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 «Междисциплинарный проектный практикум-2» для студентов направления подготовки 09.03.04 Программная инженерия
Направленность (профиль)
«Разработка и сопровождение программного обеспечения»

Ставрополь 2026

Оглавление

Введение 2
Лабораторная работа №3 56
Лабораторная работа №4 75
Лабораторная работа №5 94
Лабораторная работа №6 128
Лабораторная работа №7 157
Лабораторная работа №8 186
Лабораторная работа №9 223
Лабораторная работа №10 238
Лабораторная работа №11 270
Лабораторная работа №12 296
Лабораторная работа №13 322
Лабораторная работа №14 353
Лабораторная работа №15 384
Лабораторная работа №16 414
Лабораторная работа №17 434
Лабораторная работа №18 451
Лабораторная работа №19 470
Лабораторная работа №20 494
Лабораторная работа №21 513
Лабораторная работа №22 529
Лабораторная работа №23 550
Лабораторная работа №24 568
Лабораторная работа №25 582
Лабораторная работа №26 616
Лабораторная работа №27 631

Введение

Дисциплина «Междисциплинарный проектный практикум-2» является ключевым элементом профессиональной подготовки студентов направления «Программная инженерия». Она интегрирует знания и умения, полученные при изучении таких дисциплин, как «Базы данных», «Технологии разработки программного обеспечения», «UI/UX дизайн», «Разработка Web-

приложений», и направлена на формирование практических навыков сквозной разработки веб-приложений в условиях командной проектной деятельности.

В рамках курса студенты проходят полный цикл создания современного веб-приложения - от анализа требований и проектирования архитектуры до production-развертывания, мониторинга и финальной защиты проекта.

2. Сквозной проект

В основе курса лежит **сквозной проект «Умный ежедневник» (Smart Diary)** - интеллектуальная платформа для управления задачами, проектами и личной продуктивностью.

Основные функциональные блоки проекта:

Блок	Описание
Управление задачами	CRUD-операции, приоритеты, дедлайны, статусы, подзадачи
Управление проектами	Группировка задач, цветовая индикация
Система тегов	Гибкая категоризация задач
Аутентификация	Регистрация, вход, JWT-токены, ролевая модель
Статистика и аналитика	Метрики продуктивности, экспорт отчетов (CSV, JSON, PDF)
Визуализация	Календарь, Kanban-доска с drag-and-drop
Реальное время	WebSocket-уведомления о дедлайнах
Совместная работа	Шаринг задач и проектов, роли, комментарии
Внешние интеграции	Погода (OpenWeatherMap), Google Calendar
PWA	Установка на устройство, офлайн-режим, push-уведомления
AI-ассистент	Генерация описаний, рекомендация приоритетов, планирование дня
Администрирование	Управление пользователями, системные логи, дашборды
DevOps	Docker, CI/CD, мониторинг (Prometheus +

Блок	Описание
	Grafana), логирование (Loki)

3. Технологические траектории

Курс поддерживает **две параллельные технологические траектории**, которые реализуют один и тот же сквозной проект на разных технологических стеках. Студент выбирает траекторию в начале 3 семестра и следует ей на протяжении всего курса.

Траектория А: Django + React

Компонент	Технологии
Backend	Python 3.11, Django 5, Django REST Framework
ORM	Django ORM
Аутентификация	JWT (djangorestframework-simplejwt)
База данных	PostgreSQL
Кэш / очереди	Redis, Celery
Real-time	Django Channels
Frontend	React 18, TypeScript, Vite
Управление состоянием	Redux Toolkit
UI-библиотека	Material-UI (MUI)
Тестирование	pytest, pytest-django, Jest, React Testing Library, Cypress

Траектория Б: ASP.NET Core + React

Компонент	Технологии
Backend	C#, ASP.NET Core 8, Entity Framework Core
ORM	Entity Framework Core
Аутентификация	ASP.NET Core Identity + JWT

Компонент	Технологии
База данных	PostgreSQL / SQL Server
Кэш / очереди	Redis, Hangfire
Real-time	SignalR
Frontend	React 18, TypeScript, Vite
Управление состоянием	Redux Toolkit
UI-библиотека	Material-UI (MUI)
Тестирование	xUnit, Moq, Jest, React Testing Library, Cypress

Важно: C# и ASP.NET Core изучаются студентами параллельно в 4 семестре в дисциплине «Технологии разработки программного обеспечения», что позволяет плавно освоить вторую траекторию.

4. Структура курса

Курс состоит из **27 лабораторных работ**, распределённых по двум семестрам.

Семестр	Количество работ	Основной фокус	Итоговый контроль
3 семестр	9	Проектирование, Backend, Базы данных, Аутентификация, Базовый фронтенд	Дифференцированный зачет
4 семестр	18	Продвинутый фронтенд, Реальное время, DevOps, AI, Тестирование, Деплой	Экзамен (защита проекта)

5. Формируемые компетенции

Выполнение лабораторных работ направлено на формирование следующих **профессиональных компетенций (ПК)**:

Компетенция	**Описание**
ПК-1	Управление программными проектами (Git, задачи, сроки)

Компетенция	Описание
ПК-2	Проектирование архитектуры программных систем (MVC, паттерны)
ПК-3	Разработка ПО (код на современных языках, веб-приложения, БД)
ПК-4	Тестирование и отладка (unit, integration, e2e, нагрузочное)
ПК-5	Развертывание и эксплуатация (Docker, CI/CD, облако)
ПК-6	Работа с данными (SQL, ORM, миграции, интеграции)
ПК-7	Современные ИТ и DevOps (инструменты, облака, автоматизация)
ПК-8	Применение ИИ/МО (интеграция готовых моделей)
ПК-9	Документирование и коммуникация (техническая документация, презентации)
ПК-10	Информационная безопасность (аутентификация, защита от уязвимостей)

Каждая лабораторная работа в своем описании содержит явное указание на формируемые компетенции.

6. Структура лабораторной работы

Каждая лабораторная работа имеет единую структуру:

Раздел	Содержание
Цель работы	Краткое описание того, что студент должен освоить
Задачи	Конкретные технические задачи с указанием компетенций
Теоретическая часть	Базовые понятия, необходимые для выполнения работы
Ход выполнения работы	Пошаговые инструкции с кодом и пояснениями

Раздел	Содержание
Контрольные вопросы	Вопросы для самопроверки и подготовки к защите
Критерии оценки	Распределение баллов по критериям
Дополнительные задания	Для углублённого изучения (по желанию)
Результат выполнения	Что должно быть получено после выполнения

7. Правила оформления и сдачи

7.1. Репозиторий

Все лабораторные работы должны быть сохранены в **Git-репозитории** (GitHub / GitLab). Рекомендуемая структура:

```
text
smart-diary/
├── backend/              # Бэкенд приложение
├── frontend/             # Фронтенд приложение
├── docs/                 # Документация
├── monitoring/           # Конфигурация мониторинга
├── terraform/            # IaC конфигурации
├── docker-compose.yml
├── .github/workflows/    # CI/CD пайплайны
└── README.md
```

7.2. Отчет по лабораторной работе

Каждая лабораторная работа должна сопровождаться **отчётом** в формате Markdown или PDF, содержащим:

- Титульный лист (название работы, ФИО, группа, дата)
- Цель работы
- Ход выполнения (с кодом и скриншотами ключевых этапов)
- Ответы на контрольные вопросы
- Выводы

7.3. Критерии оценки

Каждая лабораторная работа оценивается по **10-балльной шкале**. Критерии оценки указаны в конце каждой работы. Для получения зачёта/экзамена необходимо набрать не менее **60%** от максимального балла за семестр.

7.4. Защита лабораторной работы

Лабораторная работа считается сданной только после **защиты**, которая включает:

- Демонстрацию работающего кода
- Объяснение принятых решений
- Ответы на контрольные вопросы

8. Ресурсы и инструменты

8.1. Обязательные инструменты

Инструмент	Назначение
Git	Контроль версий
Python 3.11+ / .NET 8	Backend
Node.js 18+	Frontend
Docker Desktop	Контейнеризация
PostgreSQL 15+	База данных
VS Code / Visual Studio / PyCharm	IDE

8.2. Дополнительные ресурсы

1. Django Documentation: https://docs.djangoproject.com/
2. Django REST Framework Documentation: https://www.django-rest-framework.org/
3. ASP.NET Core Documentation: https://learn.microsoft.com/en-us/aspnet/core/
4. React Documentation: https://react.dev/
5. Redux Toolkit Documentation: https://redux-toolkit.js.org/
6. Docker Documentation: https://docs.docker.com/
7. GitHub Actions Documentation: https://docs.github.com/en/actions

9. Рекомендации по выполнению

1. Не пропускайте лабораторные работы. Каждая последующая работа опирается на результаты предыдущей.
2. Ведите регулярные коммиты. Каждая выполненная работа должна быть зафиксирована в Git.
3. Пишите тесты. Тестирование — неотъемлемая часть профессиональной разработки.
4. Документируйте код. Используйте комментарии, docstring, README.
5. Не бойтесь задавать вопросы. Преподаватель и ассистенты помогут разобраться со сложными моментами.
6. Используйте CI/CD. Автоматизация тестирования и деплоя экономит время и повышает качество.
7. Следите за безопасностью. SQL-инъекции, XSS, CSRF — недопустимы в итоговом проекте.

