hub?
8. Как настроить SSH-соединение для деплоя?
9. Как выполнить команды на удалённом сервере?
10. Как добавить секреты в GitHub Actions?
11. Как настроить разные окружения (staging/production)?
12. Как настроить матрицу тестирования (разные версии)?
13. Как настроить кэширование для ускорения сборки?
14. Как настроить уведомления о статусе деплоя?
15. Как настроить откат (rollback) при ошибке?

7. Требования к отчёту

Титульный лист (название работы, ФИО, группа, вариант)

Цель работы

Теоретическое обоснование (своими словами)

Выполнение практического примера (workflow.yml, скриншоты)

Выполнение индивидуального задания (конфигурации, скриншоты)

Ответы на контрольные вопросы (не менее 5 по своему уровню)

Вывод

8. Критерии оценивания

Критерий	Базовый (max 35)	Средний (max 50)	Продвинутый (max 65)
Создание workflow-файла	15	10	10
Запуск тестов	15	10	10

Критерий	Базовый (max 35)	Средний (max 50)	Продвинутый (max 65)
Сборка Docker-образа	—	15	10
Пуш в registry	—	10	10
Автоматический деплой	—	5	10
Разные окружения (staging/prod)	—	—	10
Качество документации	5	—	5
Итого	35	50	65

Шкала перевода баллов в оценку:

Оценка	Базовый (35)	Средний (50)	Продвинутый (65)
5 (отлично)	32-35	46-50	59-65
4 (хорошо)	28-31	40-45	51-58
3 (удовлетворительно)	22-27	32-39	42-50
2	0-21	0-31	0-41

Оценка	Базовый (35)	Средний (50)	Продвинутый (65)
(неудовлетворительно)			

Лабораторная работа №33

Автоматизация с Ansible

1. Цель работы

Научиться описывать инфраструктуру как код (Infrastructure as Code, IaC) с использованием Ansible. Освоить установку Ansible, создание инвентарных файлов, написание плейбуков для автоматической установки Docker и настройки окружения, а также запуск плейбуков для управления серверами.

2. Теоретическое обоснование

2.1. Что такое Infrastructure as Code (IaC)?

Infrastructure as Code (IaC) - это подход к управлению инфраструктурой, при котором конфигурация серверов, сетей и других компонентов описывается в виде кода, который можно хранить в системе контроля версий, тестировать и применять автоматически.

Аналогия: Вместо того чтобы вручную настраивать каждый сервер (устанавливать пакеты, создавать пользователей, настраивать firewall), вы пишете "рецепт" (плейбук). Этот рецепт можно выполнить на любом сервере, и он приведёт сервер в нужное состояние.

Преимущества IaC:

Преимущество	Описание
Воспроизводимость	Одинаковая конфигурация на всех серверах
Версионность	Изменения в Git, можно откатиться
Автоматизация	Не нужно настраивать вручную
Документация	Код сам является документацией

2.2. Что такое Ansible?

Ansible -это инструмент для автоматизации IT-инфраструктуры (управление конфигурациями, оркестрация, развертывание приложений). Он работает без агентов на целевых серверах, используя SSH для подключения.

Сравнение Ansible с альтернативами:

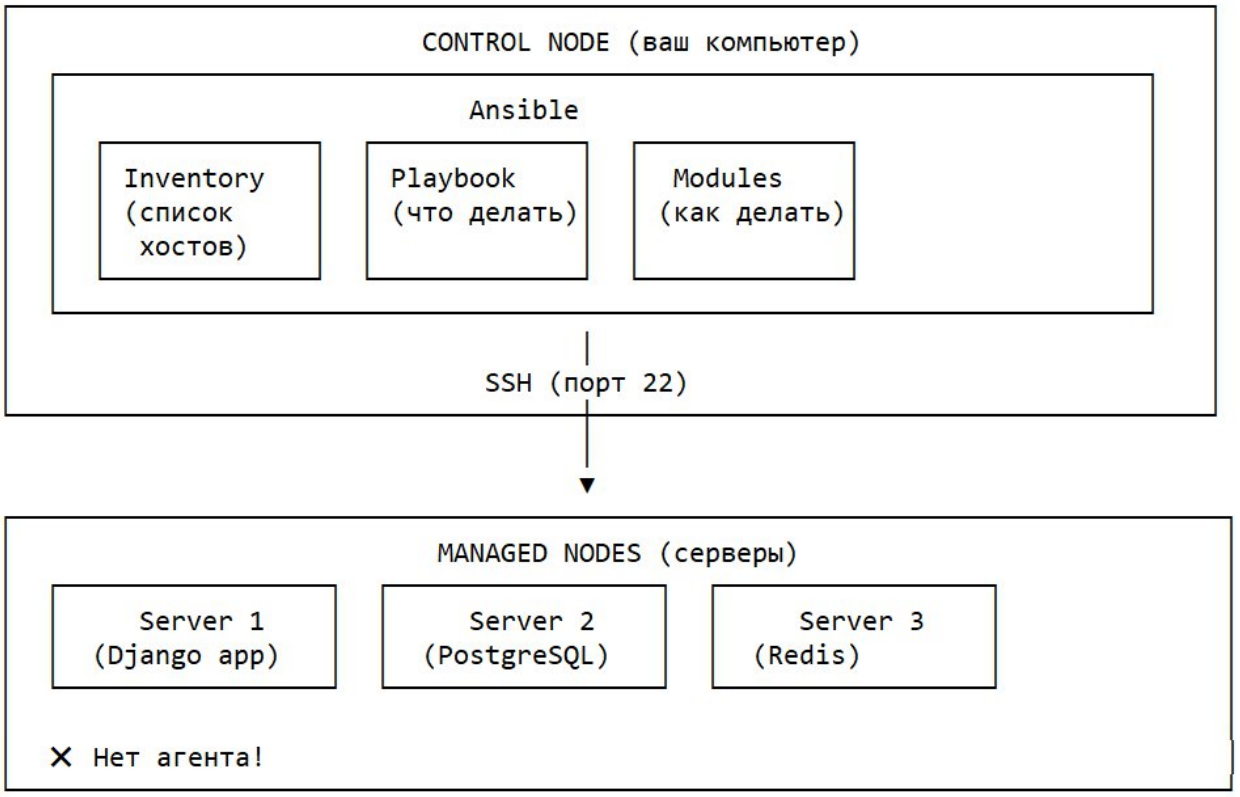
Инструмент	Язык	Агент	Сложность	Масштаб
Ansible	YAML	Без агента	Низкая	Любой
Terraform	HCL	Без агента	Средняя	Облачная инфраструктура
Chef	Ruby	Да	Высокая	Крупные проекты
Puppet	DSL	Да	Высокая	Крупные проекты

Инструмент	Язык	Агент	Сложность	Масштаб
SaltStack	YAML/Python	Да	Средняя	Высоконагруженные системы

2.3. Почему Ansible?

Преимущество	Описание
Без агентов	Не нужно устанавливать ПО на целевые серверы
Простой синтаксис	YAML - понятный человеко-читаемый формат
Идемпотентность	Запуск плейбука много раз даёт тот же результат
Большое сообщество	Много готовых ролей и модулей
Интеграция с Docker	Модули для управления контейнерами

2.4. Архитектура Ansible



2.5. Основные понятия Ansible

Понятие	Описание	Пример
Control Node	Компьютер, с которого запускается Ansible	Ваш ноутбук, GitHub Actions runner
Managed Node	Сервер, который настраивается	VPS с Ubuntu
Inventory	Список управляемых хостов	inventory.yml
Playboo	YAML-файл с	playbook.yml

Понятие	Описание	Пример
k	описанием задач	
Task	Одна операция (установка пакета, копирование файла)	apt: name=docker.io
Module	Реализация задачи (встроенная)	apt, copy, service
Role	Набор задач, переменных, шаблонов	ansible-role-docker
Handler	Задача, выполняемая при изменении	restart nginx
Variable	Переменная для параметризации	{{ ansible_host }}

3. Практический пример

Что мы будем делать: Настроим Ansible для автоматической установки Docker, Docker Compose, создания пользователя и настройки окружения на VPS.

Шаг 1. Установка Ansible

1.1. Установка на Linux/macOS

```
# Ubuntu/Debian
sudo apt update
```

```
sudo apt install -y ansible

# macOS (через Homebrew)
brew install ansible

# Установка через pip
pip install ansible

# Проверка установки
ansible --version
```

1.2. Установка на Windows (через WSL)

```
# В WSL (Ubuntu)
sudo apt update
sudo apt install -y ansible
```

Шаг 2. Структура проекта Ansible

```
ansible/
├── ansible.cfg              # Конфигурация Ansible
├── inventory.yml            # Инвентарь (список серверов)
├── group_vars/              # Переменные для групп хостов
│   └── all.yml
├── host_vars/               # Переменные для конкретных хостов
│   └── vps.yml
├── playbooks/               # Плейбуки
│   ├── setup-server.yml     # Базовая настройка сервера
│   ├── deploy-docker.yml    # Установка Docker
│   └── deploy-app.yml       # Развертывание приложения
├── roles/                   # Роли (переиспользуемые задачи)
```

```
│   └── common/
│       ├── tasks/
│       │   └── main.yml
│       └── vars/
│           └── main.yml
└── files/                  # Статические файлы
    └── ssh/
        └── authorized_keys
```

Шаг 3. Создание конфигурационного файла

Файл ansible/ansible.cfg:

```
[defaults]
# Инвентарь по умолчанию
inventory = inventory.yml

# Путь к директории с ролями
roles_path = roles

# Отключение проверки SSH-ключей хостов (только для разработки!)
host_key_checking = False

# Количество параллельных процессов
forks = 5

# Таймаут подключения
timeout = 30

# Логирование
```

```
log_path = /var/log/ansible.log

# Вывод в формате YAML
stdout_callback = yaml

# Подключение через SSH
transport = smart

[ssh_connection]
# Ускорение SSH-соединений
pipelining = True
control_path = /tmp/ansible-ssh-%%h-%%p-%%r
```

Шаг 4. Создание инвентарного файла

Файл ansible/inventory.yml:

```
all:
  vars:
    # Переменные для всех хостов
    ansible_user: root
    ansible_port: 22
    ansible_python_interpreter: /usr/bin/python3
    timezone: Europe/Moscow
    deploy_user: deploy
    deploy_user_home: /home/deploy
    project_name: myproject
    project_path: /home/deploy/myproject

  children:
```

```yaml
# Группа production серверов
production:
  vars:
    environment: production
    env_file: .env.prod
  hosts:
    vps1:
      ansible_host: 159.89.123.45
      ansible_user: root
      hostname: vps1.example.com
    vps2:
      ansible_host: 159.89.123.46
      ansible_user: root
      hostname: vps2.example.com

# Группа staging серверов
staging:
  vars:
    environment: staging
    env_file: .env.staging
  hosts:
    staging1:
      ansible_host: 159.89.123.47
      ansible_user: root
      hostname: staging.example.com

# Группа баз данных
database:
  vars:
```

```
      db_role: primary
    hosts:
      postgres1:
        ansible_host: 159.89.123.48
        ansible_user: root
        db_type: postgresql
      redis1:
        ansible_host: 159.89.123.49
        ansible_user: root
        db_type: redis

  # Группы для ролей серверов
  webserver:
    hosts:
      vps1:
      vps2:
      staging1:

  appserver:
    hosts:
      vps1:
      vps2
```

Шаг 5. Переменные для групп хостов

Файл ansible/group_vars/all.yml:

```
# Общие переменные для всех хостов
---
timezone: Europe/Moscow
```

```yaml
locale: ru_RU.UTF-8

# Пользователь для приложения
deploy_user: deploy
deploy_user_home: /home/deploy
deploy_user_shell: /bin/bash
deploy_user_ssh_key: "{{ lookup('file', '~/.ssh/id_ed25519.pub') }}"

# Настройки Docker
docker_edition: 'ce'
docker_package: docker.io
docker_compose_version: '2.23.0'

# Порты
app_port: 8000
nginx_port: 80
nginx_ssl_port: 443

# Пути
project_path: "{{ deploy_user_home }}/{{ project_name }}"
static_path: "{{ project_path }}/static"
media_path: "{{ project_path }}/media"
logs_path: "/var/log/{{ project_name }}"

# Настройки Python
python_version: '3.11'
virtualenv_path: "{{ project_path }}/venv"

# Репозиторий
```

```
git_repo: "git@github.com:username/{{ project_name }}.git"
git_branch: "{{ environment }}"
```

Шаг 6. Плейбук для настройки сервера

Файл ansible/playbooks/setup-server.yml:

```
---
- name: Basic server setup
  hosts: all
  become: yes
  gather_facts: yes
  vars:
    packages:
      - apt-transport-https
      - ca-certificates
      - curl
      - software-properties-common
      - git
      - htop
      - nginx
      - ufw

  tasks:
    # 1. Обновление системы
    - name: Update apt cache
      apt:
        update_cache: yes
        cache_valid_time: 3600
```

```
- name: Upgrade all packages
  apt:
    upgrade: dist

# 2. Установка необходимых пакетов
- name: Install required packages
  apt:
    name: "{{ packages }}"
    state: present

# 3. Настройка часового пояса
- name: Set timezone
  timezone:
    name: "{{ timezone }}"

# 4. Настройка локали
- name: Set locale
  locale_gen:
    name: "{{ locale }}"
    state: present

# 5. Создание пользователя deploy
- name: Create deploy user
  user:
    name: "{{ deploy_user }}"
    shell: "{{ deploy_user_shell }}"
    groups: sudo docker
    append: yes
    state: present
```

```
    create_home: yes

# 6. Добавление SSH ключа для deploy пользователя
- name: Add SSH key for deploy user
  authorized_key:
    user: "{{ deploy_user }}"
    state: present
    key: "{{ deploy_user_ssh_key }}"

# 7. Настройка sudo без пароля для deploy
- name: Configure sudo without password
  copy:
    content: |
      {{ deploy_user }} ALL=(ALL) NOPASSWD:ALL
    dest: /etc/sudoers.d/{{ deploy_user }}
    mode: 0440
    validate: 'visudo -cf %s'

# 8. Создание директорий
- name: Create application directories
  file:
    path: "{{ item }}"
    state: directory
    owner: "{{ deploy_user }}"
    group: "{{ deploy_user }}"
    mode: '0755'
  loop:
    - "{{ project_path }}"
    - "{{ static_path }}"
```

```
    - "{{ media_path }}"
    - "{{ logs_path }}"

# 9. Настройка UFW (фаервол)
- name: Configure UFW
  ufw:
    rule: "{{ item.rule }}"
    port: "{{ item.port }}"
    proto: "{{ item.proto }}"
  loop:
    - { rule: allow, port: '22', proto: tcp }
    - { rule: allow, port: '80', proto: tcp }
    - { rule: allow, port: '443', proto: tcp }
    - { rule: allow, port: '8000', proto: tcp }

- name: Enable UFW
  ufw:
    state: enabled
    policy: deny
    direction: incoming

# 10. Настройка fail2ban (защита от брутфорса)
- name: Install fail2ban
  apt:
    name: fail2ban
    state: present

- name: Configure fail2ban
  copy:
```

```
    content: |
      [DEFAULT]
      bantime = 3600
      findtime = 600
      maxretry = 5

      [sshd]
      enabled = true
      port = ssh
      filter = sshd
      logpath = /var/log/auth.log
      maxretry = 3
    dest: /etc/fail2ban/jail.local
  notify: restart fail2ban

handlers:
  - name: restart fail2ban
    service:
      name: fail2ban
      state: restarted
```

Шаг 7. Плейбук для установки Docker

Файл ansible/playbooks/deploy-docker.yml:

```
---
- name: Install Docker and Docker Compose
  hosts: all
  become: yes
  gather_facts: yes
```

```yaml
tasks:
  # 1. Удаление старых версий Docker
  - name: Remove old Docker packages
    apt:
      name:
        - docker
        - docker-engine
        - docker.io
        - containerd
        - runc
      state: absent

  # 2. Установка зависимостей
  - name: Install prerequisites
    apt:
      name:
        - apt-transport-https
        - ca-certificates
        - curl
        - gnupg
        - lsb-release
      state: present

  # 3. Добавление официального GPG-ключа Docker
  - name: Add Docker GPG key
    apt_key:
      url: https://download.docker.com/linux/ubuntu/gpg
      state: present
```

```
    # 4. Добавление репозитория Docker
    - name: Add Docker repository
      apt_repository:
        repo: "deb [arch=amd64] https://download.docker.com/linux/ubuntu
{{ ansible_lsb.codename }} stable"
        state: present

    # 5. Установка Docker
    - name: Install Docker
      apt:
        name:
          - docker-ce
          - docker-ce-cli
          - containerd.io
          - docker-buildx-plugin
          - docker-compose-plugin
        update_cache: yes
        state: present

    # 6. Добавление пользователя deploy в группу docker
    - name: Add deploy user to docker group
      user:
        name: "{{ deploy_user }}"
        groups: docker
        append: yes

    # 7. Установка Docker Compose V2 (альтернативный способ)
    - name: Install Docker Compose V2
```

```
    get_url:
      url:
"https://github.com/docker/compose/releases/download/v{{ docker_compose_versi
on }}/docker-compose-linux-x86_64"
      dest: /usr/local/bin/docker-compose
      mode: '0755'

  # 8. Запуск Docker
  - name: Start Docker service
    systemd:
      name: docker
      state: started
      enabled: yes

  # 9. Проверка установки
  - name: Check Docker version
    command: docker --version
    register: docker_version
    changed_when: false

  - name: Display Docker version
    debug:
      msg: "Docker version: {{ docker_version.stdout }}"

  # 10. Настройка Docker daemon (опционально)
  - name: Configure Docker daemon
    copy:
      content: |
        {
```

```
        "log-driver": "json-file",
        "log-opts": {
          "max-size": "10m",
          "max-file": "3"
        },
        "storage-driver": "overlay2",
        "default-ulimits": {
          "nofile": {
            "Name": "nofile",
            "Hard": 64000,
            "Soft": 64000
          }
        }
      }
    dest: /etc/docker/daemon.json
  notify: restart docker

handlers:
  - name: restart docker
    systemd:
      name: docker
      state: restarted
      daemon_reload: yes
```

Шаг 8. Плейбук для развертывания приложения

Файл ansible/playbooks/deploy-app.yml:

```
---
- name: Deploy application with Docker
```

```yaml
hosts: webserver
become: yes
vars:
  compose_files:
    - docker-compose.yml
    - docker-compose.prod.yml

tasks:
  # 1. Клонирование репозитория
  - name: Clone repository
    git:
      repo: "{{ git_repo }}"
      dest: "{{ project_path }}"
      version: "{{ git_branch }}"
      force: yes
    become_user: "{{ deploy_user }}"

  # 2. Копирование файла .env
  - name: Copy environment file
    copy:
      src: "../files/{{ env_file }}"
      dest: "{{ project_path }}/.env"
      owner: "{{ deploy_user }}"
      group: "{{ deploy_user }}"
      mode: '0600'
    when: env_file is defined

  # 3. Создание директорий для volumes
  - name: Create volume directories
```

```
    file:
      path: "{{ item }}"
      state: directory
      owner: "{{ deploy_user }}"
      group: "{{ deploy_user }}"
      mode: '0755'
    loop:
      - "{{ project_path }}/postgres_data"
      - "{{ project_path }}/redis_data"
      - "{{ project_path }}/static_volume"
      - "{{ project_path }}/media_volume"

  # 4. Копирование Docker Compose файлов
  - name: Copy Docker Compose files
    copy:
      src: "{{ item }}"
      dest: "{{ project_path }}/{{ item }}"
      owner: "{{ deploy_user }}"
      group: "{{ deploy_user }}"
      mode: '0644'
    loop: "{{ compose_files }}"
    when: compose_files is defined

  # 5. Pull последних образов
  - name: Pull Docker images
    docker_compose:
      project_src: "{{ project_path }}"
      files: "{{ compose_files }}"
      pull: yes
```

```
  become_user: "{{ deploy_user }}"

# 6. Запуск контейнеров
- name: Start containers
  docker_compose:
    project_src: "{{ project_path }}"
    files: "{{ compose_files }}"
    state: present
    restarted: yes
  become_user: "{{ deploy_user }}"

# 7. Применение миграций
- name: Run Django migrations
  docker_compose:
    project_src: "{{ project_path }}"
    files: "{{ compose_files }}"
    services: backend
    command: python manage.py migrate --noinput
  become_user: "{{ deploy_user }}"

# 8. Сбор статики
- name: Collect static files
  docker_compose:
    project_src: "{{ project_path }}"
    files: "{{ compose_files }}"
    services: backend
    command: python manage.py collectstatic --noinput
  become_user: "{{ deploy_user }}"
```

```yaml
# 9. Проверка статуса контейнеров
- name: Check containers status
  docker_compose:
    project_src: "{{ project_path }}"
    files: "{{ compose_files }}"
    state: present
  register: compose_status

- name: Display compose status
  debug:
    var: compose_status

# 10. Настройка Nginx (если нужно)
- name: Copy Nginx configuration
  template:
    src: ../templates/nginx.conf.j2
    dest: /etc/nginx/sites-available/{{ project_name }}
    owner: root
    group: root
    mode: '0644'
  notify: reload nginx

- name: Enable Nginx site
  file:
    src: "/etc/nginx/sites-available/{{ project_name }}"
    dest: "/etc/nginx/sites-enabled/{{ project_name }}"
    state: link
  notify: reload nginx
```

```
    - name: Remove default Nginx site
      file:
        path: /etc/nginx/sites enabled/default
        state: absent
      notify: reload nginx

  handlers:
    - name: reload nginx
      systemd:
        name: nginx
        state: reloaded
```

Шаг 9. Шаблон Nginx конфигурации (Jinja2)

Файл ansible/templates/nginx.conf.j2:

```
upstream {{ project_name }} {
    server 127.0.0.1:8000;
}

server {
    listen 80;
    server_name {{ ansible_host }};
    client_max_body_size 100M;

    location /static/ {
        alias {{ static_path }}/;
        expires 30d;
    }
```

```
    location /media/ {
        alias {{ media_path }}/;
        expires 7d;
    }

    location / {
        proxy_pass http://{{ project_name }};
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    access_log {{ logs_path }}/nginx_access.log;
    error_log {{ logs_path }}/nginx_error.log;
}
```

Шаг 10. Роль для переиспользования

Файл ansible/roles/common/tasks/main.yml:

```
---
- name: Update apt cache
  apt:
    update_cache: yes
    cache_valid_time: 3600

- name: Install common packages
  apt:
    name:
```

```
      - curl
      - wget
      - git
      - htop
      - vim
      - tree
      - net-tools
    state: present

- name: Set hostname
  hostname:
    name: "{{ hostname | default(inventory_hostname) }}"

- name: Configure timezone
  timezone:
    name: "{{ timezone }}"

- name: Create swap file
  command: fallocate -l 2G /swapfile
  args:
    creates: /swapfile

- name: Set swap permissions
  file:
    path: /swapfile
    mode: '0600'

- name: Make swap
  command: mkswap /swapfile
```

```
  when: not ansible_swaptotal_mb > 0

- name: Enable swap
  command: swapon /swapfile
  when: not ansible_swaptotal_mb > 0

- name: Add swap to fstab
  lineinfile:
    path: /etc/fstab
    line: '/swapfile none swap sw 0 0'
    state: present
```

Шаг 11. Запуск плейбуков

```
# Проверка подключения к хостам
ansible all -i inventory.yml -m ping

# Вывод:
# vps1 | SUCCESS => {
#     "changed": false,
#     "ping": "pong"
# }

# Проверка конкретного хоста
ansible vps1 -i inventory.yml -m ping

# Просмотр фактов (информации о системе)
ansible vps1 -i inventory.yml -m setup | less
```

```
# Запуск базовой настройки сервера
ansible-playbook -i inventory.yml playbooks/setup-server.yml

# Запуск установки Docker
ansible-playbook -i inventory.yml playbooks/deploy-docker.yml

# Запуск развертывания приложения
ansible-playbook -i inventory.yml playbooks/deploy-app.yml

# Запуск конкретного плейбука с ограничением по хостам
ansible-playbook -i inventory.yml playbooks/deploy-app.yml --limit staging

# Запуск с дополнительными переменными
ansible-playbook -i inventory.yml playbooks/deploy-app.yml -e "git_branch=develop"

# Проверка синтаксиса
ansible-playbook -i inventory.yml playbooks/deploy-app.yml --syntax-check

# Пробный запуск (dry-run)
ansible-playbook -i inventory.yml playbooks/deploy-app.yml --check

# Подробный вывод
ansible-playbook -i inventory.yml playbooks/deploy-app.yml -vvv
```

4. Продвинутые возможности Ansible

4.1. Использование ansible-vault для секретов

```
# Создание зашифрованного файла
```

```
ansible-vault create secrets.yml

# Редактирование зашифрованного файла
ansible-vault edit secrets.yml

# Просмотр зашифрованного файла
ansible-vault view secrets.yml

# Запуск плейбука с шифрованными данными
ansible-playbook playbook.yml --ask-vault-pass
ansible-playbook playbook.yml --vault-password-file ~/.vault_pass
```

4.2. Динамический инвентарь (AWS, DigitalOcean)

```
// digitalocean_inventory.py (пример)
{
  "digitalocean": {
    "hosts": ["159.89.123.45", "159.89.123.46"],
    "vars": {
      "ansible_user": "root",
      "provider": "digitalocean"
    }
  }
}
```

4.3. Теги для выборочного выполнения

```
- name: Install Docker
  tags: [docker, setup]
  apt:
```

```
    name: docker.io
    state: present

- name: Deploy application
  tags: [deploy, app]
  docker_compose:
    project_src: /home/deploy/myproject
    state: present

# Запуск только задач с тегом docker
ansible-playbook playbook.yml --tags docker

# Запуск всех задач, кроме deploy
ansible-playbook playbook.yml --skip-tags deploy
```

5. Индивидуальные задания

5.1. Базовый уровень (15 вариантов)

Требования: Создать инвентарь и написать плейбук для установки Docker и Docker Compose.

Вариант	Дополнительная настройка
1	Установка Nginx
2	Установка PostgreSQL
3	Установка Redis
4	Создание swap файла
5	Настройка UFW

Вариант	Дополнительная настройка
6	Настройка fail2ban
7	Установка Node.js
8	Установка Python 3.11
9	Установка Poetry
10	Установка Supervisor
11	Установка Certbot
12	Настройка логирования
13	Настройка мониторинга
14	Настройка бэкапов
15	Настройка cron задач

5.2. Средний уровень (15 вариантов)

Требования: Написать плейбук для развертывания Django-приложения с Docker.

Вариант	Дополнительная настройка
1	Развертывание с PostgreSQL
2	Развертывание с Redis
3	Развертывание с Nginx

Вариант	Дополнительная настройка
4	Развертывание с Celery
5	Развертывание с Flower
6	Настройка HTTPS (Let's Encrypt)
7	Настройка бэкапов базы данных
8	Настройка мониторинга (Prometheus)
9	Настройка алертов (Alertmanager)
10	Настройка Grafana
11	Настройка логирования (Loki)
12	Настройка трассировки (Jaeger)
13	Настройка автоматических миграций
14	Настройка сборки статики
15	Настройка автоматического обновления

5.3. Продвинутый уровень (15 вариантов)

Требования: Написать роль для настройки сервера с нуля (базовая конфигурация + Docker + приложение).

Вариант	Тип приложения	Дополнительная настройка
1	Django + PostgreSQL + Nginx	Zero downtime deploy
2	Django + Celery + Redis	Автоматические миграции
3	React + Node.js	CI/CD интеграция
4	Flask + MongoDB + Redis	Аутентификация
5	FastAPI + PostgreSQL + Kafka	Асинхронная обработка
6	Laravel + MySQL + Redis	Очереди
7	Spring Boot + PostgreSQL	Микросервисы
8	Ruby on Rails + PostgreSQL	WebSocket
9	Express + MongoDB	JWT аутентификация
10	Next.js + PostgreSQL	SSR настройка

Вариант	Тип приложения	Дополнительная настройка
11	Nuxt.js + MongoDB	SSR настройка
12	Wordpress + MySQL + Nginx	W3TC настройка
13	Magento + MySQL + Redis	Индексация
14	OpenCart + MySQL + Nginx	SEO настройка
15	Drupal + PostgreSQL + Redis	Кэширование

6. Контрольные вопросы

1. Что такое Infrastructure as Code (IaC)?
2. Что такое Ansible и для чего он используется?
3. Чем Ansible отличается от других инструментов (Terraform, Chef)?
4. Как установить Ansible?
5. Что такое инвентарь (inventory) в Ansible?
6. Как написать простой плейбук?
7. Что такое модули Ansible? Приведите примеры.
8. Как использовать переменные в плейбуках?
9. Как выполнить плейбук на удалённом сервере?
10. Как отлаживать выполнение плейбука?
11. Что такое роли (roles) и как их создавать?

12. Как использовать ansible-vault для шифрования секретов?
13. Как настроить динамический инвентарь?
14. Как использовать теги для выборочного выполнения?
15. Как интегрировать Ansible в CI/CD?

7. Требования к отчёту

Титульный лист (название работы, ФИО, группа, вариант)

Цель работы

Теоретическое обоснование (своими словами)

Выполнение практического примера (инвентарь, плейбук, команды, скриншоты)

Выполнение индивидуального задания (конфигурации, скриншоты)

Ответы на контрольные вопросы (не менее 5 по своему уровню)

Вывод

8. Критерии оценивания

Критерий	Базовый (max 35)	Средний (max 50)	Продвинутый (max 65)
Установка и настройка Ansible	10	10	10
Создание инвентаря	10	10	10
Написание плейбука (установка Docker)	15	10	10
Написание плейбука	—	15	15

Критерий	Базовый (max 35)	Средний (max 50)	Продвинутый (max 65)
(развертывание приложения)			
Создание ролей	—	5	10
Интеграция в CI/CD	—	—	10
Итого	35	50	65

Шкала перевода баллов в оценку:

Оценка	Базовый (35)	Средний (50)	Продвинутый (65)
5 (отлично)	32-35	46-50	59-65
4 (хорошо)	28-31	40-45	51-58
3 (удовлетворительно)	22-27	32-39	42-50
2 (неудовлетворительно)	0-21	0-31	0-41

Лабораторная работа №34

Централизованное логирование и мониторинг

1. Цель работы

Освоить сбор логов и мониторинг production-системы с использованием современных инструментов. Научиться настраивать Loki и Promtail для

централизованного сбора логов, Prometheus для сбора метрик, Grafana для визуализации данных, а также настраивать алерты для оповещения о проблемах.

2. Теоретическое обоснование

2.1. Что такое логирование и мониторинг?

Логирование - это процесс сбора и хранения событий, происходящих в приложении и инфраструктуре (ошибки, предупреждения, информационные сообщения, запросы пользователей).

Мониторинг - это процесс наблюдения за состоянием системы и приложения (использование CPU, памяти, диска, количество запросов, время ответа, ошибки).

Аналогия: Представьте, что вы управляете заводом:

Логи - это журнал, где записывается каждый чих на заводе (кто зашёл, что сломалось, когда починили).

Метрики - это датчики (температура, давление, скорость конвейера).

Мониторинг - это пульт управления, где вы видите все датчики.

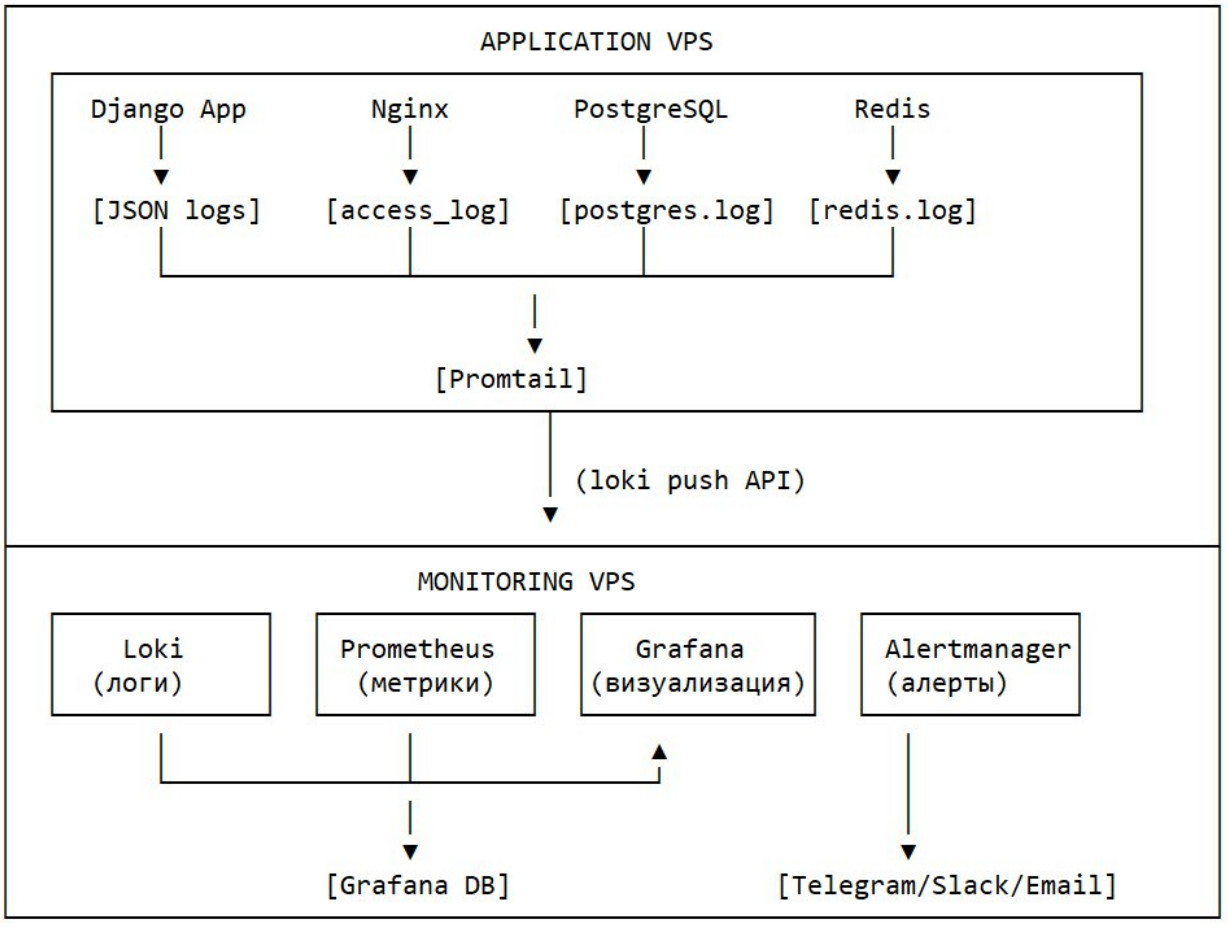
Алерты - это сирена, которая срабатывает, когда что-то идёт не так.