Успешное выполнение всех лабораторных работ позволит вам:

- Создать **полноценное production-готовое веб-приложение**
- Получить **практический опыт** работы с современным технологическим стеком
- Сформировать **портфолио** для будущего трудоустройства

- Развить **навыки командной работы** и проектной деятельности

Лабораторная работа №1. Анализ требований и Проектирование архитектуры

Цель работы: Получить навыки анализа требований к программному продукту, проектирования структуры базы данных и организации архитектуры веб-приложения на основе паттерна MVC в выбранной технологии.

Задачи

1. Провести анализ пользовательских ролей и сценариев использования.
2. Спроектировать модели данных и связи между ними.
3. Построить ER-диаграмму базы данных.
4. Описать реализацию паттерна MVC в проекте.
5. Создать структуру проекта в выбранной технологии.

Теоретическая часть (общая для всех)

Паттерн MVC (Model-View-Controller)

- Model (Модель) — представляет данные и бизнес-логику. Отвечает за получение, обработку и валидацию данных, взаимодействие с БД.

- View (Представление) - отвечает за отображение данных пользователю. Получает данные от контроллера и формирует интерфейс.

- Controller (Контроллер) - обрабатывает пользовательский ввод, взаимодействует с моделью, выбирает представление для ответа.

Роли пользователей в системе "Умный ежедневник"

Роль	Описание	Возможности
Анонимный пользователь	Неавторизованный посетитель	Просмотр публичной информации (если есть), регистрация, вход в систему
Авторизованный пользователь	Зарегистрированный пользователь	Создание/редактирование/удаление своих задач, проектов, тегов; просмотр своих задач; установка

		приоритетов и дедлайнов; отметка о выполнении
Администратор	Привилегированный пользователь	Управление пользователями (блокировка), просмотр статистики, управление всеми задачами (при необходимости)

Сценарии использования (Use Cases)

Анонимный пользователь:

1. Зарегистрироваться в системе

2. Войти в систему (если уже зарегистрирован)

Авторизованный пользователь:

1. Создать новую задачу (указать заголовок, описание, дедлайн, приоритет, проект, теги)

2. Просмотреть список своих задач (с фильтрацией по статусу, проекту, дате)

3. Редактировать существующую задачу

4. Удалить задачу

5. Отметить задачу как выполненную

6. Создать новый проект (указать название, описание, цвет)

7. Создать новый тег

8. Просмотреть статистику (сколько задач выполнено, сколько просрочено)

Администратор:

1. Просмотреть список всех пользователей

2. Заблокировать/разблокировать пользователя

3. Просмотреть задачи любого пользователя

4. Просмотреть общую статистику системы

Ход выполнения работы

Шаг 1. Проектирование моделей данных

1.1. Выделение сущностей и их атрибутов

На основе анализа предметной области выделяем следующие сущности:

Пользователь (User)

- ID (уникальный идентификатор)
- Имя пользователя (UserName)
- Email (уникальный)
- Пароль (хэш)
- Аватар (путь к файлу)
- Дата регистрации
- Настройки (JSON-поле или отдельная таблица)

Проект (Project)

- ID
- Название (Name) - обязательное
- Описание (Description)
- Цвет (Color) - для визуального оформления
- Дата создания
- ID пользователя-владельца (OwnerId)

Тег (Tag)

- ID
- Название (Name) - уникальное в рамках пользователя
- ID пользователя-владельца (OwnerId)

Задача (Task)

- ID
- Заголовок (Title) - обязательное

- Описание (Description)

- Дата создания (CreatedAt)

- Дедлайн (Deadline) - может быть null

- Статус (Status): "Новая", "В работе", "Выполнена"

- Приоритет (Priority): "Низкий", "Средний", "Высокий"

- ID проекта (ProjectId) - внешний ключ

- ID пользователя-владельца (UserId) - внешний ключ

Связь задача-тег (TaskTag)

- ID задачи (TaskId)

- ID тега (TagId)

1.2. Описание связей между сущностями

- User → Project: один ко многим (один пользователь может иметь много проектов)

- User → Task: один ко многим (один пользователь может иметь много задач)

- User → Tag: один ко многим (один пользователь может иметь много тегов)

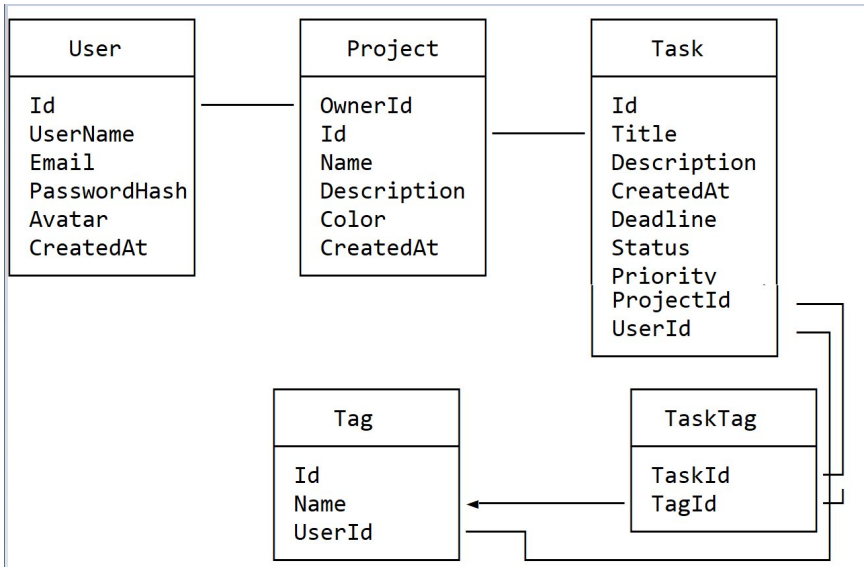
- Project → Task: один ко многим (один проект может содержать много задач)

- Task → Tag: многие ко многим (одна задача может иметь много тегов, один тег может быть у многих задач)

Шаг 2. Построение ER-диаграммы

Используя любой инструмент (draw.io, LucidChart, PlantUML, или даже от руки), необходимо построить ER-диаграмму.

Пример описания диаграммы в текстовом виде:



Требования к диаграмме:

- Указать все сущности
- Указать все атрибуты с типами данных
- Обозначить первичные ключи (PK)
- Обозначить внешние ключи (FK)
- Показать типы связей (1:∞, ∞:∞)

Шаг 3. Описание реализации MVC в проекте

Необходимо описать, как паттерн MVC будет реализован технически:

Компонент	**Роль в проекте**	**Примеры классов/файлов**
Model	Классы, представляющие данные и бизнес-логику	User.cs, Task.cs, Project.cs, Tag.cs; логика валидации, методы вычисления (например, IsOverdue)
View	Пользовательский интерфейс	Страница списка задач, форма создания задачи, канбан-доска (HTML + CSS + JS)
Controller	Обработка запросов, работа с моделью,	TaskController (методы Index, Create, Edit, Delete),

	выбор представления	ProjectController

Особенности для каждой траектории:

Для ASP.NET:

- Models - папка с классами C (POCO-классы)
- Views - папка с Razor-шаблонами (.cshtml)
- Controllers - папка с классами-контроллерами, наследующими Controller
- Data - папка с контекстом БД (ApplicationDbContext) и миграциями

Для Django + React:

- Models - файл models.py в приложении tasks (классы Python, наследующие models.Model)
- Views (Django) - файл views.py (функции или классы, обрабатывающие запросы и возвращающие JSON для React)
- Templates - папка для хранения React-приложения (SPA будет загружаться через один шаблон)
- API - сериализаторы (DRF) для преобразования моделей в JSON

Практическая часть

Для траектории ASP.NET

Шаг 1. Создание структуры решения

1. Открыть Visual Studio.
2. Создать новое решение (пустое или с проектом):
   - File → New → Project
   - Выбрать шаблон ASP.NET Core Web App (Model-View-Controller)
   - Назвать проект SmartDiary.Web

- Назвать решение SmartDiary
- Выбрать .NET 6/7/8

3. Изучить созданную структуру:

```
SmartDiary/
├── SmartDiary.Web/
│   ├── Controllers/          (пусто, но будет
HomeController)
│   ├── Views/                (папки с представлениями)
│   ├── Models/                (пусто)
│   ├── Data/                  (пусто, но будет DbContext)
│   ├── wwwroot/               (статический контент)
│   ├── appsettings.json
│   └── Program.cs
└── SmartDiary.sln
```

Шаг 2. Создание папок для моделей и данных

1. В проекте SmartDiary.Web создать папку Data (если её нет).
2. Создать папку Models (если её нет).

Шаг 3. Создание классов моделей (пока без реализации)

1. В папке Models создать пустые классы для будущего использования:
   - User.cs (пока пустой)
   - Task.cs
   - Project.cs
   - Tag.cs

2. В папке Data создать пустой класс контекста:
   - ApplicationDbContext.cs

Шаг 4. Настройка системы контроля версий

1. Инициализировать Git-репозиторий (если не сделано автоматически).
2. Создать файл .gitignore (можно взять стандартный для Visual Studio).
3. Сделать первый коммит:
   bash
   git add .
   git commit -m "Initial project structure for Smart Diary"

Шаг 5. Оформление отчета

В отчете по лабораторной работе должны быть:
- Титульный лист
- Цель работы
- Результаты аналитики (роли и сценарии)
- ER-диаграмма (скриншот или фото)
- Описание архитектуры MVC для проекта
- Скриншот структуры решения в Visual Studio
- Выводы

Для траектории Django + React

Шаг 1. Создание виртуального окружения и установка Django

1. Открыть терминал (командную строку).
2. Создать папку проекта:
   bash
   mkdir smart_diary
   cd smart_diary

3. Создать виртуальное окружение:

bash

python -m venv venv

4. Активировать виртуальное окружение:

- Windows:

bash

venv\Scripts\activate

- Mac/Linux:

bash

source venv/bin/activate

5. Установить Django:

bash

pip install django

6. Создать файл requirements.txt:

bash

pip freeze > requirements.txt

Шаг 2. Создание проекта Django

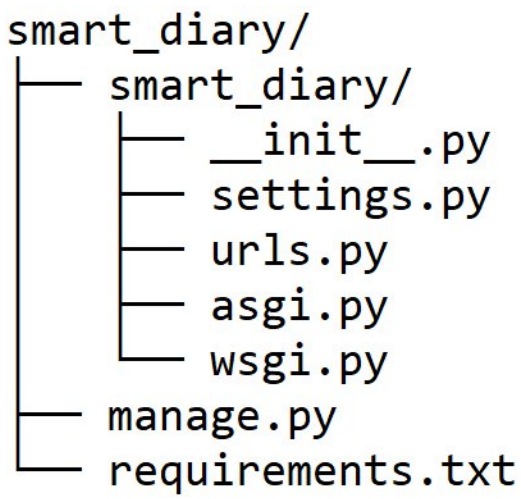
1. Создать проект:

bash

django-admin startproject smart_diary .

Обратите внимание на точку в конце — она создаст проект в текущей папке.

2. Проверить структуру:



Шаг 3. Создание приложения tasks

1. Создать приложение:

bash

python manage.py startapp tasks

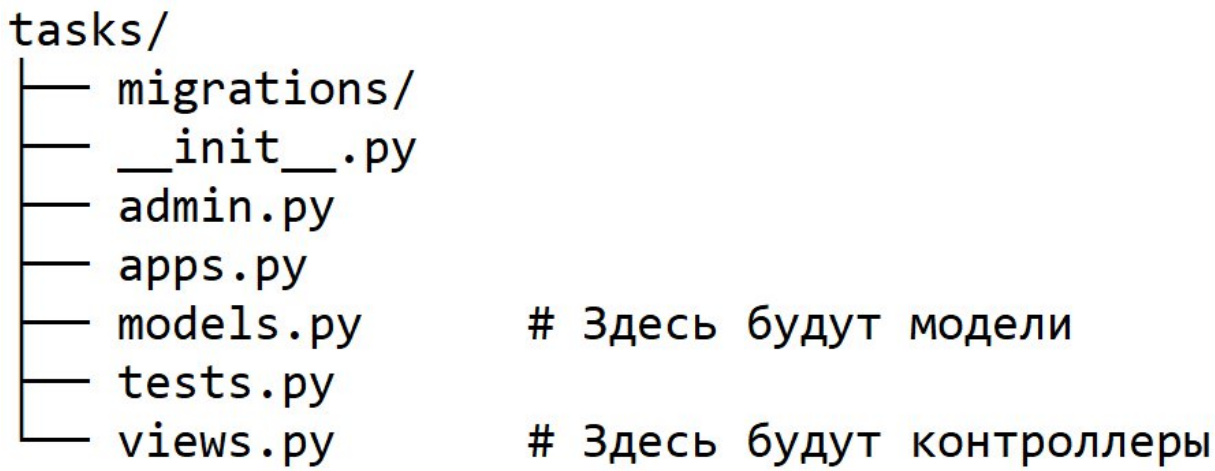
2. Добавить приложение в settings.py:

python

smart_diary/settings.py

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'tasks',   Добавить это
]
```

3. Изучить структуру приложения:



Шаг 4. Создание дополнительных папок

1. В папке tasks создать папку templates (для будущих Django-шаблонов, если понадобятся):

bash

mkdir tasks/templates

2. В папке tasks создать папку api (для будущих API-сериализаторов):

bash

mkdir tasks/api

Шаг 5. Настройка системы контроля версий

1. Инициализировать Git-репозиторий:

bash

git init

2. Создать файл .gitignore (можно взять с github/gitignore для Python).

3. Сделать первый коммит:

bash

git add .

git commit -m "Initial project structure for Smart Diary"

Шаг 6. Оформление отчета

В отчете по лабораторной работе должны быть:

- Титульный лист
- Цель работы
- Результаты аналитики (роли и сценарии)
- ER-диаграмма (скриншот или фото)
- Описание архитектуры MVC для Django+React проекта
- Скриншоты структуры проекта (дерево папок)
- Выводы

Контрольные вопросы

1. Что такое паттерн MVC? Опишите его компоненты.
2. Какие сущности были выделены в проекте и почему?
3. Объясните связи между таблицами в базе данных.
4. Какие роли пользователей есть в системе и чем они различаются?
5. Как в выбранной технологии (ASP.NET/Django) реализуется компонент Model?
6. Для чего нужна ER-диаграмма при разработке?

Дополнительные задания (для продвинутых)

1. Добавить в модель Task поле Recurrence (повторение задачи: ежедневно, еженедельно) и описать, как это повлияет на структуру БД.

2. Спроектировать модель Comment (комментарии к задачам) и добавить её в ER-диаграмму.

3. Описать, как будет реализована система уведомлений о приближающемся дедлайне (какие модели потребуются).

Лабораторная работа №2: Модели данных, Миграции и Начальное наполнение

Цель работы: Получить практические навыки создания моделей данных, настройки связей между ними с использованием ORM, выполнения миграций для создания базы данных и наполнения её тестовыми данными (seed data).

Задачи

1. Реализовать классы моделей (сущностей) с правильными типами данных.

2. Добавить атрибуты валидации на стороне сервера.

3. Настроить связи между моделями (One-to-Many, Many-to-Many).

4. Сгенерировать и применить миграции для создания схемы БД.

5. Реализовать автоматическое наполнение базы тестовыми данными при первом запуске.

Теоретическая часть (общая для всех)

ORM (Object-Relational Mapping)

ORM - технология, позволяющая работать с базой данных через объекты языка программирования, а не через SQL-запросы. В данной работе используются:

- ASP.NET: Entity Framework Core (EF Core)
- Django: Django ORM (встроенный)

Миграции

Миграции - это способ последовательного изменения схемы базы данных (создание таблиц, добавление колонок и т.д.) с возможностью отката изменений. Миграции создаются на основе изменений в классах моделей.

Seed Data (Начальные данные)

Seed data - это тестовые или справочные данные, которые автоматически добавляются в базу при её создании или первом запуске приложения. Это необходимо для:

- Тестирования функциональности без ручного ввода данных
- Наличия справочных данных (например, статусы задач)
- Демонстрации работы приложения

Валидация на стороне сервера

Валидация данных должна происходить как на клиенте (для удобства пользователя), так и на сервере (для безопасности). В данной работе реализуется серверная валидация с использованием атрибутов/декораторов.

Ход выполнения работы

Общие требования к моделям (для обеих траекторий)

Необходимо реализовать следующие модели (классы) в соответствии с ER-диаграммой из лабораторной работы №1:

Модель User (Пользователь)

Поле	Тип	Ограничения/Валидация
Id	int/автоинкремент	Первичный ключ
Username	string	Уникальное, не null, макс. 100 символов
Email	string	Уникальное, не null, формат email
PasswordHash	string	Не null
Avatar	string?	Путь к файлу, может быть null
CreatedAt	DateTime	Не null, значение по умолчанию - текущее время
Settings	string?	JSON-строка с настройками

Модель Project (Проект)

Поле	Тип	Ограничения/Валидация
Id	int/автоинкремент	Первичный ключ
Name	string	Не null, макс. 100 символов
Description	string?	Может быть null
Color	string	Не null, hex-код (RRGGBB)
CreatedAt	DateTime	Не null, значение по умолчанию - текущее время
OwnerId	int	Внешний ключ на User.Id

Модель Tag (Тег)

Поле	Тип	Ограничения/Валидация
Id	int/автоинкремент	Первичный ключ
Name	string	Не null, макс. 50 символов

OwnerId	int	Внешний ключ на User.Id

Модель Task (Задача)

Поле	Тип	Ограничения/Валидация
Id	int/автоинкремент	Первичный ключ
Title	string	Не null, макс. 200 символов
Description	string?	Может быть null
CreatedAt	DateTime	Не null, значение по умолчанию - текущее время
Deadline	DateTime?	Может быть null, если указан - не может быть в прошлом
Status	string	Не null, одно из: "New", "InProgress", "Completed"
Priority	string	Не null, одно из: "Low", "Medium", "High"
ProjectId	int?	Внешний ключ на Project.Id (может быть null)
UserId	int	Внешний ключ на User.Id

Связующая модель TaskTag (для связи многие-ко-многим)

Поле	**Тип**	**Ограничения/Валидация**
TaskId	int	Внешний ключ на Task.Id (часть составного ключа)
TagId	int	Внешний ключ на Tag.Id (часть составного ключа)

Для траектории ASP.NET (Entity Framework Core)

Шаг 1. Установка необходимых пакетов NuGet

1. Открыть проект SmartDiary.Web из предыдущей работы.
2. Установить пакеты для Entity Framework Core:

Tools → NuGet Package Manager → Package Manager Console

Выполнить команды:
Powershell

Install-Package Microsoft.EntityFrameworkCore.SqlServer
Install-Package Microsoft.EntityFrameworkCore.Tools
Install-Package Microsoft.EntityFrameworkCore.Design

Если планируется использовать SQLite:
powershell
Install-Package Microsoft.EntityFrameworkCore.Sqlite

Шаг 2. Реализация моделей с валидацией

Создайте классы в папке Models:

Models/User.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace SmartDiary.Web.Models
{
    public class User
    {
```

```
        public int Id { get; set; }

        [Required(ErrorMessage = "Имя пользователя обязательно")]
        [StringLength(100, MinimumLength = 3, ErrorMessage = "Имя должно содержать от 3 до 100 символов")]
        public string Username { get; set; }

        [Required(ErrorMessage = "Email обязателен")]
        [EmailAddress(ErrorMessage = "Некорректный формат email")]
        public string Email { get; set; }

        [Required]
        public string PasswordHash { get; set; }

        public string? Avatar { get; set; }

        [DataType(DataType.DateTime)]
        public DateTime CreatedAt { get; set; } = DateTime.UtcNow;

        public string? Settings { get; set; } // JSON с настройками

        // Навигационные свойства
        public ICollection<Project> Projects { get; set; } = new List<Project>();
        public ICollection<Task> Tasks { get; set; } = new List<Task>();
        public ICollection<Tag> Tags { get; set; } = new List<Tag>();
    }
}
```