2.2. Стек инструментов (Loki + Promtail + Prometheus + Grafana)

Инструмент	Назначение	Аналогия
Promtail	Сбор логов с серверов и отправка в Loki	Сборщик данных с датчиков
Loki	Хранение и индексация логов	База данных для

Инструмент	Назначение	Аналогия
		журналов
Prometheus	Сбор и хранение метрик	Система сбора показателей
Grafana	Визуализация данных (дашборды)	Пульт управления

2.3. Архитектура системы мониторинга



2.4. Структурированные логи

Неструктурированный лог (плохо):

2024-01-15 10:30:45 ERROR User 123 failed to login

Структурированный лог в JSON (хорошо):

```
{
  "timestamp": "2024-01-15T10:30:45Z",
  "level": "error",
  "event": "login_failed",
  "user_id": 123,
  "ip": "192.168.1.1",
  "message": "Invalid password"
}
```

Преимущества структурированных логов:

1. Легко парсить и фильтровать
2. Можно добавлять любые поля
3. Удобно искать в Loki
4. Можно строить дашборды на основе полей

2.5. Метрики для мониторинга

Тип метрики	Что измеряет	Пример
Counter	Счётчик (только увеличивается)	Количество запросов, ошибок
Gauge	Текущее значение (может увеличиваться/уменьшаться)	Использование памяти, CPU

Тип метрики	Что измеряет	Пример
Histogram	Распределение значений	Время ответа, размер запроса
Summary	Квантили	95-й перцентиль времени ответа

3. Практический пример

Что мы будем делать: Настроим централизованный сбор логов и мониторинг для Django-приложения с использованием Loki, Promtail, Prometheus и Grafana.

Шаг 1. Подготовка Django приложения

1.1. Настройка структурированного логирования в Django

Файл backend/myproject/settings.py (дополнение для логирования):

```
import os
import sys
from datetime import datetime
import json
from pythonjsonlogger import jsonlogger

# Установка библиотеки
# pip install python-json-logger

# Настройки логирования
LOGGING = {
    'version': 1,
```

```
        'disable_existing_loggers': False,
        'formatters': {
            'json': {
                '()': 'pythonjsonlogger.jsonlogger.JsonFormatter',
                'format':
'%(asctime)s %(levelname)s %(name)s %(module)s %(funcName)s %(lineno)d %(
message)s',
                'datefmt': '%Y-%m-%dT%H:%M:%SZ',
            },
            'verbose': {
                'format': '{levelname} {asctime} {module} {process:d} {thread:d}
{message}',
                'style': '{',
            },
        },
        'handlers': {
            'console': {
                'class': 'logging.StreamHandler',
                'formatter': 'json',
                'stream': sys stdout,
            },
            'file': {
                'class': 'logging.handlers.RotatingFileHandler',
                'filename': '/var/log/myproject/django.log',
                'maxBytes': 1024 * 1024 * 10,  # 10 MB
                'backupCount': 5,
                'formatter': 'json',
            },
        },
```

```
    'loggers': {
        'django': {
            'handlers': ['console', 'file'],
            'level': 'INFO',
            'propagate': False,
        },
        'django.request': {
            'handlers': ['console', 'file'],
            'level': 'INFO',
            'propagate': False,
        },
        'api': {
            'handlers': ['console', 'file'],
            'level': 'DEBUG',
            'propagate': False,
        },
    },
    'root': {
        'handlers': ['console', 'file'],
        'level': 'INFO',
    },
}

# Настройка для сбора метрик через django-prometheus
# pip install django-prometheus

INSTALLED_APPS = [
    'django_prometheus',
    # ... другие приложения
```

```
]

MIDDLEWARE = [
    'django_prometheus.middleware.PrometheusBeforeMiddleware',
    # ... другие middleware
    'django_prometheus.middleware.PrometheusAfterMiddleware',
]

# URL для метрик Prometheus
urlpatterns = [
    path('', include('django_prometheus.urls')),
    # ... другие URL
]
```

1.2. Сбор кастомных метрик

Файл backend/api/metrics.py:

```
from prometheus_client import Counter, Histogram, Gauge, Summary
import time

# Счётчики
news_created_counter = Counter('news_created_total', 'Total number of news created', ['category'])
news_viewed_counter = Counter('news_viewed_total', 'Total number of news views', ['category'])
login_attempts_counter = Counter('login_attempts_total', 'Total login attempts', ['status'])

# Гистограммы
```

```
request_duration_histogram = Histogram(
    'http_request_duration_seconds',
    'HTTP request duration in seconds',
    ['method', 'endpoint'],
    buckets=[0.1, 0.5, 1, 2, 5, 10]
)

# Сводки (Summary)
api_response_time = Summary('api_response_time_seconds', 'API response time', ['endpoint'])

# Измерители (Gauge)
active_users_gauge = Gauge('active_users', 'Number of active users')
db_connections_gauge = Gauge('db_connections', 'Database connections')

# Декоратор для измерения времени выполнения
def measure_time(endpoint):
    def decorator(func):
        def wrapper(*args, **kwargs):
            start = time.time()
            result = func(*args, **kwargs)
            duration = time.time() - start
            api_response_time.labels(endpoint=endpoint).observe(duration)
            return result
        return wrapper
    return decorator
```

Использование в views.py:

python

```
from . metrics import news_created_counter, request_duration_histogram,
measure_time

class NewsCreateView(generics CreateAPIView):
    def perform_create(self, serializer):

news_created_counter labels(category=serializer validated_data['category'] name) i
nc()
        serializer save()

@measure_time('news_list')
def news_list(request):
    start = time time()
    # ... логика
    request_duration_histogram labels(method='GET',
endpoint='/api/news/') observe(time time() - start)
```

Шаг 2. Настройка Loki и Promtail с Docker Compose

Файл docker-compose.monitoring.yml:

```
version: '3.8'

services:
  # ========== LOKI - хранение логов ==========
  loki:
    image: grafana/loki latest
    container_name: monitoring-loki
    restart: unless stopped
    ports:
```

```
    - "3100:3100"
  volumes:
    - ./monitoring/loki /loki
    - ./monitoring/loki config.yaml /etc/loki/loki config.yaml
  command: -config.file=/etc/loki/loki config.yaml
  networks:
    - monitoring network

# ========== PROMTAIL - сбор логов ==========
promtail:
  image: grafana/promtail latest
  container_name: monitoring-promtail
  restart: unless stopped
  volumes:
    - /var/log /var/log
    - /var/lib/docker/containers /var/lib/docker/containers ro
    - ./monitoring/promtail config.yaml /etc/promtail/promtail config.yaml
  command: -config.file=/etc/promtail/promtail config.yaml
  depends_on:
    - loki
  networks:
    - monitoring network

# ========== PROMETHEUS - сбор метрик ==========
prometheus:
  image: prom/prometheus latest
  container_name: monitoring-prometheus
  restart: unless stopped
  ports:
```

```
    - "9090:9090"
  volumes:
    - ./monitoring/prometheus.yml /etc/prometheus/prometheus.yml
    - prometheus_data /prometheus
  command:
    - '--config.file=/etc/prometheus/prometheus.yml'
    - '--storage.tsdb.path=/prometheus'
    - '--web.enable-lifecycle'
  networks:
    - monitoring_network

# ========== GRAFANA - визуализация ==========
grafana:
  image: grafana/grafana:latest
  container_name: monitoring_grafana
  restart: unless-stopped
  ports:
    - "3000:3000"
  environment:
    - GF_SECURITY_ADMIN_USER=admin
    - GF_SECURITY_ADMIN_PASSWORD=admin123
    - GF_INSTALL_PLUGINS=grafana-piechart-panel
  volumes:
    - grafana_data:/var/lib/grafana
    - ./monitoring/grafana/provisioning:/etc/grafana/provisioning
  depends_on:
    - prometheus
    - loki
  networks:
```

```
      - monitoring network

    # ========== ALERTMANAGER - управление алертами ==========
    alertmanager:
      image: prom/alertmanager latest
      container_name: monitoring-alertmanager
      restart: unless-stopped
      ports:
        - "9093:9093"
      volumes:
        - ./monitoring/alertmanager.yml:/etc/alertmanager/alertmanager.yml
      command:
        - '--config.file=/etc/alertmanager/alertmanager.yml'
      networks:
        - monitoring network

  volumes:
    prometheus_data:
    grafana_data:

  networks:
    monitoring-network:
      driver: bridge
```

Шаг 3. Конфигурационные файлы

Файл monitoring/loki-config.yaml:

```
auth_enabled: false
```

```yaml
server:
  http_listen_port: 3100

common:
  ring:
    instance_addr: 127.0.0.1
    kvstore:
      store: inmemory
  replication_factor: 1
  path_prefix: /loki

schema_config:
  configs:
    - from: 2020-10-24
      store: boltdb-shipper
      object_store: filesystem
      schema: v11
      index:
        prefix: index_
        period: 24h

storage_config:
  boltdb_shipper:
    active_index_directory: /loki/boltdb-shipper-active
    cache_location: /loki/boltdb-shipper-cache
    cache_ttl: 24h
    shared_store: filesystem
  filesystem:
```

```
    directory: /loki/chunks

compactor:
  working_directory: /loki/compactor
  shared_store: filesystem

limits_config:
  retention_period: 720h  # 30 дней
  reject_old_samples: true
  reject_old_samples_max_age: 168h

table_manager:
  retention_deletes_enabled: true
  retention_period: 720h
```

Файл monitoring/promtail-config.yaml:

```
server:
  http_listen_port: 9080
  grpc_listen_port: 0

positions:
  filename: /tmp/positions.yaml

clients:
  - url: http //loki 3100/loki/api/v1/push

scrape_configs:
  # Django приложение (JSON логи)
  - job_name: django
```

```
    static_configs:
      - targets:
          - localhost
        labels:
          job: django
          host: vps
          app: myproject
          __path__: /var/log/myproject/*.log
    pipeline_stages:
      - json:
          expressions:
            level: level
            message: message
            timestamp: asctime
            module: name
            funcName: funcName
      - labels:
          level: level
          module: module

  # Nginx логи (access log)
  - job_name: nginx_access
    static_configs:
      - targets:
          - localhost
        labels:
          job: nginx
          type: access
          app: myproject
```

```
      __path__: /var/log/nginx/access.log
    pipeline_stages:
      - regex:
          expression: '^(?P<ip>\S+) - - \[(?P<timestamp>[^\]]+)\] "(?P<method>\S+) (?P<path>[^"]+) (?P<protocol>\S+)" (?P<status>\d+) (?P<size>\d+) "(?P<referrer>[^"]*)" "(?P<user_agent>[^"]*)"'
      - labels:
          method: method
          status: status
      - timestamp:
          source: timestamp
          format: "02/Jan/2006:15:04:05 -0700"

  # Nginx ошибки (error log)
  - job_name: nginx-error
    static_configs:
      - targets:
          - localhost
        labels:
          job: nginx
          type: error
          app: myproject
          __path__: /var/log/nginx/error.log

  # Docker контейнеры
  - job_name: docker
    docker_sd_configs:
      - host: unix:///var/run/docker.sock
        refresh_interval: 10s
```

```
    relabel_configs:
      - source_labels: ['__meta_docker_container_name']
        regex: '/(.*)'
        target_label: 'container'
      - source_labels: ['__meta_docker_container_label_com_docker_compose_project']
        target_label: 'compose_project'
    pipeline_stages:
      - json:
          expressions:
            log: log
            stream: stream
      - labels:
          stream: stream
```

Файл monitoring/prometheus.yml:

```
global:
  scrape_interval: 15s
  evaluation_interval: 15s
  external_labels:
    monitor: 'myproject'

alerting:
  alertmanagers:
    - static_configs:
        - targets:
            - alertmanager:9093
```

```yaml
rule_files:
  - "alerts.yml"

scrape_configs:
  # Сам Prometheus
  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090']

  # Django приложение (через django-prometheus)
  - job_name: 'django'
    metrics_path: '/metrics'
    static_configs:
      - targets: ['backend:8000']
        labels:
          app: 'myproject'
          service: 'django'

  # PostgreSQL экспортёр
  - job_name: 'postgresql'
    static_configs:
      - targets: ['postgres-exporter:9187']
        labels:
          app: 'myproject'
          service: 'postgresql'

  # Redis экспортёр
  - job_name: 'redis'
    static_configs:
```

```
    - targets: ['redis-exporter:9121']
      labels:
        app: 'myproject'
        service: 'redis'

  # Node экспортёр (системные метрики сервера)
  - job_name: 'node'
    static_configs:
      - targets: ['node-exporter:9100']
        labels:
          app: 'myproject'
          service: 'node'

  # Nginx экспортёр
  - job_name: 'nginx'
    static_configs:
      - targets: ['nginx-exporter:9113']
        labels:
          app: 'myproject'
          service: 'nginx'
```

Файл monitoring/alerts.yml (правила алертов):

```
groups:
  - name: django_alerts
    interval: 30s
    rules:
      # Высокая частота 5xx ошибок
      - alert: High5xxErrorRate
```

```
      expr: rate(django_http_responses_total_by_status_view_method{status=~"5.."}[5m]) > 0.05
      for: 2m
      labels:
        severity: critical
      annotations:
        summary: "High 5xx error rate"
        description: "5xx error rate is {{ $value }} errors per second"

    # Медленные запросы
    - alert: SlowRequests
      expr: histogram_quantile(0.95, rate(django_http_requests_latency_seconds_bucket[5m])) > 1
      for: 5m
      labels:
        severity: warning
      annotations:
        summary: "Slow HTTP requests"
        description: "95th percentile response time is {{ $value }}s"

    # Много активных пользователей
    - alert: HighActiveUsers
      expr: active_users > 1000
      for: 5m
      labels:
        severity: info
      annotations:
        summary: "High number of active users"
```

```
      description: "Active users count is {{ $value }}"

  - name: infrastructure_alerts
    interval: 30s
    rules:
      # Высокая загрузка CPU
      - alert: HighCPUUsage
        expr: 100 - (avg by (instance) (rate(node_cpu_seconds_total{mode="idle"}[5m])) * 100) > 80
        for: 5m
        labels:
          severity: warning
        annotations:
          summary: "High CPU usage"
          description: "CPU usage is {{ $value }}%"

      # Мало свободной памяти
      - alert: LowMemory
        expr: (1 - (node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes)) * 100 > 90
        for: 5m
        labels:
          severity: critical
        annotations:
          summary: "Low memory"
          description: "Memory usage is {{ $value }}%"

      # Заканчивается место на диске
      - alert: LowDiskSpace
```

```
      expr: (node_filesystem_free_bytes{mountpoint="/"} /
node_filesystem_size_bytes{mountpoint="/"}) * 100 < 10
      for: 5m
      labels:
        severity: critical
      annotations:
        summary: "Low disk space"
        description: "Disk space is {{ $value }}% free"

    # Отключенный экспортёр
    - alert: TargetDown
      expr: up == 0
      for: 1m
      labels:
        severity: critical
      annotations:
        summary: "Target down"
        description: "{{ $labels.job }} target is down"

    # Docker контейнер не работает
    - alert: ContainerDown
      expr: container_last_seen{container!=""} == 0
      for: 1m
      labels:
        severity: critical
      annotations:
        summary: "Container down"
        description: "Container {{ $labels.container }} is not running"
```

Файл monitoring/alertmanager.yml:

```yaml
global:
  resolve_timeout: 5m
  slack_api_url: 'https://hooks.slack.com/services/...'

route:
  group_by: ['alertname', 'cluster', 'service']
  group_wait: 10s
  group_interval: 10s
  repeat_interval: 1h
  receiver: 'default-receiver'
  routes:
    - match:
        severity: critical
      receiver: critical-receiver
      continue: true
    - match:
        severity: warning
      receiver: warning-receiver

receivers:
  - name: 'default-receiver'
    slack_configs:
      - channel: '#alerts'
        title: 'Alert: {{ .GroupLabels.alertname }}'
        text: '{{ range .Alerts }}{{ .Annotations.description }}{{ end }}'

  - name: 'critical-receiver'
    slack_configs:
```

```
      - channel: '#critical'
        title: '🚨 CRITICAL: {{ .GroupLabels.alertname }}'
        text: '{{ range .Alerts }}{{ .Annotations.description }}{{ end }}'
    email_configs:
      - to: 'admin@example.com'
        subject: 'CRITICAL: {{ .GroupLabels.alertname }}'

  - name: 'warning-receiver'
    slack_configs:
      - channel: '#warnings'
        title: '⚠️ WARNING: {{ .GroupLabels.alertname }}'
        text: '{{ range .Alerts }}{{ .Annotations.description }}{{ end }}'
```

Шаг 4. Настройка Grafana

Файл monitoring/grafana/provisioning/datasources/datasources.yml:

```
apiVersion: 1

datasources:
  - name: Prometheus
    type: prometheus
    access: proxy
    url: http://prometheus:9090
    isDefault: true
    editable: true
    jsonData:
      timeInterval: 15s

  - name: Loki
    type: loki
    access: proxy
```

```
    url: http://loki:3100
    editable: true
    jsonData:
      maxLines: 1000
```