Models/Project.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace SmartDiary.Web.Models
{
    public class Project
    {
        public int Id { get; set; }

        [Required(ErrorMessage = "Название проекта обязательно")]
        [StringLength(100, ErrorMessage = "Название не может превышать 100 символов")]
        public string Name { get; set; }

        public string? Description { get; set; }

        [Required]
        [RegularExpression("^([A-Fa-f0-9]{6})$", ErrorMessage = "Цвет должен быть в формате HEX (RRGGBB)")]
        public string Color { get; set; } = "808080"; // Серый по умолчанию

        [DataType(DataType.DateTime)]
        public DateTime CreatedAt { get; set; } = DateTime.UtcNow;

        // Внешний ключ
        public int OwnerId { get; set; }

        // Навигационные свойства
```

```
        [ForeignKey("OwnerId")]
        public User Owner { get; set; }

        public ICollection<Task> Tasks { get; set; } = new List<Task>();
    }
}
```

Models/Tag.cs

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace SmartDiary.Web.Models
{
    public class Tag
    {
        public int Id { get; set; }

        [Required(ErrorMessage = "Название тега обязательно")]
        [StringLength(50, ErrorMessage = "Название тега не может превышать 50 символов")]
        public string Name { get; set; }

        // Внешний ключ
        public int OwnerId { get; set; }

        // Навигационные свойства
        public User Owner { get; set; }

        // Связь многие-ко-многим с Task
```

```
        public ICollection<TaskTag> TaskTags { get; set; } = new
List<TaskTag>();
    }
}
```

Models/Task.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace SmartDiary.Web.Models
{
    public class Task
    {
        public int Id { get; set; }

        [Required(ErrorMessage = "Заголовок задачи обязателен")]
        [StringLength(200, ErrorMessage = "Заголовок не может превышать
200 символов")]
        public string Title { get; set; }

        public string? Description { get; set; }

        [DataType(DataType.DateTime)]
        public DateTime CreatedAt { get; set; } = DateTime.UtcNow;

        [DataType(DataType.DateTime)]
        [CustomValidation(typeof(Task), nameof(ValidateDeadline))]
```

```
        public DateTime? Deadline { get; set; }

        [Required]
       public string Status { get; set; } = "New"; // New, InProgress, Completed

        [Required]
        public string Priority { get; set; } = "Medium"; // Low, Medium, High

        // Внешние ключи
        public int? ProjectId { get; set; }
        public int UserId { get; set; }

        // Навигационные свойства
        [ForeignKey("ProjectId")]
        public Project? Project { get; set; }

        [ForeignKey("UserId")]
        public User User { get; set; }

        // Связь многие-ко-многим с Tag
        public ICollection<TaskTag> TaskTags { get; set; } = new
List<TaskTag>();

        // Кастомная валидация для дедлайна
        public static ValidationResult? ValidateDeadline(DateTime? deadline,
ValidationContext context)
        {
         if (deadline.HasValue && deadline.Value < DateTime.UtcNow.Date)
           {
            return new ValidationResult("Дедлайн не может быть в прошлом");
```

```
            }
            return ValidationResult.Success;
        }
    }
}
```

Models/TaskTag.cs (связующая модель)

```
using Microsoft.EntityFrameworkCore;

namespace SmartDiary.Web.Models
{
    [PrimaryKey(nameof(TaskId), nameof(TagId))] // Составной первичный ключ
    public class TaskTag
    {
        public int TaskId { get; set; }
        public int TagId { get; set; }

        // Навигационные свойства
        public Task Task { get; set; } = null!;
        public Tag Tag { get; set; } = null!;
    }
}
```

Шаг 3. Настройка контекста базы данных

В папке Data создайте класс контекста:

Data/ApplicationDbContext.cs

```
using Microsoft.EntityFrameworkCore;
using SmartDiary.Web.Models;

namespace SmartDiary.Web.Data
{
    public class ApplicationDbContext : DbContext
    {
        public
ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
            : base(options)
        {
        }

        // DbSet для каждой сущности
        public DbSet<User> Users { get; set; }
        public DbSet<Project> Projects { get; set; }
        public DbSet<Tag> Tags { get; set; }
        public DbSet<Task> Tasks { get; set; }
        public DbSet<TaskTag> TaskTags { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);

            // Настройка уникальности Email в User
            modelBuilder.Entity<User>()
                .HasIndex(u => u.Email)
                .IsUnique();
```

```
            // Настройка уникальности Username в User
            modelBuilder.Entity<User>()
                .HasIndex(u => u.Username)
                .IsUnique();

            // Настройка уникальности Name в рамках одного пользователя для Tag
            modelBuilder.Entity<Tag>()
                .HasIndex(t => new { t.Name, t.OwnerId })
                .IsUnique();

            // Настройка связей
            modelBuilder.Entity<Project>()
                .HasOne(p => p.Owner)
                .WithMany(u => u.Projects)
                .HasForeignKey(p => p.OwnerId)
                .OnDelete(DeleteBehavior.Cascade);

            modelBuilder.Entity<Tag>()
                .HasOne(t => t.Owner)
                .WithMany(u => u.Tags)
                .HasForeignKey(t => t.OwnerId)
                .OnDelete(DeleteBehavior.Cascade);

            modelBuilder.Entity<Task>()
                .HasOne(t => t.User)
                .WithMany(u => u.Tasks)
                .HasForeignKey(t => t.UserId)
                .OnDelete(DeleteBehavior.Cascade);
```

```
            modelBuilder.Entity<Task>()
                .HasOne(t => t.Project)
                .WithMany(p => p.Tasks)
                .HasForeignKey(t => t.ProjectId)
                .OnDelete(DeleteBehavior.SetNull); // При удалении проекта задача остаётся без проекта

            // Настройка связи многие-ко-многим через TaskTag
            modelBuilder.Entity<TaskTag>()
                .HasOne(tt => tt.Task)
                .WithMany(t => t.TaskTags)
                .HasForeignKey(tt => tt.TaskId)
                .OnDelete(DeleteBehavior.Cascade);

            modelBuilder.Entity<TaskTag>()
                .HasOne(tt => tt.Tag)
                .WithMany(t => t.TaskTags)
                .HasForeignKey(tt => tt.TagId)
                .OnDelete(DeleteBehavior.Cascade);
        }
    }
}
```

Шаг 4. Регистрация контекста в Program.cs

Отредактируйте Program.cs:

```
using Microsoft.EntityFrameworkCore;
using SmartDiary.Web.Data;
```

```
var builder = WebApplication.CreateBuilder(args);

// Добавление сервисов в контейнер
builder.Services.AddControllersWithViews();

// Регистрация контекста БД
// Вариант с SQL Server:
builder.Services.AddDbContext<ApplicationDbContext>(options =>
options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));

// Вариант с SQLite:
// builder.Services.AddDbContext<ApplicationDbContext>(options =>
// options.UseSqlite(builder.Configuration.GetConnectionString("SQLiteConnection")));

var app = builder.Build();

// Остальной код...
```

Шаг 5. Настройка строки подключения

В файле appsettings.json добавьте строку подключения:

```
json
{
  "ConnectionStrings": {
```

```
    "DefaultConnection":
"Server=(localdb)\\mssqllocaldb;Database=SmartDiaryDB;Trusted_Connection=True;MultipleActiveResultSets=true",
    "SQLiteConnection": "Data Source=smartdiary.db"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": ""
}
```

Шаг 6. Создание и применение миграций

В Package Manager Console выполните:

powershell

Создание миграции

```
Add-Migration InitialCreate
```

Применение миграции к БД

```
Update-Database
```

Если используется SQLite, команды те же.

Шаг 7. Реализация Seed Data

Добавьте метод для инициализации данных в Program.cs:

```
// Добавьте этот метод в Program.cs перед app.Run()
static void SeedData(IServiceProvider serviceProvider)
{
    using (var scope = serviceProvider.CreateScope())
    {
        var context = scope.ServiceProvider.GetRequiredService<ApplicationDbContext>();

        // Применяем миграции, если БД не создана
        context.Database.Migrate();

        // Проверяем, есть ли уже данные
        if (context.Users.Any())
        {
            return; // Данные уже есть
        }

        // Создаем тестового пользователя
        var user = new User
        {
            Username = "testuser",
            Email = "test@example.com",
            PasswordHash = "AQAAAAIAAYagAAAAE..." // Здесь будет хэш пароля, пока заглушка
        };
        context.Users.Add(user);
        context.SaveChanges(); // Сохраняем, чтобы получить Id пользователя

        // Создаем проекты
```

```
            var projects = new[]
            {
             new Project { Name = "Личные дела", Description = "Личные задачи",
Color = "FF5733", OwnerId = user.Id },
                new Project { Name = "Работа", Description = "Рабочие задачи",
Color = "33FF57", OwnerId = user.Id },
                new Project { Name = "Учеба", Description = "Учебные задачи",
Color = "3357FF", OwnerId = user.Id }
            };
            context.Projects.AddRange(projects);
            context.SaveChanges();

            // Создаем теги
            var tags = new[]
            {
                new Tag { Name = "Важное", OwnerId = user.Id },
                new Tag { Name = "Срочное", OwnerId = user.Id },
                new Tag { Name = "Идея", OwnerId = user.Id }
            };
            context.Tags.AddRange(tags);
            context.SaveChanges();

            // Создаем задачи
            var tasks = new[]
            {
                new Task
                {
                    Title = "Купить продукты",
                    Description = "Молоко, хлеб, яйца",
                    Status = "New",
```

```
        Priority = "Medium",
        UserId = user.Id,
        ProjectId = projects[0].Id,
        Deadline = DateTime.UtcNow.AddDays(1)
    },
    new Task
    {
        Title = "Сдать отчет",
        Description = "Подготовить квартальный отчет",
        Status = "InProgress",
        Priority = "High",
        UserId = user.Id,
        ProjectId = projects[1].Id,
        Deadline = DateTime.UtcNow.AddHours(5)
    },
    new Task
    {
        Title = "Прочитать книгу",
        Description = "Глава 3",
        Status = "New",
        Priority = "Low",
        UserId = user.Id,
        ProjectId = projects[2].Id,
        Deadline = null
    }
};
context.Tasks.AddRange(tasks);
context.SaveChanges();

// Связываем задачи с тегами
```

```
            var taskTags = new[]
            {
             new TaskTag { TaskId = tasks[0].Id, TagId = tags[1].Id }, // Продукты
- Срочное
               new TaskTag { TaskId = tasks[1].Id, TagId = tags[0].Id }, // Отчет -
Важное
               new TaskTag { TaskId = tasks[1].Id, TagId = tags[1].Id }, // Отчет -
Срочное
               new TaskTag { TaskId = tasks[2].Id, TagId = tags[2].Id }  // Книга -
Идея
            };
            context.TaskTags.AddRange(taskTags);
            context.SaveChanges();
        }
    }
```

И вызовите этот метод после построения приложения:

```
var app = builder.Build();

// Seed data
SeedData(app.Services);

// Остальной код...
app.Run();
```

Шаг 8. Проверка результатов

1. Запустите приложение (F5).

2. Поставьте точку останова в любом месте, чтобы убедиться, что код отработал.

3. Откройте SQL Server Object Explorer в Visual Studio и проверьте, что база данных создана и содержит таблицы и данные.

Для траектории Django + React

Шаг 1. Установка необходимых пакетов

Активируйте виртуальное окружение и установите дополнительные пакеты:

bash
pip install django
Для работы с изображениями (аватар)
pip install Pillow

Обновите requirements.txt:
bash
pip freeze > requirements.txt

Шаг 2. Реализация моделей с валидацией

Отредактируйте файл tasks/models.py:

from django.db import models
from django.contrib.auth.models import AbstractUser

```
from django.core.validators import MinLengthValidator, RegexValidator, ValidationError
from django.utils import timezone
import json
from datetime import date


 Кастомная модель пользователя (расширяем встроенную)
class User(AbstractUser):
    Добавляем дополнительные поля
    avatar = models.ImageField(upload_to='avatars/', null=True, blank=True)
    created_at = models.DateTimeField(auto_now_add=True)
    settings = models.JSONField(default=dict, blank=True)  JSON поле для настроек

    def __str__(self):
        return self.username

    class Meta:
        db_table = 'users'



class Project(models.Model):
    name = models.CharField(
        max_length=100,
        validators=[MinLengthValidator(3, "Название должно содержать минимум 3 символа")]
    )
    description = models.TextField(blank=True, null=True)
    color = models.CharField(
        max_length=7,
```

```
            default="808080",
            validators=[RegexValidator(
                regex='^([A-Fa-f0-9]{6})$',
                message='Цвет должен быть в формате HEX (RRGGBB)'
            )]
        )
        created_at = models.DateTimeField(auto_now_add=True)
        owner = models.ForeignKey(User, on_delete=models.CASCADE, related_name='projects')

        def __str__(self):
            return self.name

        class Meta:
            db_table = 'projects'
             Уникальность названия проекта в рамках пользователя
            unique_together = ['name', 'owner']


    class Tag(models.Model):
        name = models.CharField(max_length=50)
        owner = models.ForeignKey(User, on_delete=models.CASCADE, related_name='tags')

        def __str__(self):
            return self.name

        class Meta:
            db_table = 'tags'
             Уникальность названия тега в рамках пользователя
```

```
            unique_together = ['name', 'owner']


        class Task(models.Model):
            Статусы задачи
            class Status(models.TextChoices):
                NEW = 'New', 'Новая'
                IN_PROGRESS = 'InProgress', 'В работе'
                COMPLETED = 'Completed', 'Выполнена'

            Приоритеты
            class Priority(models.TextChoices):
                LOW = 'Low', 'Низкий'
                MEDIUM = 'Medium', 'Средний'
                HIGH = 'High', 'Высокий'

            title = models.CharField(
                max_length=200,
                validators=[MinLengthValidator(3, "Заголовок должен содержать минимум 3 символа")]
            )
            description = models.TextField(blank=True, null=True)
            created_at = models.DateTimeField(auto_now_add=True)
            deadline = models.DateTimeField(null=True, blank=True)
            status = models.CharField(
                max_length=20,
                choices=Status.choices,
                default=Status.NEW
            )
            priority = models.CharField(
```

```
            max_length=20,
            choices=Priority.choices,
            default=Priority.MEDIUM
        )
        project = models.ForeignKey(
            Project,
            on_delete=models.SET_NULL,
            null=True,
            blank=True,
            related_name='tasks'
        )
        user = models.ForeignKey(User, on_delete=models.CASCADE, related_name='tasks')
        tags = models.ManyToManyField(Tag, through='TaskTag', related_name='tasks')

        def clean(self):
            """Кастомная валидация"""
            if self.deadline and self.deadline.date() < timezone.now().date():
                raise ValidationError({'deadline': 'Дедлайн не может быть в прошлом'})

        def save(self, args, kwargs):
            self.clean()
            super().save(args, kwargs)

        def __str__(self):
            return self.title

        class Meta:
```

```
        db_table = 'tasks'
        ordering = ['-created_at']


class TaskTag(models.Model):
    """Промежуточная модель для связи Task и Tag"""
    task = models.ForeignKey(Task, on_delete=models.CASCADE)
    tag = models.ForeignKey(Tag, on_delete=models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)

    class Meta:
        db_table = 'task_tags'
        unique_together = ['task', 'tag']   Запрещаем дубликаты
```

Шаг 3. Настройка проекта для использования кастомной модели пользователя

В файле smart_diary/settings.py добавьте:

После импортов