Файл monitoring/grafana/provisioning/dashboards/dashboards.yml:

```
apiVersion: 1

providers:
  - name: 'default'
    orgId: 1
    folder: ''
    type: file
    disableDeletion: false
    editable: true
    options:
      path: /etc/grafana/provisioning/dashboards
```

Шаг 5. Запуск системы мониторинга

```
# Запуск всех сервисов
docker compose -f docker-compose.monitoring.yml up -d

# Просмотр логов
docker compose -f docker-compose.monitoring.yml logs -f

# Проверка статуса
docker compose -f docker-compose.monitoring.yml ps

# Проверка доступности Loki
```

```
curl http://localhost:3100/ready

# Проверка Prometheus
curl http://localhost:9090/api/v1/query?query=up

# Открытие Grafana в браузере
# http://localhost:3000
# Логин: admin
# Пароль: admin123
```

Шаг 6. Создание дашбордов в Grafana

6.1. Дашборд для Django приложения (JSON)

```
{
  "dashboard": {
    "title": "Django Monitoring",
    "panels": [
      {
        "title": "Request Rate",
        "type": "graph",
        "targets": [
          {
            "expr": "rate(django_http_requests_total[5m])",
            "legendFormat": "{{ method }} {{ view }}"
          }
        ]
      },
      {
        "title": "Response Time (95th percentile)",
        "type": "graph",
```

```
            "targets": [
              {
                "expr": "histogram_quantile(0.95, rate(django_http_requests_latency_seconds_bucket[5m]))",
                "legendFormat": "{{ method }} {{ view }}"
              }
            ]
          },
          {
            "title": "Error Rate (5xx)",
            "type": "graph",
            "targets": [
              {
                "expr": "rate(django_http_responses_total_by_status_view_method{status=~\"5..\"}[5m])",
                "legendFormat": "{{ method }} {{ view }}"
              }
            ]
          },
          {
            "title": "Active Users",
            "type": "stat",
            "targets": [
              {
                "expr": "active_users"
              }
            ]
          },
```

```
    {
      "title": "Database Connections",
      "type": "gauge",
      "targets": [
        {
          "expr": "db_connections"
        }
      ]
    }
  ]
 }
}
```

6.2. Дашборд для системных метрик

```
{
 "dashboard": {
  "title": "System Monitoring",
  "panels": [
    {
      "title": "CPU Usage",
      "type": "graph",
      "targets": [
        {
          "expr": "100 - (avg by (instance) (rate(node_cpu_seconds_total{mode=\"idle\"}[5m])) * 100)",
          "legendFormat": "{{ instance }}"
        }
      ]
    },
```

```
        {
          "title": "Memory Usage",
          "type": "graph",
          "targets": [
            {
              "expr": "(1 - (node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes)) * 100",
              "legendFormat": "{{ instance }}"
            }
          ]
        },
        {
          "title": "Disk Usage",
          "type": "graph",
          "targets": [
            {
              "expr": "(node_filesystem_size_bytes{fstype!=\"\"} - node_filesystem_free_bytes{fstype!=\"\"}) / node_filesystem_size_bytes{fstype!=\"\"} * 100",
              "legendFormat": "{{ mountpoint }}"
            }
          ]
        },
        {
          "title": "Network Traffic",
          "type": "graph",
          "targets": [
            {
              "expr": "rate(node_network_receive_bytes_total[5m])",
```

```
        "legendFormat": "{{ device }} rx"
      },
      {
        "expr": "rate(node_network_transmit_bytes_total[5m])",
        "legendFormat": "{{ device }} tx"
      }
    ]
  }
 ]
}
```

6.3. Поиск по логам в Grafana (Loki)

```
# Все ошибки Django
{job="django"} |= "ERROR"

# Ошибки с уровнем critical
{job="django"} | json | level="CRITICAL"

# Поиск по IP в логах Nginx
{job="nginx"} |= "192.168.1.1"

# Логи за последний час с ошибками 500
{job="nginx"} |~ " 500 "

# Ошибки аутентификации
{job="django"} | json | message =~ "login.*failed"

# Подсчёт ошибок по модулям
```

```
count by (module) ({job="django"} | json | level="ERROR")

# Поиск по временному диапазону
{job="nginx"} | json | status="404" | line_format "{{.remote_addr}} - {{.request}}"
```

Шаг 7. Интеграция с приложением

```
# Добавление кастомных метрик в Django
from prometheus_client import Counter, Histogram
import time

# Метрики для отслеживания бизнес-показателей
news_by_category = Counter('news_by_category', 'News created by category', ['category'])
comment_rating_histogram = Histogram('comment_rating', 'Comment rating distribution', buckets=[1, 2, 3, 4, 5])

def create_news(request):
    # ... логика создания
    news_by_category.labels(category=category_name).inc()

def add_comment(request):
    rating = request.POST.get('rating')
    comment_rating_histogram.observe(int(rating))
```

4. Индивидуальные задания

4.1. Базовый уровень (15 вариантов)

Требования: Настроить сбор логов через Loki и Promtail для одного сервиса.

Вариант	Источник логов	Формат логов
1	Django приложение	JSON
2	Nginx access log	Combined log format
3	Nginx error log	Plain text
4	PostgreSQL	CSV log
5	Redis	Plain text
6	Celery worker	JSON
7	Gunicorn access log	Combined log format
8	Gunicorn error log	Plain text
9	Docker контейнеры	JSON
10	Systemd journal	Journald
11	Apache access log	Combined log format
12	MySQL	General log
13	MongoDB	JSON
14	RabbitMQ	Plain text

Вариант	Источник логов	Формат логов
15	Elasticsearch	JSON

4.2. Средний уровень (15 вариантов)

Требования: Настроить сбор метрик через Prometheus и визуализацию в Grafana.

Вариант	Источник метрик	Дашборд
1	Django (django-prometheus)	Request rate, latency, errors
2	PostgreSQL (postgres-exporter)	Connections, queries, locks
3	Redis (redis-exporter)	Memory, hits/misses, clients
4	Nginx (nginx-exporter)	Requests, connections, cache
5	Node (node-exporter)	CPU, memory, disk, network
6	Docker (cAdvisor)	Container CPU, memory, network
7	MySQL (mysqld-exporter)	Queries, connections, slow queries

Вариант	Источник метрик	Дашборд
8	MongoDB (mongodb-exporter)	Operations, connections, memory
9	RabbitMQ (rabbitmq-exporter)	Queues, messages, consumers
10	Elasticsearch (elasticsearch-exporter)	Indices, searches, memory
11	Kafka (kafka-exporter)	Topics, partitions, lag
12	HAProxy (haproxy-exporter)	Backends, sessions, errors
13	Cloudflare (cloudflare-exporter)	Traffic, threats, cache
14	AWS (cloudwatch-exporter)	EC2, RDS, S3 метрики
15	GitHub (github-exporter)	Repositories, stars, forks

4.3. Продвинутый уровень (15 вариантов)

Требования: Настроить алерты и интеграцию с мессенджерами.

Вариант	Канал уведомлений	Условия алертов
1	Telegram	High CPU, low memory, high 5xx rate
2	Slack	High error rate, slow requests, down services
3	Discord	High response time, low disk space
4	Email	Target down, high 5xx rate
5	PagerDuty	Critical alerts
6	Opsgenie	High severity alerts
7	Webhook	Custom integration
8	Microsoft Teams	Infrastructure alerts
9	Google Chat	Application alerts
10	Rocket.Chat	Security alerts
11	Matter	Performance alerts

Вариант	Канал уведомлений	Условия алертов
	most	
12	Zulip	Business metrics alerts
13	Pushover	Critical alerts only
14	Gotify	Self-hosted notifications
15	Apprise	Multi-channel notifications

5. Контрольные вопросы

1. Что такое логирование и мониторинг?
2. Чем отличаются логи от метрик?
3. Что такое Loki и Promtail?
4. Что такое Prometheus?
5. Что такое Grafana?
6. Что такое структурированные логи и зачем они нужны?
7. Как настроить сбор логов Django в JSON формате?
8. Как добавить кастомные метрики в Django?
9. Как визуализировать данные в Grafana?
10. Как настроить алерты в Alertmanager?
11. Как настроить сбор логов из Docker контейнеров?
12. Как настроить агрегацию метрик (rate, increase, histogram_quantile)?
13. Как настроить дашборд с несколькими источниками данных?

14. Как настроить автоматическое обновление дашбордов?

15. Как настроить долгосрочное хранение метрик (Thanos, Cortex)?

6. Требования к отчёту

Титульный лист (название работы, ФИО, группа, вариант)

Цель работы

Теоретическое обоснование (своими словами)

Выполнение практического примера (конфигурации, команды, скриншоты дашбордов)

Выполнение индивидуального задания (конфигурации, скриншоты)

Ответы на контрольные вопросы (не менее 5 по своему уровню)

Вывод

7. Критерии оценивания

Критерий	Базовый (max 35)	Средний (max 50)	Продвинутый (max 65)
Настройка Loki и Promtail	15	10	10
Настройка Prometheus	10	10	10
Настройка Grafana	10	10	10
Создание дашбордов	—	15	10
Настройка	—	5	15

Критерий	Базовый (max 35)	Средний (max 50)	Продвинутый (max 65)
алертов			
Интеграция с приложением	—	—	10
Итого	35	50	65

Шкала перевода баллов в оценку:

Оценка	Базовый (35)	Средний (50)	Продвинутый (65)
5 (отлично)	32-35	46-50	59-65
4 (хорошо)	28-31	40-45	51-58
3 (удовлетворительно)	22-27	32-39	42-50
2 (неудовлетворительно)	0-21	0-31	0-41

Лабораторная работа №35

Настройка резервного копирования и отказоустойчивости

1. Цель работы

Научиться настраивать резервное копирование (бэкапы) для баз данных и медиа-файлов, а также восстанавливать данные из бэкапов. Освоить

автоматизацию бэкапов с помощью cron, настройку ротации бэкапов и интеграцию с облачными хранилищами (S3).

2. Теоретическое обоснование

2.1. Что такое резервное копирование и зачем оно нужно?

Резервное копирование (бэкап) - это процесс создания копии данных, которую можно использовать для восстановления в случае их потери или повреждения.

Аналогия: Представьте, что вы пишете книгу. Вы делаете копии черновиков и храните их в разных местах (на флешке, в облаке, у друга). Если оригинал потеряется или испортится, у вас есть копия, и вы можете продолжить работу.

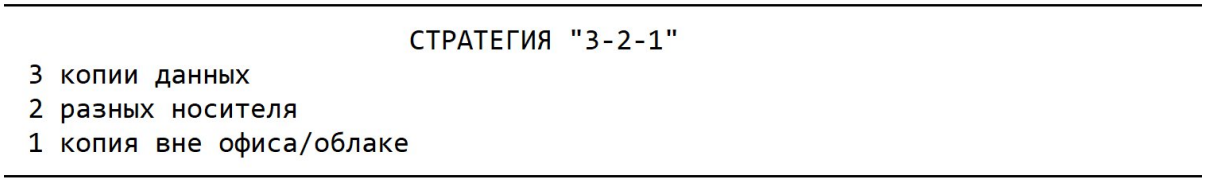
Что может случиться с данными:

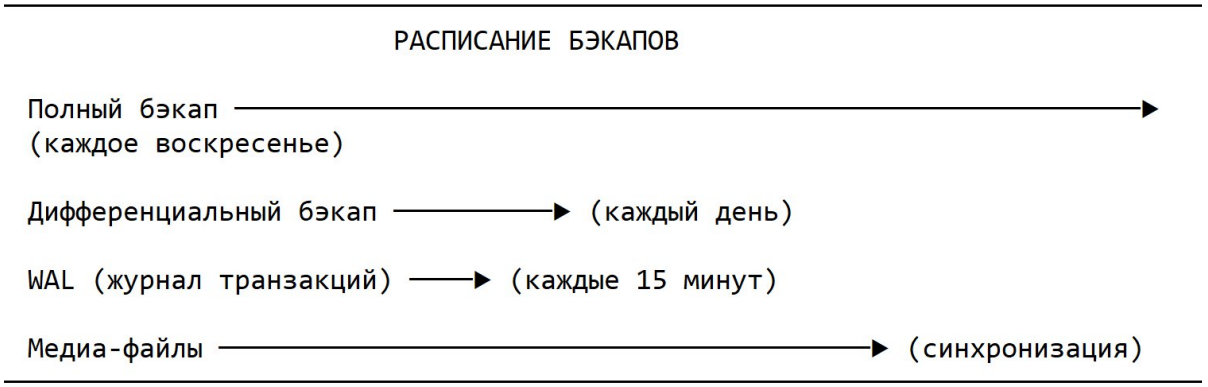
Угроза	Пример	Решение
Аппаратный сбой	Сломался жёсткий диск	Бэкап на другой диск/S3
Ошибка человека	Случайно удалил таблицу	Бэкап перед удалением
Атака (рансомваре)	Зашифровали все файлы	Бэкап на отдельном хранилище
Ошибка в коде	Баг испортил данные	Бэкап до деплоя
Природная катастрофа	Пожар в дата-центре	Бэкап в другом регионе

2.2. Типы бэкапов

Тип	Описание	Плюсы	Минусы	Когда использовать
Полный (Full)	Копия всех данных	Простое восстановление	Много места, долго	Раз в неделю/месяц
Инкрементальный	Только изменения с прошлого бэкапа	Экономит место и время	Сложное восстановление	Раз в день
Дифференциальный	Только изменения с последнего полного бэкапа	Баланс места и времени	Больше места, чем инкремент	Раз в день

2.3. Стратегии бэкапа для приложения





2.4. Что нужно бэкапить

Компонент	Что бэкапить	Частота	Хранение
PostgreSQL	Дамп базы данных (pg_dump)	Ежеднев но	S3 + локально
Медиа-файлы	Папка media/	Постоянно	S3 (синхронизация)
Статические файлы	Папка static/	Редко (при изменениях)	S3
Конфигурация	.env, настройки Nginx	При изменениях	Git + S3
Логи	/var/log/myproject	Ежедневно	S3 (опционально)

3. Практический пример (пошагово, для новичков)

Что мы будем делать: Настроим автоматическое резервное копирование PostgreSQL базы данных, медиа-файлов, настроим ротацию бэкапов и восстановление данных.

Шаг 1. Бэкап PostgreSQL с pg_dump

1.1. Ручной бэкап

```
# Подключение к серверу
ssh deploy@your-vps-ip

# Бэкап всей базы данных
pg_dump -U myproject_user -h localhost myproject_db > backup_$(date +%Y%m%d_%H%M%S).sql

# Бэкап с сжатием
pg_dump -U myproject_user -h localhost myproject_db | gzip > backup_$(date +%Y%m%d_%H%M%S).sql.gz

# Бэкап только схемы (без данных)
pg_dump -U myproject_user -h localhost --schema-only myproject_db > schema_$(date +%Y%m%d).sql

# Бэкап только данных (без схемы)
pg_dump -U myproject_user -h localhost --data-only myproject_db > data_$(date +%Y%m%d).sql

# Бэкап определённой таблицы
pg_dump -U myproject_user -h localhost --table=api_news myproject_db > news_backup.sql
```

```
# Бэкап с использованием .pgpass (без ввода пароля)
# Создать файл ~/.pgpass с содержимым:
# localhost:5432:myproject_db:myproject_user:password
# chmod 600 ~/.pgpass
```

1.2. Скрипт автоматического бэкапа

Файл /home/deploy/scripts/backup_db.sh:

```
#!/bin/bash

# Конфигурация
set -e  # Остановка при ошибке

# Переменные
BACKUP_DIR="/home/deploy/backups/postgres"
DATE=$(date +%Y%m%d_%H%M%S)
DB_NAME="myproject_db"
DB_USER="myproject_user"
DB_HOST="localhost"
DB_PORT="5432"
RETENTION_DAYS=30
S3_BUCKET="s3://myproject-backups/postgres"

# Создание директории для бэкапов
mkdir -p $BACKUP_DIR

echo "==========================================="
echo "Starting PostgreSQL backup at $(date)"
echo "==========================================="
```

```
# 1. Полный бэкап (сжатый)
echo "Creating full backup..."
pg_dump -U $DB_USER -h $DB_HOST -p $DB_PORT -d $DB_NAME \
    --format=custom \
    --file="$BACKUP_DIR/${DB_NAME}_full_${DATE}.dump"

# 2. SQL-дамп (текстовый)
echo "Creating SQL dump..."
pg_dump -U $DB_USER -h $DB_HOST -p $DB_PORT -d $DB_NAME \
    --clean \
    --if-exists \
    --no-owner \
    > "$BACKUP_DIR/${DB_NAME}_sql_${DATE}.sql"

# 3. Сжатие SQL-дампа
echo "Compressing SQL dump..."
gzip -f "$BACKUP_DIR/${DB_NAME}_sql_${DATE}.sql"

# 4. Бэкап только схемы (раз в неделю, в воскресенье)
if [ $(date +%u) -eq 7 ]; then
    echo "Creating schema-only backup (weekly)..."
    pg_dump -U $DB_USER -h $DB_HOST -p $DB_PORT -d $DB_NAME
\

        --schema-only \
        --no-owner \
        > "$BACKUP_DIR/${DB_NAME}_schema_${DATE}.sql"
    gzip -f "$BACKUP_DIR/${DB_NAME}_schema_${DATE}.sql"
fi
```

```
    # 5. Создание контрольной суммы (для проверки целостности)
    echo "Creating checksum..."
    md5sum "$BACKUP_DIR/${DB_NAME}_full_${DATE}.dump" > "$BACKUP_DIR/${DB_NAME}_full_${DATE}.dump.md5"
    md5sum "$BACKUP_DIR/${DB_NAME}_sql_${DATE}.sql.gz" > "$BACKUP_DIR/${DB_NAME}_sql_${DATE}.sql.gz.md5"

    # 6. Загрузка в S3 (если настроен)
    if command -v aws &> /dev/null; then
        echo "Uploading to S3..."
        aws s3 sync $BACKUP_DIR $S3_BUCKET --exclude "*.md5" --include "*.dump" --include "*.sql.gz"
        aws s3 cp "$BACKUP_DIR/${DB_NAME}_full_${DATE}.dump.md5" $S3_BUCKET/
        aws s3 cp "$BACKUP_DIR/${DB_NAME}_sql_${DATE}.sql.gz.md5" $S3_BUCKET/
    fi

    # 7. Ротация старых бэкапов
    echo "Rotating old backups (older than $RETENTION_DAYS days)..."
    find $BACKUP_DIR -name "*.dump" -mtime +$RETENTION_DAYS -delete
    find $BACKUP_DIR -name "*.sql" -mtime +$RETENTION_DAYS -delete
    find $BACKUP_DIR -name "*.sql.gz" -mtime +$RETENTION_DAYS -delete
    find $BACKUP_DIR -name "*.md5" -mtime +$RETENTION_DAYS -delete
```

```
# 8. Отчёт о размере
echo "Backup size:"
du -sh $BACKUP_DIR

echo "==========================================="
echo "Backup completed successfully at $(date)"
echo "==========================================="
```