```
import os
from pathlib import Path

Указываем кастомную модель пользователя
AUTH_USER_MODEL = 'tasks.User'

Настройки для медиа-файлов (аватары)
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```

Шаг 4. Создание и применение миграций

bash

Создаем миграции

```
python manage.py makemigrations
```

Применяем миграции

```
python manage.py migrate
```

Шаг 5. Регистрация моделей в админ-панели

Для удобства просмотра данных создайте файл tasks/admin.py:

```
from django.contrib import admin
from .models import User, Project, Tag, Task, TaskTag

@admin.register(User)
class UserAdmin(admin.ModelAdmin):
    list_display = ('username', 'email', 'created_at')
    search_fields = ('username', 'email')

@admin.register(Project)
class ProjectAdmin(admin.ModelAdmin):
    list_display = ('name', 'owner', 'created_at')
    list_filter = ('owner',)

@admin.register(Tag)
class TagAdmin(admin.ModelAdmin):
    list_display = ('name', 'owner')
```

```
    list_filter = ('owner',)

@admin.register(Task)
class TaskAdmin(admin.ModelAdmin):
    list_display = ('title', 'user', 'status', 'priority', 'deadline')
    list_filter = ('status', 'priority', 'user')
    search_fields = ('title', 'description')
    filter_horizontal = ('tags',)   Удобный виджет для выбора тегов

@admin.register(TaskTag)
class TaskTagAdmin(admin.ModelAdmin):
    list_display = ('task', 'tag', 'created_at')
```

Шаг 6. Создание суперпользователя для доступа к админке

bash

```
python manage.py createsuperuser
```

Введите имя пользователя, email и пароль

Шаг 7. Создание скрипта для начального наполнения (Seed Data)

Способ 1: Фикстуры (JSON)

1. Создайте файл tasks/fixtures/initial_data.json:

json

```
[
  {
```

```
    "model": "tasks.user",
    "pk": 1,
    "fields": {
      "username": "testuser",
      "email": "test@example.com",
      "password": "pbkdf2_sha256$..." // Здесь должен быть хэш пароля, но проще создать через скрипт
    }
  }
]
```

Загрузка фикстур:

```
bash
python manage.py loaddata initial_data.json
```

Способ 2: Кастомная команда Django (рекомендуемый)

Создайте структуру для кастомной команды:

```
bash
mkdir -p tasks/management/commands
touch tasks/management/__init__.py
touch tasks/management/commands/__init__.py
```

Создайте файл tasks/management/commands/seed_data.py:

```
from django.core.management.base import BaseCommand
from django.contrib.auth.hashers import make_password
```

```
from django.utils import timezone
from tasks.models import User, Project, Tag, Task, TaskTag

class Command(BaseCommand):
    help = 'Заполняет базу данных тестовыми данными'

    def handle(self, args, kwargs):
        Проверяем, есть ли уже пользователи
        if User.objects.exists():
            self.stdout.write(self.style.WARNING('Данные уже существуют. Пропускаем...'))
            return

        self.stdout.write(self.style.SUCCESS('Создание тестовых данных...'))

        Создаем тестового пользователя
        user = User.objects.create(
            username='testuser',
            email='test@example.com',
            password=make_password('testpass123'),  Хэшируем пароль
            settings={'theme': 'light', 'notifications': True}
        )
        self.stdout.write(f'Создан пользователь: {user.username}')

        Создаем проекты
        projects = []
        project_data = [
            ('Личные дела', 'Личные задачи и планы', 'FF5733'),
            ('Работа', 'Рабочие проекты', '33FF57'),
            ('Учеба', 'Курсы и обучение', '3357FF'),
```

```
]
for name, desc, color in project_data:
    project = Project.objects.create(
        name=name,
        description=desc,
        color=color,
        owner=user
    )
    projects.append(project)
    self.stdout.write(f'Создан проект: {project.name}')

 Создаем теги
tags = []
tag_names = ['Важное', 'Срочное', 'Идея', 'Личное', 'Рабочее']
for tag_name in tag_names:
    tag = Tag.objects.create(name=tag_name, owner=user)
    tags.append(tag)
    self.stdout.write(f'Создан тег: {tag.name}')

 Создаем задачи
tasks_data = [
    {
        'title': 'Купить продукты',
        'description': 'Молоко, хлеб, яйца, сыр',
        'status': Task.Status.NEW,
        'priority': Task.Priority.MEDIUM,
        'project': projects[0],
        'deadline': timezone.now() + timezone.timedelta(days=1),
        'tags': [tags[1], tags[3]]   Срочное, Личное
    },
```

```
    {
        'title': 'Сдать отчет',
        'description': 'Подготовить квартальный отчет для начальника',
        'status': Task.Status.IN_PROGRESS,
        'priority': Task.Priority.HIGH,
        'project': projects[1],
        'deadline': timezone.now() + timezone.timedelta(hours=5),
        'tags': [tags[0], tags[1], tags[4]]   Важное, Срочное, Рабочее
    },
    {
        'title': 'Прочитать книгу',
        'description': 'Глава 3 "Изучаем Python"',
        'status': Task.Status.NEW,
        'priority': Task.Priority.LOW,
        'project': projects[2],
        'deadline': None,
        'tags': [tags[2], tags[3]]   Идея, Личное
    },
    {
        'title': 'Позвонить маме',
        'description': '',
        'status': Task.Status.NEW,
        'priority': Task.Priority.MEDIUM,
        'project': None,   Без проекта
        'deadline': timezone.now() + timezone.timedelta(days=2),
        'tags': [tags[3]]   Личное
    },
]

for task_data in tasks_data:
```

```
            tags_list = task_data.pop('tags')
            task = Task.objects.create(
                user=user,
                task_data
            )
            task.tags.set(tags_list)   Добавляем теги через many-to-many
            self.stdout.write(f'Создана задача: {task.title}')

        self.stdout.write(self.style.SUCCESS(f'Успешно создано:'
                                             f' {User.objects.count()} пользователь,'
                                             f' {Project.objects.count()} проектов,'
                                             f' {Tag.objects.count()} тегов,'
                                             f' {Task.objects.count()} задач'))
```

Запустите команду:

bash

python manage.py seed_data

Шаг 8. Проверка результатов

1. Запустите сервер разработки:

   bash

   python manage.py runserver

2. Откройте админ-панель: http://127.0.0.1:8000/admin/

   - Войдите под суперпользователем

   - Проверьте, что все модели отображаются и содержат данные

3. Проверьте наличие данных в таблицах через админку.

Контрольные вопросы

1. Что такое ORM и зачем он нужен?
2. Чем отличаются связи One-to-Many и Many-to-Many? Приведите примеры из проекта.
3. Что такое миграции и для чего они используются?
4. Как обеспечивается уникальность полей (например, email) в БД?
5. Какие способы валидации данных на стороне сервера были использованы?
6. Чем отличается подход к seed data в ASP.NET и Django?
7. Что такое каскадное удаление (Cascade Delete) и когда его стоит применять?

Дополнительные задания (для продвинутых)

1. Индексы: Добавить индексы для часто запрашиваемых полей (например, Task.Status, Task.Deadline) и объяснить зачем.
2. Аудит: Добавить поля UpdatedAt во все модели и настроить автоматическое обновление при изменении записи.
3. Составные ключи: В Django реализовать составной первичный ключ в TaskTag через Meta.unique_together.
4. Расширенная валидация: Добавить проверку, что у задачи не может быть дедлайн раньше даты создания.

5. Транзакции: Обернуть создание seed data в транзакцию, чтобы в случае ошибки не оставались частично созданные данные.

Лабораторная работа №3

Реализация бизнес-логики. Контроллеры и сервисы

Цель работы: Получить практические навыки создания контроллеров для обработки HTTP-запросов, реализации CRUD-операций и внедрения сервисного слоя на основе паттерна MVC.

Задачи

№	Задача
1	Реализовать контроллеры для обработки запросов к основным сущностям (Задачи, Проекты, Теги)
2	Реализовать CRUD-операции для модели Task (Задача)
3	Реализовать CRUD-операции для моделей Project и Tag
4	Реализовать фильтрацию и пагинацию списков
5	Внедрить сервисный слой для отделения бизнес-логики от контроллеров
6	Настроить валидацию входных данных на стороне сервера

Теоретическая часть

Контроллеры в MVC

Controller - компонент, обрабатывающий входящие HTTP-запросы, взаимодействующий с моделями и возвращающий ответ клиенту.

CRUD-операции

Операция	HTTP метод	Типичный URL	Действие
Create	POST	/tasks/create	Создание новой записи
Read	GET	/tasks	Получение списка записей
Read (один)	GET	/tasks/{id}	Получение одной записи
Update	POST/PUT	/tasks/edit/{id}	Обновление записи
Delete	POST/DELETE	/tasks/delete/{id}	Удаление записи

Сервисный слой (Service Layer)

Сервисный слой - это промежуточный слой между контроллерами и репозиториями/моделями, который содержит бизнес-логику приложения.