Делаем скрипт исполняемым:

bash

```
chmod +x /home/deploy/scripts/backup_db.sh
```

Шаг 2. Настройка резервного копирования медиа-файлов в S3

2.1. Установка AWS CLI

```
# Установка AWS CLI
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install

# Настройка AWS credentials
aws configure
# AWS Access Key ID: your-access-key
# AWS Secret Access Key: your-secret-key
# Default region name: eu-central-1
# Default output format: json
```

2.2. Скрипт синхронизации медиа-файлов

Файл /home/deploy/scripts/backup_media.sh:

```bash
#!/bin/bash

set -e

# Конфигурация
MEDIA_DIR="/home/deploy/myproject/media"
STATIC_DIR="/home/deploy/myproject/static"
BACKUP_DIR="/home/deploy/backups/media"
DATE=$(date +%Y%m%d_%H%M%S)
S3_BUCKET="s3://myproject-backups/media"
RETENTION_DAYS=30

echo "============================================="
echo "Starting media backup at $(date)"
echo "============================================="

# 1. Создание архива медиа-файлов
echo "Creating media archive..."
tar -czf "$BACKUP_DIR/media_${DATE}.tar.gz" -C "$MEDIA_DIR" . 2>/dev/null || echo "No media files"

# 2. Создание архива статических файлов (если нужно)
echo "Creating static archive..."
tar -czf "$BACKUP_DIR/static_${DATE}.tar.gz" -C "$STATIC_DIR" . 2>/dev/null || echo "No static files"

# 3. Контрольная сумма
```

```
    md5sum "$BACKUP_DIR/media_${DATE}.tar.gz" > "$BACKUP_DIR/media_${DATE}.tar.gz.md5"
    md5sum "$BACKUP_DIR/static_${DATE}.tar.gz" > "$BACKUP_DIR/static_${DATE}.tar.gz.md5"

    # 4. Синхронизация с S3 (aws s3 sync для инкрементального копирования)
    if command -v aws &> /dev/null; then
        echo "Syncing with S3..."

        # Синхронизация медиа-файлов (инкрементально)
        aws s3 sync $MEDIA_DIR $S3_BUCKET/media/ --exclude "*.tmp" --delete

        # Синхронизация архивов
        aws s3 sync $BACKUP_DIR $S3_BUCKET/archives/ --exclude "*.md5"
        aws s3 sync $BACKUP_DIR $S3_BUCKET/archives/ --include "*.md5"
    fi

    # 5. Ротация старых архивов
    find $BACKUP_DIR -name "*.tar.gz" -mtime +$RETENTION_DAYS -delete
    find $BACKUP_DIR -name "*.md5" -mtime +$RETENTION_DAYS -delete

    echo "Media backup completed at $(date)"
```

Шаг 3. Настройка автоматического выполнения через cron

```
# Редактирование crontab
```

```
crontab -e

# Добавить следующие строки:
# PostgreSQL бэкап каждый день в 2:00
0 2 * * * /home/deploy/scripts/backup_db.sh >> /home/deploy/logs/backup_db.log 2>&1

# Медиа бэкап каждый день в 3:00
0 3 * * * /home/deploy/scripts/backup_media.sh >> /home/deploy/logs/backup_media.log 2>&1

# Полный бэкап (база + медиа) по воскресеньям в 4:00
0 4 * * 0 /home/deploy/scripts/backup_full.sh >> /home/deploy/logs/backup_full.log 2>&1

# Очистка старых логов бэкапов (хранить 60 дней)
0 5 * * * find /home/deploy/logs -name "backup_*.log" -mtime +60 -delete
```

Шаг 4. Настройка бэкапа через Docker контейнер

Файл docker-compose.backup.yml:

```
version: '3.8'

services:
  # Бэкап для PostgreSQL
  postgres-backup:
    image: prodrigestivill/postgres-backup-local:latest
    container_name: postgres-backup
    restart: always
```

```
    environment:
      POSTGRES_HOST: postgres
      POSTGRES_DB: ${DB_NAME}
      POSTGRES_USER: ${DB_USER}
      POSTGRES_PASSWORD: ${DB_PASSWORD}
      POSTGRES_EXTRA_OPTS: '-Z6 --schema=public'
      SCHEDULE: '@daily'
      BACKUP_KEEP_DAYS: 30
      BACKUP_KEEP_WEEKS: 4
      BACKUP_KEEP_MONTHS: 6
    volumes:
      - ./backups/postgres:/backups
    networks:
      - app_network

  # Бэкап для Redis
  redis-backup:
    image: schickling/redis-backup:latest
    container_name: redis_backup
    restart: always
    environment:
      REDIS_HOST: redis
      REDIS_PORT: 6379
      REDIS_PASSWORD: ${REDIS_PASSWORD}
      BACKUP_INTERVAL: 3600
    volumes:
      - ./backups/redis:/backup
    networks:
      - app_network
```

```
  # Синхронизация с S3
  s3-sync:
    image: amazon/aws-cli:latest
    container_name: s3-sync
    restart: no
    entrypoint: ""
    command: >
      sh -c "aws s3 sync /backups s3://myproject-backups/ --delete"
    volumes:
      - ./backups:/backups
      - ~/.aws:/root/.aws
    environment:
      - AWS_ACCESS_KEY_ID=${AWS_ACCESS_KEY_ID}
      - AWS_SECRET_ACCESS_KEY=${AWS_SECRET_ACCESS_KEY}
      - AWS_DEFAULT_REGION=${AWS_DEFAULT_REGION}
    networks:
      - app-network
```

Шаг 5. Восстановление данных из бэкапа

5.1. Восстановление PostgreSQL

```
# Скачивание бэкапа из S3
aws s3 cp s3://myproject-backups/postgres/myproject_db_full_20240115_020000.dump ./backup.dump

# Проверка целостности
md5sum -c backup.dump.md5
```

```
# Восстановление из custom формата (.dump)
pg_restore -U myproject_user -h localhost -d myproject_db --clean --if-exists backup.dump

# Восстановление из SQL-дампа
psql -U myproject_user -h localhost -d myproject_db < backup.sql

# Восстановление из сжатого SQL-дампа
gunzip -c backup.sql.gz | psql -U myproject_user -h localhost -d myproject_db

# Восстановление с пересозданием базы данных
dropdb -U myproject_user myproject_db
createdb -U myproject_user myproject_db
pg_restore -U myproject_user -d myproject_db backup.dump
```

5.2. Восстановление медиа-файлов

```
# Скачивание из S3
aws s3 sync s3://myproject-backups/media/ /home/deploy/myproject/media/

# Восстановление из архива
tar -xzf backup/media_20240115_030000.tar.gz -C /home/deploy/myproject/media/

# Восстановление конкретного файла
aws s3 cp s3://myproject-backups/media/images/photo.jpg /home/deploy/myproject/media/images/
```

5.3. Скрипт восстановления

Файл /home/deploy/scripts/restore.sh:

```
#!/bin/bash

set -e

# Конфигурация
BACKUP_FILE=$1
RESTORE_TYPE=$2  # full, media, or config

if [ -z "$BACKUP_FILE" ]; then
    echo "Usage: ./restore.sh <backup_file> [full|media|config]"
    exit 1
fi

echo "============================================"
echo "Starting restoration from $BACKUP_FILE"
echo "============================================"

case $RESTORE_TYPE in
    full)
        echo "Restoring full system..."

        # 1. Остановка приложения
        echo "Stopping application..."
        cd /home/deploy/myproject
        docker compose down
```

```
            # 2. Восстановление базы данных
            echo "Restoring database..."
            docker compose exec -T postgres psql -U myproject_user -c "DROP DATABASE IF EXISTS myproject_db;"
            docker compose exec -T postgres psql -U myproject_user -c "CREATE DATABASE myproject_db;"
            docker compose exec -T postgres pg_restore -U myproject_user -d myproject_db < "$BACKUP_FILE"

            # 3. Восстановление медиа-файлов
            echo "Restoring media files..."
            tar -xzf "$(dirname $BACKUP_FILE)/media_$(basename $BACKUP_FILE .dump).tar.gz" -C /home/deploy/myproject/media/

            # 4. Запуск приложения
            echo "Starting application..."
            docker compose up -d

            echo "Full restoration completed!"
            ;;

        media)
            echo "Restoring media files only..."
            tar -xzf "$BACKUP_FILE" -C /home/deploy/myproject/media/
            echo "Media restoration completed!"
            ;;

        config)
            echo "Restoring configuration..."
```

```
            cp "$BACKUP_FILE" /home/deploy/myproject/.env
            docker compose restart
            echo "Configuration restoration completed!"
            ;;

        *)
            echo "Unknown restore type: $RESTORE_TYPE"
            echo "Available types: full, media, config"
            exit 1
            ;;
    esac
```

Шаг 6. Настройка резервного копирования с помощью Docker Volume

```
    # Создание бэкапа volume
    docker run --rm -v myproject_postgres_data:/data -v $(pwd):/backup alpine tar czf /backup/postgres_data_backup.tar.gz -C /data .

    # Восстановление volume
    docker run --rm -v myproject_postgres_data:/data -v $(pwd):/backup alpine tar xzf /backup/postgres_data_backup.tar.gz -C /data

    # Создание бэкапа всех volumes
    docker run --rm \
      -v myproject_postgres_data:/postgres \
      -v myproject_redis_data:/redis \
      -v myproject_static_volume:/static \
      -v myproject_media_volume:/media \
      -v $(pwd):/backup \
```

```
  alpine tar czf /backup/all_volumes_$(date +%Y%m%d).tar.gz \
  -C / postgres redis static media
```

Шаг 7. Настройка мониторинга бэкапов

```
# Скрипт проверки бэкапов
#!/bin/bash

# Проверка существования бэкапов
BACKUP_DIR="/home/deploy/backups/postgres"
LATEST_BACKUP=$(ls -t $BACKUP_DIR/*.dump | head -1)

if [ -z "$LATEST_BACKUP" ]; then
    echo "ERROR: No backup found!"
    # Отправить уведомление
    curl -X POST -H 'Content-type: application/json' \
        --data '{"text":"🚨 No database backup found!"}' \
        https://hooks.slack.com/services/...
    exit 1
fi

# Проверка возраста бэкапа (не старше 2 дней)
BACKUP_AGE=$(($(date +%s) - $(stat -c %Y $LATEST_BACKUP)))
MAX_AGE=$((48 * 3600))  # 48 часов

if [ $BACKUP_AGE -gt $MAX_AGE ]; then
    echo "WARNING: Latest backup is older than 48 hours"
    curl -X POST -H 'Content-type: application/json' \
        --data '{"text":"⚠ Database backup is older than 48 hours!"}' \
```

```
        https://hooks.slack.com/services/...
    fi

    # Проверка размера бэкапа
    BACKUP_SIZE=$(stat -c %s $LATEST_BACKUP)
    MIN_SIZE=$((1024 * 1024))  # 1 MB

    if [ $BACKUP_SIZE -lt $MIN_SIZE ]; then
        echo "ERROR: Backup file is too small ($BACKUP_SIZE bytes)"
        curl -X POST -H 'Content-type: application/json' \
            --data '{"text":"🚨 Database backup file is too small!"}' \
            https://hooks.slack.com/services/...
    fi
```

Шаг 8. Тестирование восстановления

```
    # 1. Создание тестовой базы данных
    createdb -U myproject_user test_restore_db

    # 2. Восстановление бэкапа в тестовую БД
    pg_restore -U myproject_user -d test_restore_db latest_backup.dump

    # 3. Проверка данных
    psql -U myproject_user -d test_restore_db -c "SELECT COUNT(*) FROM api_news;"

    # 4. Сравнение с оригиналом
    psql -U myproject_user -d myproject_db -c "SELECT COUNT(*) FROM api_news;"
```

```
# 5. Очистка тестовой БД
dropdb -U myproject_user test_restore_db

# 6. Регулярное автоматическое тестирование (в cron)
0 6 * * 0 /home/deploy/scripts/test_restore.sh >> /home/deploy/logs/test_restore.log 2>&1
```

4. Индивидуальные задания

4.1. Базовый уровень (15 вариантов)

Требования: Настроить автоматический бэкап PostgreSQL с ротацией.

Вариант	Частота бэкапа	Сжатие	Хранение
1	Ежедневно	gzip	Локально
2	Ежедневно	gzip	S3
3	Ежедневно	zstd	Локально + S3
4	Каждые 6 часов	gzip	Локально
5	Каждые 6 часов	gzip	S3
6	Раз в 2 дня	gzip	Локально + S3

Вариант	Частота бэкапа	Сжатие	Хранение
7	Еженедельно	gzip	Локально
8	Еженедельно	gzip	S3
9	Еженедельно	zstd	Локально + S3
10	Ежемесячно	gzip	Локально
11	Ежемесячно	gzip	S3
12	Раз в 3 дня	gzip	Локально + S3
13	Дважды в день	gzip	Локально
14	Дважды в день	gzip	S3
15	Дважды в день	zstd	Локально + S3

4.2. Средний уровень (15 вариантов)

Требования: Настроить бэкап медиа-файлов в S3 с синхронизацией и ротацией.

Вариант	Тип бэкапа	Хранилище	Ротация
1	Инкрементальный (rsync)	Локальный диск	30 дней
2	Инкрементальный (rsync)	S3	30 дней
3	Полный (tar)	Локальный диск	7 дней
4	Полный (tar)	S3	7 дней
5	Синхронизация (aws s3 sync)	S3	Бесконечно
6	Синхронизация (rclone)	Google Drive	90 дней
7	Синхронизация (rclone)	Yandex Disk	60 дней
8	Инкрементальный + Полный	Локальный + S3	30/90 дней
9	Синхронизация + Архив	S3 + Glacier	30/365 дней
10	Синхронизация (s3cmd)	S3	14 дней
11	Синхронизация (gcloud)	GCS	30 дней
12	Синхронизация (azcopy)	Azure Blob	30 дней
13	Инкрементальный (duplicity)	S3	60 дней
14	Полный + Инкрементальный	Локальный +	14/60 дней

Вариант	**Тип бэкапа**	**Хранилище**	**Ротация**
		S3	
15	Синхронизация с версионированием	S3	90 дней

4.3. Продвинутый уровень (15 вариантов)

Требования: Настроить полную стратегию бэкапа (база + медиа + конфиги) с мониторингом и автоматическим тестированием.

Вариант	Дополнительные компоненты	Цель RPO/RTO
1	WAL архивация + PITR	15 минут / 1 час
2	Репликация (master-slave)	5 минут / 30 минут
3	Бэкап + репликация	10 минут / 1 час
4	WAL + S3 Glacier	1 час / 6 часов
5	Multi-region репликация	1 минута / 5 минут
6	Kubernetes PV snapshot	5 минут / 30 минут
7	Docker volume + S3	30 минут / 2 часа
8	WAL + TimescaleDB	15 минут / 1 час
9	Patroni (автоматическая репликация)	1 секунда / 5 минут
10	Бэкап + мониторинг	1 час / 4 часа

Вариант	Дополнительные компоненты	Цель RPO/RTO
	(Prometheus)	
11	WAL + Terraform восстанавливаемость	1 час / 2 часа
12	GitOps + ArgoCD	5 минут / 15 минут
13	Бэкап + Ansible восстановление	30 минут / 1 час
14	WAL + сквозное шифрование	1 час / 2 часа
15	Бэкап + проверка целостности	30 минут / 4 часа

5. Контрольные вопросы

1. Что такое резервное копирование и зачем оно нужно?
2. Какие типы бэкапов вы знаете?
3. Как сделать бэкап PostgreSQL с помощью pg_dump?
4. Как восстановить базу данных из бэкапа?
5. Как настроить автоматический бэкап через cron?
6. Что такое PITR (Point-In-Time Recovery)?
7. Как настроить бэкап медиа-файлов в S3?
8. Как проверить целостность бэкапа с помощью контрольной суммы?
9. Как настроить ротацию старых бэкапов?

10. Как восстановить данные из бэкапа на другой сервер?
11. Как настроить WAL архивацию для PostgreSQL?
12. Что такое стратегия 3-2-1 и как её реализовать?
13. Как настроить репликацию базы данных?
14. Как автоматизировать тестирование восстановления?
15. Как настроить мониторинг бэкапов и уведомления?

6. Требования к отчёту

Титульный лист (название работы, ФИО, группа, вариант)

Цель работы

Теоретическое обоснование (своими словами)

Выполнение практического примера (скрипты, команды, скриншоты)

Выполнение индивидуального задания (конфигурации, скриншоты)

Ответы на контрольные вопросы (не менее 5 по своему уровню)

Вывод

7. Критерии оценивания

Критерий	Базовый (max 35)	Средний (max 50)	Продвинутый (max 65)
Бэкап PostgreSQL (pg_dump, cron)	20	15	10
Бэкап медиа-файлов в S3	—	15	10
Настройка ротации	10	10	10
Восстановление из	5	5	10

Критерий	Базовый (max 35)	Средний (max 50)	Продвинутый (max 65)
бэкапа			
WAL архивация / репликация	—	—	15
Мониторинг и тестирование	—	5	10
Итого	35	50	65

Шкала перевода баллов в оценку:

Оценка	Базовый (35)	Средний (50)	Продвинутый (65)
5 (отлично)	32-35	46-50	59-65
4 (хорошо)	28-31	40-45	51-58
3 (удовлетворительно)	22-27	32-39	42-50
2 (неудовлетворительно)	0-21	0-31	0-41

Лабораторная работа №36

Финальный проект: интеграция ИИ-модели в веб-приложение

1. Цель работы

Освоить полный цикл интеграции готовой модели искусственного интеллекта в веб-приложение на основе стека Django + React. Научиться создавать API-обертку для модели на Django REST Framework, интегрировать её с React-фронтендом, реализовывать асинхронную обработку через Celery для длительных задач, настраивать кэширование результатов и документировать созданное API.

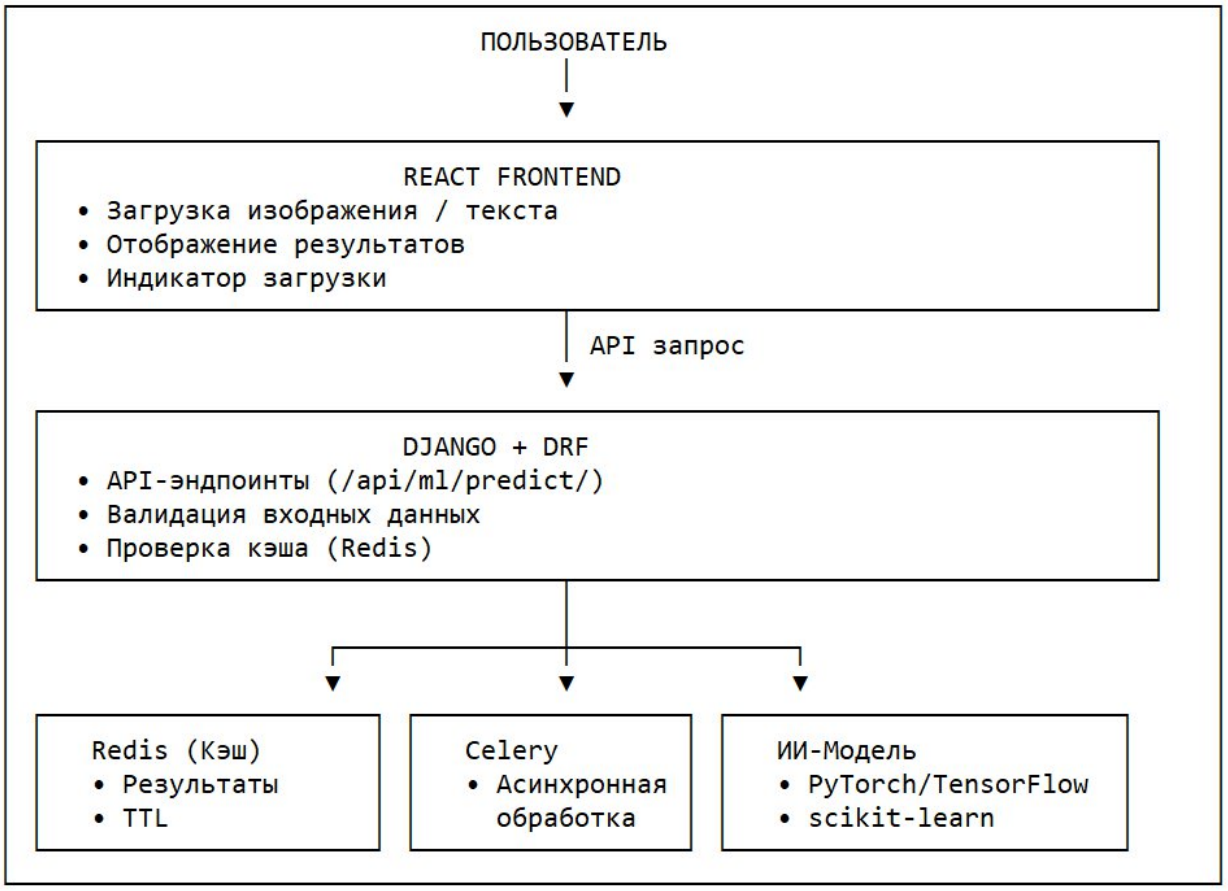
2. Теоретическое обоснование

2.1. Что такое интеграция ИИ в веб-приложение?

Интеграция ИИ в веб-приложение - это процесс внедрения моделей машинного обучения в существующую веб-инфраструктуру, позволяющий пользователям взаимодействовать с ИИ через удобный веб-интерфейс.

Аналогия: Представьте, что у вас есть супер-калькулятор (ИИ-модель), который умеет делать сложные вычисления. Интеграция - это создание кнопок, экрана и проводов, чтобы любой человек мог нажать кнопку и получить результат, не зная, как работает сам калькулятор.

Типичная архитектура:



2.2. Зачем нужна асинхронная обработка (Celery)?

Некоторые ИИ-задачи могут занимать много времени (несколько секунд или даже минут). Если обрабатывать их синхронно в HTTP-запросе, пользователь будет ждать, а сервер может зависнуть.

Сравнение подходов:

Характеристика	**Синхронная обработка**	**Асинхронная (Celery)**
Время ответа	Долгое (ждём модель)	Мгновенное (получаем task_id)
Риск таймаута	Высокий	Нет
Масштабируемо	Ограничена	Хорошая

Характеристика	Синхронная обработка	Асинхронная (Celery)
сть		
Сложность	Низкая	Средняя
Прогресс выполнения	Не виден	Можно отслеживать

2.3. Зачем нужно кэширование результатов?

Если один и тот же запрос к ИИ поступает многократно (например, классификация одного и того же изображения), можно сохранить результат в кэше и не тратить ресурсы на повторный запуск модели .

Стратегия кэширования	Время первого запроса	Время кэшированного запроса	Ускорение
Без кэша	100-1000 мс	100-1000 мс	1x
In-memory (локально)	100-1000 мс	< 0.1 мс	1000x+
Redis	100-1000 мс	2-5 мс	20-50x
DiskCache	100-1000 мс	10-20 мс	5-10x

3. Практический пример

Что мы будем делать: Создадим приложение для классификации изображений с использованием предобученной модели (например, ResNet50). Пользователь загружает изображение через React-интерфейс, Django API обрабатывает запрос (асинхронно через Celery), кэширует результат и возвращает предсказанные классы с вероятностями.

Шаг 1. Выбор и подготовка ИИ-модели

1.1. Установка зависимостей

```
# Установка ML-библиотек
pip install torch torchvision --index-url https://download.pytorch.org/whl/cpu
pip install scikit-learn numpy pandas pillow
pip install joblib

# Для асинхронной обработки
pip install celery redis
```

1.2. Создание ML-приложения в Django

```
python manage.py startapp ml_engine
```

1.3. Структура ML-приложения

```
ml_engine/
├── __init__.py
├── admin.py
├── apps.py
├── models.py          # Модели для хранения истории запросов
├── serializers.py     # Сериализаторы для API
├── views.py           # API-эндпоинты
├── urls.py
```

```
├── tasks.py           # Celery задачи
├── ml_services/       # Сервисный слой для ML
│   ├── __init__.py
│   ├── classifier.py  # Класс для работы с моделью
│   ├── cache.py       # Кэширование результатов
│   └── preprocess.py  # Предобработка изображений
└── tests.py
```

1.4. Класс для работы с моделью

Файл ml_engine/ml_services/classifier.py:

```
import torch
import torchvision.models as models
import torchvision.transforms as transforms
from PIL import Image
import joblib
import json
import os
from pathlib import Path

class ImageClassifier:
    """
    Класс для классификации изображений с использованием предобученной модели ResNet50.
    """

    def __init__(self):
        self.device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
        self.model = None
```

```
self classes = None
self transform = None
self _load_model()
self _load_classes()

def _load_model(self):
"""Загрузка предобученной модели ResNet50"""
# Загружаем модель без последнего слоя (классификатора)
self model = models resnet50(pretrained=True)
self model eval()  # Режим оценки (не обучения)
self model = self model to(self device)

# Трансформации для входного изображения
self transform = transforms Compose([
transforms Resize(256),
transforms CenterCrop(224),
transforms ToTensor(),
transforms Normalize(
mean=[0.485, 0.456, 0.406],
std=[0.229, 0.224, 0.225]
)
])

print(f"Модель ResNet50 загружена на устройство: {self device}")

def _load_classes(self):
"""Загрузка меток классов ImageNet (1000 классов)"""
# В реальном проекте можно загрузить из файла
# Здесь используем упрощённый вариант
```

```
classes_path = Path(__file__).parent / 'imagenet_classes.json'
if classes_path.exists():
with open(classes_path) as f:
self.classes = json.load(f)
else:
# Fallback: базовые классы для демонстрации
self.classes = [f"class_{i}" for i in range(1000)]

def preprocess_image(self, image_bytes):
"""Преобразование байтов изображения в тензор PyTorch"""
image = Image.open(image_bytes).convert('RGB')
image_tensor = self.transform(image).unsqueeze(0)
return image_tensor.to(self.device)

def predict(self, image_bytes, top_k=5):
"""

Классификация изображения.

Args:
image_bytes: Байты загруженного изображения
top_k: Количество лучших предсказаний

Returns:
list: Список словарей с классами и вероятностями
"""
# Предобработка
input_tensor = self.preprocess_image(image_bytes)
```

```
# Инференс модели
with torch no_grad():
outputs = self model(input_tensor)
probabilities = torch nn functional softmax(outputs[0], dim=0)

# Получение top-k предсказаний
top_probs, top_indices = torch topk(probabilities, top_k)

results = []
for i in range(top_k):
idx = top_indices[i] item()
prob = top_probs[i] item()
class_name = self classes[idx] if idx < len(self classes) else f"class_{idx}"
results append({
'class_id': idx,
'class_name': class_name,
'probability': round(prob * 100, 2)
})

return results


# Создаём глобальный экземпляр (один на всё приложение)
classifier = ImageClassifier()
```

1.5. Модуль кэширования

Файл ml_engine/ml_services/cache.py:

```
import hashlib
```

```
import json
from django.core.cache import cache

def get_cache_key(image_bytes, model_version='v1', top_k=5):
    """
    Генерация уникального ключа кэша на основе содержимого изображения.
    """
    # Вычисляем хеш изображения
    image_hash = hashlib.sha256(image_bytes).hexdigest()
    return f"ml_prediction:{model_version}:{image_hash}:{top_k}"

def get_cached_prediction(image_bytes, model_version='v1', top_k=5):
    """Получение предсказания из кэша"""
    cache_key = get_cache_key(image_bytes, model_version, top_k)
    return cache.get(cache_key)

def set_cached_prediction(image_bytes, prediction, model_version='v1', top_k=5, timeout=3600):
    """Сохранение предсказания в кэш"""
    cache_key = get_cache_key(image_bytes, model_version, top_k)
    cache.set(cache_key, prediction, timeout)
```

Шаг 2. Настройка Celery для асинхронной обработки

2.1. Создание Celery приложения

Файл myproject/celery.py:

```
import os
from celery import Celery
```

```
# Устанавливаем переменную окружения для настроек Django
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'myproject.settings')

app = Celery('myproject')

# Загрузка настроек из Django settings
app.config_from_object('django.conf:settings', namespace='CELERY')

# Автоматическое обнаружение задач в приложениях
app.autodiscover_tasks()

@app.task(bind=True)
def debug_task(self):
print(f'Request: {self.request!r}')
```

Файл myproject/__init__.py (для загрузки Celery):

```
from .celery import app as celery_app

__all__ = ('celery_app',)
```

2.2. Настройка Celery в settings.py

```
# Celery настройки
CELERY_BROKER_URL = os.getenv('REDIS_URL', 'redis://localhost:6379/0')
```

```
CELERY_RESULT_BACKEND = os.getenv('REDIS_URL', 'redis://localhost:6379/0')
CELERY_ACCEPT_CONTENT = ['json']
CELERY_TASK_SERIALIZER = 'json'
CELERY_RESULT_SERIALIZER = 'json'
CELERY_TIMEZONE = 'Europe/Moscow'
CELERY_TASK_TRACK_STARTED = True
CELERY_TASK_TIME_LIMIT = 30 * 60  # 30 минут
CELERY_TASK_SOFT_TIME_LIMIT = 25 * 60  # 25 минут

# Настройки кэша Redis
CACHES = {
'default': {
'BACKEND': 'django_redis.cache.RedisCache',
'LOCATION': os.getenv('REDIS_URL', 'redis://localhost:6379/1'),
'OPTIONS': {
'CLIENT_CLASS': 'django_redis.client.DefaultClient',
}
}
}
```

2.3. Создание Celery задач

Файл ml_engine/tasks.py:

```
from celery import shared_task
from .ml_services.classifier import classifier
from .ml_services.cache import get_cached_prediction, set_cached_prediction
from .models import PredictionHistory
```

```
import base64

@shared_task(bind=True, max_retries=3)
def classify_image_task(self, image_base64, user_id=None):
"""
Асинхронная задача для классификации изображения.

Args:
image_base64: Изображение в формате base64
user_id: ID пользователя (для логирования)

Returns:
dict: Результаты классификации
"""
try:
# Декодируем base64 в байты
image_bytes = base64.b64decode(image_base64)

# Проверяем кэш
cached_result = get_cached_prediction(image_bytes)
if cached_result:
return {'result': cached_result, 'cached': True}

# Выполняем предсказание
predictions = classifier.predict(image_bytes)

# Сохраняем в кэш (на 1 час)
set_cached_prediction(image_bytes, predictions, timeout=3600)
```

```
        # Сохраняем историю (если есть пользователь)
        if user_id:
            PredictionHistory.objects.create(
                user_id=user_id,
                image_hash=hashlib.md5(image_bytes).hexdigest(),
                predictions=predictions
            )

        return {'result': predictions, 'cached': False}

    except Exception as e:
        # Повторная попытка при ошибке
        self.retry(exc=e, countdown=60)
```

Шаг 3. Создание API на Django REST Framework

3.1. Модели

Файл ml_engine/models.py:

```
from django.db import models
from django.contrib.auth.models import User

class PredictionHistory(models.Model):
    """Модель для хранения истории предсказаний"""
    user = models.ForeignKey(User, on_delete=models.CASCADE, null=True, blank=True)
    image_hash = models.CharField(max_length=64, db_index=True)
    predictions = models.JSONField()
    created_at = models.DateTimeField(auto_now_add=True)
```

```
class Meta:
ordering = ['-created_at']
indexes = [
models Index(fields=['user', 'created_at']),
]

def __str__(self):
return f"Prediction by {self user} at {self created_at}"
```

3.2. Сериализаторы

Файл ml_engine/serializers.py:

```
from rest_framework import serializers
from  models import PredictionHistory

class PredictionRequestSerializer(serializers Serializer):
"""Сериализатор для запроса на предсказание"""
image = serializers ImageField()
top_k = serializers IntegerField(min_value=1, max_value=10, default=5)

class PredictionResponseSerializer(serializers Serializer):
"""Сериализатор для ответа с предсказанием"""
class PredictionItemSerializer(serializers Serializer):
class_id = serializers IntegerField()
class_name = serializers CharField()
probability = serializers FloatField()

result = PredictionItemSerializer(many=True)
cached = serializers BooleanField()
```

```
task_id = serializers.CharField(required=False)
status = serializers.CharField()

class TaskStatusSerializer(serializers.Serializer):
"""Сериализатор для статуса асинхронной задачи"""
task_id = serializers.CharField()
status = serializers.CharField()
result = PredictionResponseSerializer(required=False)
error = serializers.CharField(required=False)

class PredictionHistorySerializer(serializers.ModelSerializer):
class Meta:
model = PredictionHistory
fields = ['id', 'image_hash', 'predictions', 'created_at']
```

3.3. Представления (Views)