Преимущества:

- Контроллеры становятся «тонкими» (только обработка запросов и ответов)
- Бизнес-логика переиспользуется между разными контроллерами
- Упрощается тестирование (можно тестировать сервисы независимо от HTTP)

Пагинация (Pagination)

Пагинация - разбиение списка записей на страницы для снижения нагрузки на сервер и улучшения UX.

Параметры пагинации:

- page - номер страницы
- page_size - количество записей на странице
- total - общее количество записей

Ход выполнения работы

Общие требования

1. Все операции с задачами, проектами и тегами доступны через API
2. Реализовать полноценный сервисный слой для бизнес-логики
3. Реализовать фильтрацию задач по статусу, приоритету и проекту
4. Реализовать пагинацию для списка задач
5. Настроить валидацию входных данных
6. **Аутентификация не требуется** — на данном этапе работаем с одним тестовым пользователем (например, admin)

Траектория A (Django + React)

Шаг 1. Создание сериализаторов

tasks/serializers.py:

```
from rest_framework import serializers
from .models import Task, Project, Tag

class TagSerializer(serializers.ModelSerializer):
    class Meta:
        model = Tag
        fields = ('id', 'name', 'owner_id')
        read_only_fields = ('id', 'owner_id')

class ProjectSerializer(serializers.ModelSerializer):
    class Meta:
        model = Project
        fields = ('id', 'name', 'description', 'color', 'created_at', 'owner_id')
        read_only_fields = ('id', 'created_at', 'owner_id')

class TaskSerializer(serializers.ModelSerializer):
    project = ProjectSerializer(read_only=True)
    project_id = serializers.PrimaryKeyRelatedField(
```

```
        queryset=Project.objects.all(),
        source='project',
        write_only=True,
        required=False,
        allow_null=True
    )
    tags = TagSerializer(many=True, read_only=True)
    tag_ids = serializers.PrimaryKeyRelatedField(
        queryset=Tag.objects.all(),
        source='tags',
        write_only=True,
        many=True,
        required=False
    )

    class Meta:
        model = Task
        fields = (
            'id', 'title', 'description', 'created_at', 'deadline',
            'status', 'priority', 'project', 'project_id', 'tags', 'tag_ids'
        )
        read_only_fields = ('id', 'created_at')

    def validate_title(self, value):
        if len(value.strip()) < 3:
            raise serializers.ValidationError("Заголовок должен содержать минимум 3 символа")
        return value

    def validate_deadline(self, value):
        if value and value < timezone.now():
            raise serializers.ValidationError("Дедлайн не может быть в прошлом")
        return value
```

Шаг 2. Создание сервисного слоя

tasks/services.py:

```
from django.db.models import Q
from django.core.paginator import Paginator
from .models import Task, Project, Tag
from .serializers import TaskSerializer, ProjectSerializer, TagSerializer

class TaskService:
    @staticmethod
    def get_all_tasks(user_id, filters=None, page=1, page_size=10):
        """Получение всех задач пользователя с фильтрацией и пагинацией"""
        queryset =
Task.objects.filter(user_id=user_id).select_related('project').prefetch_related('tags')

        # Фильтрация
```

```
    if filters:
        if filters.get('status'):
            queryset = queryset.filter(status=filters['status'])
        if filters.get('priority'):
            queryset = queryset.filter(priority=filters['priority'])
        if filters.get('project_id'):
            queryset = queryset.filter(project_id=filters['project_id'])
        if filters.get('search'):
            queryset = queryset.filter(
                Q(title__icontains=filters['search']) |
                Q(description__icontains=filters['search'])
            )

    # Пагинация
    paginator = Paginator(queryset, page_size)
    page_obj = paginator.get_page(page)

    return {
        'tasks': TaskSerializer(page_obj, many=True).data,
        'total': paginator.count,
        'page': page,
        'page_size': page_size,
        'total_pages': paginator.num_pages
    }

@staticmethod
def get_task_by_id(task_id, user_id):
    """Получение задачи по ID"""
    try:
        return Task.objects.select_related('project').prefetch_related('tags').get(
            id=task_id, user_id=user_id
        )
    except Task.DoesNotExist:
        return None

@staticmethod
def create_task(data, user_id):
    """Создание новой задачи"""
    data['user_id'] = user_id
    tags = data.pop('tag_ids', [])

    task = Task.objects.create(**data)

    if tags:
        task.tags.set(tags)

    return task

@staticmethod
def update_task(task_id, user_id, data):
```

```
        """Обновление задачи"""
        try:
            task = Task.objects.get(id=task_id, user_id=user_id)
        except Task.DoesNotExist:
            return None

        tags = data.pop('tag_ids', None)

        for key, value in data.items():
            setattr(task, key, value)

        task.save()

        if tags is not None:
            task.tags.set(tags)

        return task

    @staticmethod
    def delete_task(task_id, user_id):
        """Удаление задачи"""
        try:
            task = Task.objects.get(id=task_id, user_id=user_id)
            task.delete()
            return True
        except Task.DoesNotExist:
            return False


class ProjectService:
    @staticmethod
    def get_all_projects(user_id):
        return Project.objects.filter(owner_id=user_id)

    @staticmethod
    def create_project(data, user_id):
        data['owner_id'] = user_id
        return Project.objects.create(**data)

    @staticmethod
    def update_project(project_id, user_id, data):
        try:
            project = Project.objects.get(id=project_id, owner_id=user_id)
            for key, value in data.items():
                setattr(project, key, value)
            project.save()
            return project
        except Project.DoesNotExist:
            return None

    @staticmethod
```

```
    def delete_project(project_id, user_id):
        try:
            project = Project.objects.get(id=project_id, owner_id=user_id)
            project.delete()
            return True
        except Project.DoesNotExist:
            return False


class TagService:
    @staticmethod
    def get_all_tags(user_id):
        return Tag.objects.filter(owner_id=user_id)

    @staticmethod
    def create_tag(data, user_id):
        data['owner_id'] = user_id
        return Tag.objects.create(**data)

    @staticmethod
    def update_tag(tag_id, user_id, data):
        try:
            tag = Tag.objects.get(id=tag_id, owner_id=user_id)
            tag.name = data.get('name', tag.name)
            tag.save()
            return tag
        except Tag.DoesNotExist:
            return None

    @staticmethod
    def delete_tag(tag_id, user_id):
        try:
            tag = Tag.objects.get(id=tag_id, owner_id=user_id)
            tag.delete()
            return True
        except Tag.DoesNotExist:
            return False
```

Шаг 3. Создание ViewSet для API

tasks/views.py:

```
from rest_framework import viewsets, status
from rest_framework.decorators import action
from rest_framework.response import Response
from rest_framework.permissions import AllowAny  # Временное разрешение для
тестирования
from .models import Task, Project, Tag
from .serializers import TaskSerializer, ProjectSerializer, TagSerializer
from .services import TaskService, ProjectService, TagService
```

```python
# Временный ID пользователя (пока без аутентификации)
TEST_USER_ID = 1

class TaskViewSet(viewsets.ViewSet):
    permission_classes = [AllowAny]  # Временно отключаем аутентификацию

    def list(self, request):
        """GET /api/tasks/ - список задач с фильтрацией и пагинацией"""
        filters = {
            'status': request.query_params.get('status'),
            'priority': request.query_params.get('priority'),
            'project_id': request.query_params.get('project_id'),
            'search': request.query_params.get('search'),
        }
        # Убираем None значения
        filters = {k: v for k, v in filters.items() if v}

        page = int(request.query_params.get('page', 1))
        page_size = int(request.query_params.get('page_size', 10))

        result = TaskService.get_all_tasks(TEST_USER_ID, filters, page, page_size)
        return Response(result)

    def retrieve(self, request, pk=None):
        """GET /api/tasks/{id}/ - детали задачи"""
        task = TaskService.get_task_by_id(pk, TEST_USER_ID)
        if not task:
            return Response({'error': 'Задача не найдена'}, status=status.HTTP_404_NOT_FOUND)
        return Response(TaskSerializer(task).data)

    def create(self, request):
        """POST /api/tasks/ - создание задачи"""
        serializer = TaskSerializer(data=request.data)
        if not serializer.is_valid():
            return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

        task = TaskService.create_task(serializer.validated_data, TEST_USER_ID)
        return Response(TaskSerializer(task).data, status=status.HTTP_201_CREATED)

    def update(self, request, pk=None):
        """PUT /api/tasks/{id}/ - обновление задачи"""
        serializer = TaskSerializer(data=request.data)
        if not serializer.is_valid():
            return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

        task = TaskService.update_task(pk, TEST_USER_ID, serializer.validated_data)
        if not task:
            return Response({'error': 'Задача не найдена'}, status=status.HTTP_404_NOT_FOUND)
        return Response(TaskSerializer(task).data)
```

```python
    def destroy(self, request, pk=None):
        """DELETE /api/tasks/{id}/ - удаление задачи"""
        success = TaskService.delete_task(pk, TEST_USER_ID)
        if not success:
            return Response({'error': 'Задача не найдена'}, status=status.HTTP_404_NOT_FOUND)
        return Response(status=status.HTTP_204_NO_CONTENT)


class ProjectViewSet(viewsets.ViewSet):
    permission_classes = [AllowAny]

    def list(self, request):
        projects = ProjectService.get_all_projects(TEST_USER_ID)
        return Response(ProjectSerializer(projects, many=True).data)

    def create(self, request):
        serializer = ProjectSerializer(data=request.data)
        if not serializer.is_valid():
            return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

        project = ProjectService.create_project(serializer.validated_data, TEST_USER_ID)
        return Response(ProjectSerializer(project).data, status=status.HTTP_201_CREATED)

    def update(self, request, pk=None):
        serializer = ProjectSerializer(data=request.data)
        if not serializer.is_valid():
            return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

        project = ProjectService.update_project(pk, TEST_USER_ID, serializer.validated_data)
        if not project:
            return Response({'error': 'Проект не найден'}, status=status.HTTP_404_NOT_FOUND)
        return Response(ProjectSerializer(project).data)

    def destroy(self, request, pk=None):
        success = ProjectService.delete_project(pk, TEST_USER_ID)
        if not success:
            return Response({'error': 'Проект не найден'}, status=status.HTTP_404_NOT_FOUND)
        return Response(status=status.HTTP_204_NO_CONTENT)


class TagViewSet(viewsets.ViewSet):
    permission_classes = [AllowAny]

    def list(self, request):
        tags = TagService.get_all_tags(TEST_USER_ID)
        return Response(TagSerializer(tags, many=True).data)

    def create(self, request):
        serializer = TagSerializer(data=request.data)
        if not serializer.is_valid():
            return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