Файл ml_engine/views.py:

```
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework import status, permissions
from celery.result import AsyncResult
import base64
from .serializers import (
PredictionRequestSerializer, PredictionResponseSerializer,
TaskStatusSerializer, PredictionHistorySerializer
)
from .tasks import classify_image_task
from .models import PredictionHistory
```

```
class PredictSyncView(APIView):
"""
Синхронное предсказание (ждёт результат).
Подходит для быстрых моделей (< 5 секунд).
"""
permission_classes = [permissions IsAuthenticatedOrReadOnly]

def post(self, request):
serializer = PredictionRequestSerializer(data=request data)
serializer is_valid(raise_exception=True)

image = serializer validated_data['image']
top_k = serializer validated_data get('top_k', 5)

# Конвертируем изображение в base64
image_base64 = base64 b64encode(image read()) decode('utf-8')

# Запускаем синхронно (ждём результат)
result = classify_image_task(image_base64, request user id if
request user is_authenticated else None)

response_serializer = PredictionResponseSerializer({
'result': result['result'],
'cached': result get('cached', False),
'status': 'completed'
})

return Response(response_serializer data)
```

```
class PredictAsyncView(APIView):
"""
Асинхронное предсказание (возвращает task_id).
Подходит для тяжёлых моделей (> 5 секунд).
"""
permission_classes = [permissions IsAuthenticatedOrReadOnly]

def post(self, request):
serializer = PredictionRequestSerializer(data=request data)
serializer is_valid(raise_exception=True)

image = serializer validated_data['image']

# Конвертируем изображение в base64
image_base64 = base64 b64encode(image read()) decode('utf-8')

# Запускаем асинхронную задачу
task = classify_image_task delay(
image_base64,
request user id if request user is_authenticated else None
)

return Response({
'task_id': task id,
'status': 'pending',
'message': 'Задача поставлена в очередь. Используйте /api/ml/status/{task_id}/ для получения результата.'
}, status=status HTTP_202_ACCEPTED)
```

```
class TaskStatusView(APIView):
"""
Получение статуса асинхронной задачи.
"""
permission_classes = [permissions AllowAny]

def get(self, request, task_id):
task = AsyncResult(task_id)

if task failed():
return Response({
'task_id': task_id,
'status': 'failed',
'error': str(task info)
}, status=status HTTP_500_INTERNAL_SERVER_ERROR)

if task ready():
result = task result
return Response({
'task_id': task_id,
'status': 'completed',
'result': result get('result'),
'cached': result get('cached', False)
})

return Response({
'task_id': task_id,
'status': 'pending'
```

```
        })


class PredictionHistoryView(APIView):
    """
    История предсказаний пользователя.
    """
    permission_classes = [permissions.IsAuthenticated]

    def get(self, request):
        history = PredictionHistory.objects.filter(user=request.user)[:50]
        serializer = PredictionHistorySerializer(history, many=True)
        return Response(serializer.data)
```

3.4. Маршруты

Файл ml_engine/urls.py:

```
from django.urls import path
from . import views

urlpatterns = [
    # Синхронное предсказание (ждёт результат)
    path('predict/sync/', views.PredictSyncView.as_view(), name='predict-sync'),

    # Асинхронное предсказание (возвращает task_id)
    path('predict/async/', views.PredictAsyncView.as_view(), name='predict-async'),

    # Статус задачи
```

```
    path('status/<str:task_id>/', views.TaskStatusView.as_view(), name='task-status'),

    # История предсказаний
    path('history/', views.PredictionHistoryView.as_view(), name='prediction-history'),
]
```

Шаг 4. Интеграция с React-фронтендом

4.1. API-клиент для работы с ML

Файл frontend/src/api/mlApi.js:

```
import axios from 'axios';

const API_BASE_URL = 'http://localhost:8000/api/ml';

// Синхронное предсказание (простой случай)
export const predictSync = async (imageFile, topK = 5) => {
  const formData = new FormData();
  formData.append('image', imageFile);
  formData.append('top_k', topK);

  const response = await axios.post(`${API_BASE_URL}/predict/sync/`, formData, {
    headers: { 'Content-Type': 'multipart/form-data' }
  });
  return response.data;
};
```

```
// Асинхронное предсказание (для тяжёлых моделей)
export const predictAsync = async (imageFile) => {
const formData = new FormData();
formData.append('image', imageFile);

const response = await axios.post(`${API_BASE_URL}/predict/async/`, formData, {
headers: { 'Content-Type': 'multipart/form-data' }
});
return response.data; // { task_id, status }
};

// Получение статуса задачи
export const getTaskStatus = async (taskId) => {
const response = await axios.get(`${API_BASE_URL}/status/${taskId}/`);
return response.data;
};

// Получение истории предсказаний
export const getPredictionHistory = async () => {
const response = await axios.get(`${API_BASE_URL}/history/`);
return response.data;
};
```

4.2. React-компонент для загрузки и отображения

Файл frontend/src/components/ImageClassifier.jsx:

```
import { useState, useCallback } from 'react';
import { predictAsync, getTaskStatus } from '../api/mlApi';
```

```
const ImageClassifier = () => {
const [selectedImage, setSelectedImage] = useState(null);
const [previewUrl, setPreviewUrl] = useState(null);
const [predictions, setPredictions] = useState(null);
const [isLoading, setIsLoading] = useState(false);
const [taskId, setTaskId] = useState(null);
const [status, setStatus] = useState('idle'); // idle, uploading, processing, completed, error
const [error, setError] = useState(null);

// Обработка выбора изображения
const handleImageSelect = (event) => {
const file = event target files[0];
if (file) {
setSelectedImage(file);
setPreviewUrl(URL.createObjectURL(file));
setPredictions(null);
setError(null);
setStatus('idle');
}
};

// Загрузка и классификация
const handleClassify = async () => {
if (!selectedImage) return;

setIsLoading(true);
setStatus('uploading');
```

```
setError(null);

try {
// Запускаем асинхронное предсказание
const result = await predictAsync(selectedImage);
setTaskId(result task_id);
setStatus('processing');

// Поллинг статуса
pollTaskStatus(result task_id);
} catch (err) {
setError(err message);
setStatus('error');
setIsLoading(false);
}
};

// Поллинг статуса задачи
const pollTaskStatus = async (taskId) => {
const interval = setInterval(async () => {
try {
const statusData = await getTaskStatus(taskId);

if (statusData status === 'completed') {
setPredictions(statusData result);
setStatus('completed');
setIsLoading(false);
clearInterval(interval);
} else if (statusData status === 'failed') {
```

```
setError(statusData error || 'Ошибка обработки');
setStatus('error');
setIsLoading(false);
clearInterval(interval);
}
// status === 'pending' - продолжаем ждать
} catch (err) {
setError('Ошибка при проверке статуса');
setStatus('error');
setIsLoading(false);
clearInterval(interval);
}
}, 1000); // Проверяем каждую секунду

// Очистка интервала при размонтировании
return () => clearInterval(interval);
};

// Сброс
const handleReset = () => {
setSelectedImage(null);
setPreviewUrl(null);
setPredictions(null);
setTaskId(null);
setStatus('idle');
setError(null);
};

// Прогресс-бар статуса
```

```
const getStatusMessage = () => {
switch (status) {
case 'uploading': return '📤 Загрузка изображения...';
case 'processing': return '🧠 Анализ изображения ИИ...';
case 'completed': return '✅ Анализ завершён!';
case 'error': return '❌ Ошибка';
default: return '';
}
};

return (
<div style={{ maxWidth: '600px', margin: '0 auto', padding: '20px' }}>
<h2>Классификатор изображений</h2>
<p>Загрузите изображение, и ИИ определит, что на нём изображено.</p>

{/* Зона загрузки */}
<div style={{
border: '2px dashed #ccc',
borderRadius: '8px',
padding: '20px',
textAlign: 'center',
marginBottom: '20px'
}}>
{!previewUrl ? (
<>
<input
type="file"
```

```
accept="image/*"
onChange={handleImageSelect}
style={{ display: 'none' }}
id="image-input"
/>
<label htmlFor="image-input" style={{
display: 'inline-block',
padding: '10px 20px',
backgroundColor: '#007bff',
color: 'white',
borderRadius: '4px',
cursor: 'pointer'
}}>
Выбрать изображение
</label>
</>
) : (
<div>
<img src={previewUrl} alt="Preview" style={{
maxWidth: '100%',
maxHeight: '300px',
borderRadius: '8px',
marginBottom: '10px'
}} />
<div>
<button onClick={handleClassify} disabled={isLoading} style={{
padding: '10px 20px',
backgroundColor: '#28a745',
color: 'white',
```

```
border: 'none',
borderRadius: '4px',
marginRight: '10px',
cursor: 'pointer'
}}>
🔍 Классифицировать
</button>
<button onClick={handleReset} disabled={isLoading} style={{
padding: '10px 20px',
backgroundColor: '#6c757d',
color: 'white',
border: 'none',
borderRadius: '4px',
cursor: 'pointer'
}}>
Сбросить
</button>
</div>
</div>
)}
</div>

{/* Статус обработки */}
{isLoading && (
<div style={{
textAlign: 'center',
padding: '20px',
backgroundColor: '#e3f2fd',
borderRadius: '8px',
```

```
marginBottom: '20px'
}}>
<div>{getStatusMessage()}</div>
<progress style={{ width: '100%', marginTop: '10px' }} />
</div>
)}

{/* Ошибка */}
{error && (
<div style={{
padding: '15px',
backgroundColor: '#f8d7da',
color: '#721c24',
borderRadius: '8px',
marginBottom: '20px'
}}>
✕ {error}
</div>
)}

{/* Результаты классификации */}
{predictions && (
<div style={{
border: '1px solid #ddd',
borderRadius: '8px',
padding: '20px'
}}>
<h3>Результаты классификации:</h3>
<div>
```

```
{predictions.map((pred, idx) => (
<div key={idx} style={{
display: 'flex',
alignItems: 'center',
marginBottom: '15px',
padding: '10px',
backgroundColor: '#f8f9fa',
borderRadius: '4px'
}}>
<div style={{ flex: 2 }}>
<strong>{pred.class_name}</strong>
</div>
<div style={{ flex: 1 }}>
<div style={{
width: `${pred.probability}%`,
height: '20px',
backgroundColor: '#4caf50',
borderRadius: '4px',
textAlign: 'center',
color: 'white',
fontSize: '12px',
lineHeight: '20px'
}}>
{pred.probability}%
</div>
</div>
</div>
))}
</div>
```

```
{predictions cached && (
<div style={{ fontSize: '12px', color: '#666', marginTop: '10px' }}>
💾 Результат получен из кэша
</div>
)}
</div>
)}
</div>
);
};

export default ImageClassifier;
```

Шаг 5. Документирование API

5.1. Документирование через drf-spectacular

Файл ml_engine/views.py (с аннотациями):

```
from drf_spectacular.utils import extend_schema, OpenApiParameter, OpenApiExample
from drf_spectacular.types import OpenApiTypes

@extend_schema(
tags=['ML Predictions'],
summary='Асинхронная классификация изображения',
description="""
Загрузите изображение для классификации с помощью ИИ.
Возвращает task_id для отслеживания статуса.

## Как это работает
```

1. Вы загружаете изображение
2. Сервер ставит задачу в очередь Celery
3. Вы получаете task_id
4. Используйте /api/ml/status/{task_id}/ для получения результата

Модель
Используется предобученная ResNet50 (1000 классов ImageNet)
'''
request=PredictionRequestSerializer,
responses={
202: OpenApiResponse(
description='Задача поставлена в очередь',
response=TaskStatusSerializer,
examples=[
OpenApiExample(
'Пример ответа',
value={
'task_id': 'abc123-def456',
'status': 'pending',
'message': 'Задача поставлена в очередь'
}
)
]
),
400: OpenApiResponse(description='Ошибка валидации'),
401: OpenApiResponse(description='Требуется аутентификация'),
}
)
def post(self, request):

```
# ... код
```

5.2. Документирование модели (добавление в admin)

Файл ml_engine/admin.py:

```
from django contrib import admin
from  models import PredictionHistory

@admin.register(PredictionHistory)
class PredictionHistoryAdmin(admin ModelAdmin):
list_display = ['id', 'user', 'image_hash', 'created_at']
list_filter = ['created_at', 'user']
search_fields = ['user__username', 'image_hash']
readonly_fields = ['predictions']
date_hierarchy = 'created_at'
```

4. Запуск и тестирование

4.1. Запуск всех компонентов

```
# 1. Запуск Redis (для Celery и кэша)
docker run -d -p 6379:6379 redis

# 2. Запуск Celery worker (в отдельном терминале)
celery -A myproject worker -l info

# 3. Запуск Celery beat (для периодических задач, если нужны)
celery -A myproject beat -l info

# 4. Запуск Flower (мониторинг Celery)
```

```
celery -A myproject flower --port=5555

# 5. Запуск Django
python manage.py runserver

# 6. Запуск React
cd frontend && npm run dev
```

4.2. Тестирование API через curl

bash

```
# Загрузка изображения (асинхронно)
curl -X POST http://localhost:8000/api/ml/predict/async/ \
-F "image=@/path/to/image.jpg" \
-H "Authorization: Bearer <token>"

# Ответ:
# {"task_id": "abc123", "status": "pending"}

# Проверка статуса
curl http://localhost:8000/api/ml/status/abc123/

# При завершении:
# {"task_id": "abc123", "status": "completed", "result": [...]}
```

5. Индивидуальные задания

5.1. Базовый уровень (15 вариантов)

Требования: Реализовать интеграцию простой ИИ-модели (классификация, рекомендации) с синхронным API.

Вариант	Тип модели	Библиотека
1	Классификация изображений (ResNet50)	PyTorch
2	Классификация текстов (Spam detection)	scikit-learn
3	Анализ тональности текста	Transformers
4	Распознавание рукописных цифр (MNIST)	TensorFlow
5	Классификация цветов (Iris)	scikit-learn
6	Генерация эмбеддингов текста	SentenceTransformers
7	Рекомендация товаров (cosine similarity)	scikit-learn
8	Кластеризация изображений	scikit-learn
9	Детекция объектов (YOLO)	OpenCV
10	Распознавание лиц	face_recognition

Вариант	Тип модели	Библиотека
11	Генерация подписей к изображениям	Transformers
12	Перевод текста	Transformers
13	Распознавание эмоций по тексту	Transformers
14	Классификация новостей по категориям	scikit-learn
15	Predикция цен на недвижимость	scikit-learn

5.2. Средний уровень (15 вариантов)

Требования: Добавить асинхронную обработку через Celery и кэширование результатов.

Вариант	**Тип модели**	**Асинхронная обработка**
1	Генерация изображений (Stable Diffusion)	Да (долгая)
2	Транскрипция аудио (Whisper)	Да

Вариант	Тип модели	Асинхронная обработка
3	Синтез речи (TTS)	Да
4	Видеоаналитика (обнаружение сцен)	Да
5	Обработка больших PDF (извлечение текста)	Да
6	Пакетная классификация изображений	Да
7	Анализ временных рядов (прогноз)	Да
8	Генерация кода по описанию	Да
9	Семантический поиск по документам	Да
10	Вопросно-ответная система (QA)	Да
11	Реферирование текстов	Да
12	Парсинг резюме (CV parsing)	Да
13	Анализ медицинских	Да

Вариант	Тип модели	Асинхронная обработка
	снимков	
14	Обнаружение аномалий в данных	Да
15	Рекомендательная система для видео	Да

5.3. Продвинутый уровень (15 вариантов)

Требования: Полная интеграция с кэшированием, мониторингом, историей и документацией.

Вариант	Тип системы	Дополнительные возможности
1	Чат-бот на LLM	История диалогов, стриминг
2	Рекомендательная система для e-commerce	Персонализация, A/B тестирование
3	Система модерации контента	Автоматическая, очередь на проверку
4	Система распознавания лиц	Сравнение с базой, поиск
5	Генератор кода	HTML/CSS генерация

Вариант	Тип системы	Дополнительные возможности
	из изображений (UI)	
6	Анализ настроений в реальном времени	WebSocket + ML
7	Система предсказания оттока клиентов (churn)	MLflow отслеживание
8	Поисковая система на эмбеддингах	FAISS, векторный поиск
9	Система оценки кредитного риска	Интерпретируемость (SHAP/LIME)
10	Классификатор научных статей	Многометочная классификация
11	Система дополненной реальности с ML	TensorFlow.js в браузере
12	Оптимизация маршрутов на графах	ML + оптимизация
13	Система	Временные ряды + ML

Вариант	Тип системы	Дополнительные возможности
	прогнозирования спроса	
14	Детекция мошенничества в реальном времени	Stream processing + ML
15	Персонализированная лента рекомендаций	MLOps пайплайн

6. Контрольные вопросы

1. Какие этапы включает интеграция ИИ-модели в веб-приложение?
2. Как загрузить предобученную модель в Django?
3. Как создать API-эндпоинт для предсказаний на DRF?
4. Как обрабатывать загруженные изображения в Django?
5. Как подключить React-компонент к API классификации?
6. Для чего нужен Celery при интеграции ИИ?
7. Как настроить асинхронную обработку изображений?
8. Как отслеживать статус асинхронной задачи из React?
9. Как настроить кэширование результатов предсказаний?
10. Почему кэширование критично для production-систем?
11. Как реализовать поллинг (polling) или WebSocket для отслеживания прогресса?
12. Как версионировать модели ML и управлять несколькими версиями?

13. Как настроить мониторинг производительности модели в production?
14. Как реализовать A/B тестирование разных моделей?
15. Как автоматизировать переобучение модели на новых данных?

7. Требования к отчёту

Титульный лист (название работы, ФИО, группа, вариант)

Цель работы

Теоретическое обоснование (своими словами)

Выполнение практического примера (код, скриншоты)

Выполнение индивидуального задания (код, результаты, скриншоты)

Ответы на контрольные вопросы (не менее 5 по своему уровню)

Вывод

8. Критерии оценивания

Критерий	**Базовый (max 35)**	**Средний (max 50)**	**Продвинутый (max 65)**
Выбор и интеграция ИИ-модели	15	10	10
Создание API-обертки (DRF)	10	10	10
Интеграция с React	10	10	10
Асинхронная	—	15	10

Критерий	**Базовый (max 35)**	**Средний (max 50)**	**Продвинутый (max 65)**
обработка (Celery)			
Кэширование результатов	—	5	10
Документирование API	—	—	10
Качество кода	—	—	5
Итого	35	50	65

Шкала перевода баллов в оценку:

Оценка	Базовый (35)	Средний (50)	Продвинутый (65)
5 (отлично)	32-35	46-50	59-65
4 (хорошо)	28-31	40-45	51-58
3 (удовлетворительно)	22-27	32-39	42-50
2 (неудовлетворительно)	0-21	0-31	0-41

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по организации и проведению самостоятельной работы по дисциплине «Метод_Разработка Web-приложений» для студентов направления подготовки
09.03.04 Программная инженерия Направленность (профиль) «Разработка и сопровождение программного обеспечения»

Ставрополь 2026

ВВЕДЕНИЕ

Настоящие методические указания разработаны для студентов, проходящих практическую подготовку на платформе Яндекс.Практикум по направлению «backend-разработчик» в рамках освоения дисциплины «Разработка Web-приложения». Дисциплина предполагает сочетание теоретического обучения, выполнения лабораторных работ и прохождения спринтов на Яндекс.Практикуме, что позволяет сформировать у студентов комплексные практические навыки веб-разработки на Python/Django.

Цель методических указаний - обеспечить эффективную организацию самостоятельной работы студентов, синхронизировать прохождение спринтов Яндекс.Практикума с выполнением лабораторных работ, а также предоставить рекомендации по изучению теоретического материала и подготовке к контрольным мероприятиям.

1. ОБЩАЯ ХАРАКТЕРИСТИКА САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1.1. Структура курса

Курс «Веб-технологии» включает три взаимосвязанных компонента:

Компонент	**Формат**	**Объём**	**Форма контроля**
Спринты Яндекс.Практикума	Онлайн-курс с автоматической проверкой	12 спринтов	Автоматическое тестирование
Лабораторные работы	Практические задания в Django + React	36 лабораторных работ	Защита отчёта, демонстрация кода
Самостоятельная работа	Изучение теории, яндекс-	126 часов	Собеседование, опрос

Компонент	Формат	Объём	Форма контроля
	практикум подготовка к защитам		

1.2. Календарный график прохождения спринтов и лабораторных работ

Ниже представлен синхронизированный график, в котором спринты Яндекс.Практикума (Python-разработка) соотнесены с темами лабораторных работ.

Модуль 1: Основы Python и введение в веб-разработку (Спринты 1–4)

Неделя	Спринт Яндекс.Практикума	Тема спринта	Трудоёмкость	Лабораторная работа	Тема лабораторной работы
1	Спринт 1	**Основы Python. Настройка рабочего окружения**	**≈ 10 ч**	—	**Введение в веб-технологии**
2	Спринт 1 **(продолжение)**	**Основы Python (продолжение)**	**≈ 10 ч**	ЛР 1	**Создание виртуального окружения. Контроллеры и маршруты**
3	Спринт 2	**ООП и**	—	ЛР 2	**Модели.**

Неделя	Спринт Яндекс.Практикума	Тема спринта	Трудоёмкость	Лабораторная работа	Тема лабораторной работы
		другие полезные вещи, которые надо знать			**Миграции**
4	Спринт 2 **(продолжение)**	**ООП (продолжение)**	–	ЛР 3	**Основы CRUD. Консоль Django**
5	Спринт 3	**Основы веб-разработки**	–	ЛР 4	**Шаблоны. Админка Django**
6	Спринт 3 **(продолжение)**	**Основы веб-разработки (продолжение)**	–	ЛР 5	**Связи моделей. Внешний вид шаблона**
7	Спринт 4	**Базы данных и Django ORM**	–	ЛР 6	**Директивы, теги, фильтры**
8	Спринт 4 **(продолжение)**	**Django ORM (углублённ**	–	ЛР 7	**Параметры в URL. Имена маршрутов**

Неде ля	Спринт Яндекс.Практикума	Тема спринта	Трудоёмкость	Лабораторная работа	Тема лабораторной работы
		о)			
9	–	–	–	ЛР 8	**Наследование шаблонов**
10	–	–	–	ЛР 9	**Пользовательские теги шаблонов**

Модуль 2: Продвинутая бэкенд-разработка и тестирование (Спринты 5–7)

Неделя	Спринт Яндекс.Практикума	Тема спринта	Лабораторная работа	Тема лабораторной работы
11	Спринт 5	Формы в Django	ЛР 10	Обратное разрешение адресов. Статика
12	Спринт 5 (продолжение)	Формы в Django (продолжение)	ЛР 11	Работа с формами
13	Спринт 6	Тестирование на Python	ЛР 12	Классы-контроллеры

Неделя	Спринт Яндекс.Практикума	Тема спринта	Лабораторная работа	Тема лабораторной работы
14	Спринт 6 (продолжение)	Тестирование на Python (продолжение)	ЛР 13	Django ORM (продвинутый)
15	—	—	ЛР 14	Debug Toolbar. Оптимизация. Пагинация
16	Спринт 7	Работа с API	ЛР 15	Создание React-проекта и роутинг
17	Спринт 7 (продолжение)	Работа с API (продолжение)	ЛР 16	Управление состоянием с Context API

Модуль 3: Фронтенд-разработка на React (параллельно с API)

Неделя	Спринт Яндекс.Практикума	Тема спринта	Лабораторная работа	Тема лабораторной работы
18	—	—	ЛР 17	Redux Toolkit
19	—	—	ЛР 18	React Query / RTK Query

Неделя	Спринт Яндекс.Практикума	Тема спринта	Лабораторная работа	Тема лабораторной работы
20	—	—	ЛР 19	Аутентификация и защита роутов
21	—	—	ЛР 20	Формы с Formik и Yup
22	—	—	ЛР 21	UI-библиотеки (Material-UI)
23	—	—	ЛР 22	Тестирование React-компонентов
24	—	—	ЛР 23	Оптимизация производительности
25	—	—	ЛР 24	Деплой статического фронтенда

Модуль 4: API, Docker и финальный проект (Спринты 8–12)

Неделя	Спринт Яндекс.Практикума	Тема спринта	Лабораторная работа	Тема лабораторной работы
26	Спринт 8	Django	ЛР 25	Интеграция

Неделя	Спринт Яндекс.Практикума	Тема спринта	Лабораторная работа	Тема лабораторной работы
		REST Framework		фронтенда и бэкенда. CORS
27	Спринт 8 (продолжение)	Django REST Framework (продолжение)	ЛР 26	Тестирование бэкенда с pytest
28	Спринт 9	Настройка API-сервиса	ЛР 27	E2E-тестирование с Cypress
29	Спринт 9 (продолжение)	Настройка API-сервиса (продолжение)	ЛР 28	Контейнеризация с Docker
30	Спринт 10	Docker для Python-разработчика	ЛР 29	Оркестрация с Docker Compose
31	Спринт 10 (продолжение)	Docker для Python-разработчика	ЛР 30	Nginx и Gunicorn для продакшна

Неделя	Спринт Яндекс.Практикума	Тема спринта	Лабораторная работа	Тема лабораторной работы
		(продолжение)		
32	Спринт 11	Использование ИИ в разработке	ЛР 31	Развертывание на VPS
33	Спринт 11 (продолжение)	Использование ИИ в разработке (продолжение)	ЛР 32	CI/CD с GitHub Actions
34	Спринт 12	Шпаргалки со всего курса. Итоговое тестирование	ЛР 33	Автоматизация с Ansible
35	Спринт 12 (продолжение)	Проверка проектных работ студентов. Кросс-ревью	ЛР 34	Централизованное логирование и мониторинг
36	—	—	ЛР 35	Резервное

Неделя	Спринт Яндекс.Практикума	Тема спринта	Лабораторная работа	Тема лабораторной работы
				копирование
37	—	Финальный проект	ЛР 36	Интеграция ИИ-модели в веб-приложение
38	—	Кросс-ревью проектных работ	—	Защита курсового проекта

2. МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ИЗУЧЕНИЮ ТЕОРЕТИЧЕСКОГО МАТЕРИАЛА

2.1. Соотношение тем спринтов и лабораторных работ

Спринт Яндекс.Практикума	Соответствующие лабораторные работы	Ключевые темы для углублённого изучения
Спринт 1. **Основы Python**	**ЛР 1–3**	**Виртуальное окружение, pip, базовый синтаксис, функции**
Спринт 2. **ООП и другие полезные вещи**	**ЛР 2–5**	**Классы, наследование,**

Спринт Яндекс.Практикума	Соответствующие лабораторные работы	Ключевые темы для углублённого изучения
		магические методы, исключения
Спринт 3. **Основы веб-разработки**	**ЛР 4–8**	**HTTP, маршрутизация, шаблонизация, статика**
Спринт 4. **Базы данных и Django ORM**	**ЛР 2–3, ЛР 13**	**Модели, миграции, QuerySet, агрегация, связи**
Спринт 5. **Формы в Django**	**ЛР 11**	**ModelForm, валидация, CSRF, обработка форм**
Спринт 6. **Тестирование на Python**	**ЛР 26**	**pytest, unittest, фикстуры, моки, покрытие**
Спринт 7. **Работа с API**	**ЛР 18, ЛР 25**	**Fetch, Axios, REST, эндпоинты, CORS**
Спринт 8. **Django REST Framework**	**ЛР 5–9**	**Сериализаторы, ViewSet, Router, permissions**
Спринт 9. **Настройка API-сервиса**	**ЛР 25–27**	**Аутентификация, throttling,**

Спринт Яндекс.Практикума	Соответствующие лабораторные работы	Ключевые темы для углублённого изучения
		документация
Спринт 10. **Docker для Python-разработчика**	**ЛР 28–30**	**Dockerfile, docker-compose, контейнеризация**
Спринт 11. **Использование ИИ в разработке**	**ЛР 36**	**Интеграция моделей, API OpenAI, Celery**
Спринт 12. **Шпаргалки и итоговое тестирование**	**ЛР 31–35**	**CI/CD, мониторинг, бэкапы, Ansible**

2.2. Рекомендации по работе со спринтами

Проходите спринты последовательно - каждый следующий спринт опирается на предыдущий.

Выполняйте все практические задания спринтов - они формируют базу для лабораторных работ.

Сохраняйте код проектов - многие лабораторные работы могут быть основаны на коде из спринтов.

Анализируйте обратную связь от автотестов - это поможет выявить слабые места и улучшить качество кода.

Используйте «Шпаргалки со всего курса» (Спринт 12) - при подготовке к защитам лабораторных работ.

2.3. Рекомендуемая литература

Основная:

Официальная документация Django: https://docs.djangoproject.com/

Официальная документация React: https://react.dev/

Материалы Яндекс.Практикума (встроенные в спринты)

Дополнительная (по спринтам):

Спринт	Дополнительная литература
1–2	«Укус Питона» (A Byte of Python) - Swaroop C H
3–5	«Django 4 by Example» - Antonio Melé
6	«Python Testing with pytest» - Brian Okken
7–9	«Django for APIs» - William S. Vincent
10	«Docker in Action» - Jeff Nickoloff
11	«Hands-On Machine Learning with Scikit-Learn...» - Aurélien Géron

3. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

3.1. Порядок выполнения лабораторной работы

Изучите соответствующий спринт Яндекс.Практикума - он даст базовое понимание темы.

Прочитайте теоретическое обоснование в лабораторной работе.

Выполните практический пример - пошагово, разбирая каждую строчку кода.

Выберите уровень сложности (базовый / средний / продвинутый) в соответствии с уровнем подготовки.

Выполните индивидуальное задание.

Оформите отчёт по требованиям.

Подготовьтесь к защите по контрольным вопросам.

3.2. Приоритетность выполнения

Приоритет	Вид работы	Рекомендуемое время выполнения
1	Спринт Яндекс.Практикума	Начало недели (пн–ср)
2	Теоретическое обоснование ЛР	Середина недели (ср–чт)
3	Практический пример ЛР	Середина недели (чт–пт)
4	Индивидуальное задание	Конец недели (пт–сб)
5	Оформление отчёта	Перед сдачей (сб–вс)

4. ВОПРОСЫ ДЛЯ СОБЕСЕДОВАНИЯ ПО САМОСТОЯТЕЛЬНОМУ ИЗУЧЕНИЮ

4.1. Вопросы по спринтам 1–4 (Python, ООП, базы данных)

В чём разница между list, tuple и set в Python?

Что такое декораторы в Python? Приведите пример использования.

Объясните принципы ООП: инкапсуляция, наследование, полиморфизм.

Что такое ORM? Какие проблемы решает Django ORM?

Объясните разницу между ForeignKey, OneToOneField и ManyToManyField.

Что такое миграции в Django? Как создать и применить миграцию?

Как выполнить сложный запрос с использованием annotate и aggregate?

Что такое N+1 проблема в ORM? Как её решить с помощью select_related и prefetch_related?

4.2. Вопросы по спринтам 5–7 (Формы, тестирование, API)

В чём разница между Form и ModelForm в Django?

Как выполняется валидация данных в Django-формах?

Какие виды тестов существуют в Python? (unit, integration, e2e)

Что такое фикстуры в pytest? Как их использовать?

Что такое REST API? Перечислите основные принципы REST.

Какие HTTP-методы используются в REST API? Для чего каждый?

4.3. Вопросы по спринтам 8–9 (DRF, настройка API)

Что такое сериализаторы в DRF? В чём разница между Serializer и ModelSerializer?

Что такое ViewSet и Router в DRF? Какие преимущества они дают?

Какие классы прав доступа (permissions) встроены в DRF?

Что такое пагинация? Какие виды пагинации поддерживает DRF?

Как документировать API с помощью drf-spectacular?

Что такое аутентификация JWT? Как настроить JWT в DRF?

4.4. Вопросы по спринтам 10–12 (Docker, ИИ, итоговое тестирование)

Что такое Docker? В чём отличие контейнера от виртуальной машины?

Какие основные инструкции Dockerfile вы знаете?

Для чего нужен Docker Compose?

Как интегрировать ИИ-модель в Django-приложение?

Какие задачи можно решать с помощью Celery при интеграции ИИ?

Для чего нужно кэширование результатов ИИ-модели?

5. КРИТЕРИИ ОЦЕНИВАНИЯ КОМПЕТЕНЦИЙ

5.1. Весовые коэффициенты

Компонент оценки	Вес	Формируемые компетенции
Прохождение спринтов Яндекс.Практикума	30%	ПК-3 (разработка ПО)
Выполнение лабораторных работ	35%	ПК-2 (архитектура), ПК-3 (разработка)
Защита лабораторных работ (собеседование)	15%	Все компетенции
Итоговый проект (ЛР 36 + кросс-ревью)	20%	ПК-2, ПК-3, ПК-8

5.2. Оценка прохождения спринтов

Критерий	Максимальный балл за спринт
Своевременное прохождение	5
Процент выполненных заданий	до 10
Качество кода (ревью)	до 5
Участие в кросс-ревью	до 5
Итого за спринт	до 25

Всего 12 спринтов, максимальная сумма баллов за спринты — 300.

5.3. Оценка лабораторных работ (с учётом уровня сложности)

Уровень	Количество ЛР	Оценка за одну ЛР	Итоговый максимум
Базовый	36	до 35 баллов	до 1260
Средний	36	до 50 баллов	до 1800
Продвинутый	36	до 65 баллов	до 2340

Итоговая оценка нормализуется к 100-балльной шкале.

5.4. Карта компетенций

Компетенция	Формируется	Индикаторы
ПК-2 (архитектура программных систем)	ЛР 25–35, итоговый проект	ПК-2.1 – ПК-2.6
ПК-3 (разработка ПО)	ЛР 1–24, спринты 1–10	ПК-3.1 – ПК-3.7
ПК-8 (интеграция ИИ)	ЛР 36, спринт 11	ПК-8.1 – ПК-8.5

5.5. Шкала итоговой оценки

Процент выполненных заданий	Итоговая оценка
86–100%	5 (отлично)
71–85%	4 (хорошо)

Процент выполненных заданий	Итоговая оценка
56–70%	3 (удовлетворительно)
0–55%	2 (неудовлетворительно)

ЗАКЛЮЧЕНИЕ

Успешное освоение дисциплины «Разработка Web-приложений» требует системного подхода к самостоятельной работе, включающего:

Регулярное прохождение спринтов Яндекс.Практикума в соответствии с графиком. Особое внимание следует уделить спринтам по Django REST Framework (спринты 8-9) и Docker (спринт 10), так как они непосредственно связаны с ключевыми лабораторными работами.

Своевременное выполнение лабораторных работ, синхронизированное с темами спринтов. Бэкенд-лабораторные работы (ЛР 1-14) рекомендуется выполнять параллельно со спринтами 1-7, а лабораторные по интеграции и DevOps - после спринтов 8-12.

Ведение конспектов ключевых понятий и подготовку к собеседованиям с использованием контрольных вопросов.

Активное участие в кросс-ревью проектных работ на финальном этапе (спринт 12).

Данные методические указания помогут студентам эффективно распределить время, синхронизировать прохождение Яндекс.Практикума по направлению «Python-разработчик» с выполнением лабораторных работ и успешно сформировать необходимые профессиональные компетенции.