```
        tag = TagService.create_tag(serializer.validated_data, TEST_USER_ID)
        return Response(TagSerializer(tag).data, status=status.HTTP_201_CREATED)

    def update(self, request, pk=None):
        serializer = TagSerializer(data=request.data)
        if not serializer.is_valid():
            return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

        tag = TagService.update_tag(pk, TEST_USER_ID, serializer.validated_data)
        if not tag:
            return Response({'error': 'Тег не найден'}, status=status.HTTP_404_NOT_FOUND)
        return Response(TagSerializer(tag).data)

    def destroy(self, request, pk=None):
        success = TagService.delete_tag(pk, TEST_USER_ID)
        if not success:
            return Response({'error': 'Тег не найден'}, status=status.HTTP_404_NOT_FOUND)
        return Response(status=status.HTTP_204_NO_CONTENT)
```

Шаг 4. Настройка маршрутов

tasks/urls.py

```
from django.urls import path, include
from rest_framework.routers import DefaultRouter
from . import views

router = DefaultRouter()
router.register('tasks', views.TaskViewSet, basename='task')
router.register('projects', views.ProjectViewSet, basename='project')
router.register('tags', views.TagViewSet, basename='tag')

urlpatterns = [
    path('api/', include(router.urls)),
]
```

Шаг 5. Тестирование API

```
# Получение списка задач (с пагинацией)
curl -X GET "http://localhost:8000/api/tasks/?page=1&page_size=10&status=New"

# Получение деталей задачи
curl -X GET http://localhost:8000/api/tasks/1/

# Создание задачи
curl -X POST http://localhost:8000/api/tasks/ \
  -H "Content-Type: application/json" \
  -d '{"title":"Новая задача","priority":"High","status":"New","project_id":1}'
```

```
# Обновление задачи
curl -X PUT http://localhost:8000/api/tasks/1/ \
  -H "Content-Type: application/json" \
  -d '{"title":"Обновленная задача","priority":"Medium","status":"InProgress"}'

# Удаление задачи
curl -X DELETE http://localhost:8000/api/tasks/1/
```

Траектория Б (ASP.NET Core + React)

Шаг 1. Создание сервисного слоя

Services/ITaskService.cs:

```
public interface ITaskService
{
    Task<PagedResult<TaskDto>> GetUserTasksAsync(
        int userId,
        TaskFilterDto filter,
        int page,
        int pageSize);
    Task<TaskDto?> GetTaskByIdAsync(int id, int userId);
    Task<TaskDto> CreateTaskAsync(CreateTaskDto dto, int userId);
    Task<TaskDto?> UpdateTaskAsync(int id, UpdateTaskDto dto, int userId);
    Task<bool> DeleteTaskAsync(int id, int userId);
}
```

Services/TaskService.cs:

```
using Microsoft.EntityFrameworkCore;

public class TaskService : ITaskService
{
    private readonly ApplicationDbContext _context;

    public TaskService(ApplicationDbContext context)
    {
        _context = context;
    }

    public async Task<PagedResult<TaskDto>> GetUserTasksAsync(
        int userId,
        TaskFilterDto filter,
        int page,
        int pageSize)
    {
        var query = _context.Tasks
            .Include(t => t.Project)
            .Include(t => t.TaskTags)
```

```
            .ThenInclude(tt => tt.Tag)
        .Where(t => t.UserId == userId)
        .AsQueryable();

    // Фильтрация
    if (!string.IsNullOrEmpty(filter.Status))
        query = query.Where(t => t.Status == filter.Status);

    if (!string.IsNullOrEmpty(filter.Priority))
        query = query.Where(t => t.Priority == filter.Priority);

    if (filter.ProjectId.HasValue)
        query = query.Where(t => t.ProjectId == filter.ProjectId);

    if (!string.IsNullOrEmpty(filter.Search))
        query = query.Where(t =>
            t.Title.Contains(filter.Search) ||
            (t.Description != null && t.Description.Contains(filter.Search)));

    var total = await query.CountAsync();
    var items = await query
        .Skip((page - 1) * pageSize)
        .Take(pageSize)
        .Select(t => new TaskDto
        {
            Id = t.Id,
            Title = t.Title,
            Description = t.Description,
            Status = t.Status,
            Priority = t.Priority,
            Deadline = t.Deadline,
            CreatedAt = t.CreatedAt,
            Project = t.Project != null ? new ProjectDto
            {
                Id = t.Project.Id,
                Name = t.Project.Name,
                Color = t.Project.Color
            } : null,
            Tags = t.TaskTags.Select(tt => new TagDto
            {
                Id = tt.Tag.Id,
                Name = tt.Tag.Name
            }).ToList()
        })
        .ToListAsync();

    return new PagedResult<TaskDto>
    {
        Items = items,
        Total = total,
```

```
            Page = page,
            PageSize = pageSize,
            TotalPages = (int)Math.Ceiling(total / (double)pageSize)
        };
    }

    public async Task<TaskDto?> GetTaskByIdAsync(int id, int userId)
    {
        var task = await _context.Tasks
            .Include(t => t.Project)
            .Include(t => t.TaskTags)
                .ThenInclude(tt => tt.Tag)
            .FirstOrDefaultAsync(t => t.Id == id && t.UserId == userId);

        if (task == null) return null;

        return new TaskDto
        {
            Id = task.Id,
            Title = task.Title,
            Description = task.Description,
            Status = task.Status,
            Priority = task.Priority,
            Deadline = task.Deadline,
            CreatedAt = task.CreatedAt,
            Project = task.Project != null ? new ProjectDto
            {
                Id = task.Project.Id,
                Name = task.Project.Name,
                Color = task.Project.Color
            } : null,
            Tags = task.TaskTags.Select(tt => new TagDto
            {
                Id = tt.Tag.Id,
                Name = tt.Tag.Name
            }).ToList()
        };
    }

    public async Task<TaskDto> CreateTaskAsync(CreateTaskDto dto, int userId)
    {
        var task = new Task
        {
            Title = dto.Title,
            Description = dto.Description,
            Deadline = dto.Deadline,
            Status = dto.Status,
            Priority = dto.Priority,
            ProjectId = dto.ProjectId,
            UserId = userId,
```

```
            CreatedAt = DateTime.UtcNow
        };

        _context.Tasks.Add(task);
        await _context.SaveChangesAsync();

        if (dto.TagIds != null && dto.TagIds.Any())
        {
            foreach (var tagId in dto.TagIds)
            {
                _context.TaskTags.Add(new TaskTag { TaskId = task.Id, TagId = tagId });
            }
            await _context.SaveChangesAsync();
        }

        return await GetTaskByIdAsync(task.Id, userId);
    }

    public async Task<TaskDto?> UpdateTaskAsync(int id, UpdateTaskDto dto, int userId)
    {
        var task = await _context.Tasks
            .Include(t => t.TaskTags)
            .FirstOrDefaultAsync(t => t.Id == id && t.UserId == userId);

        if (task == null) return null;

        task.Title = dto.Title;
        task.Description = dto.Description;
        task.Deadline = dto.Deadline;
        task.Status = dto.Status;
        task.Priority = dto.Priority;
        task.ProjectId = dto.ProjectId;

        // Обновление тегов
        if (dto.TagIds != null)
        {
            _context.TaskTags.RemoveRange(task.TaskTags);

            foreach (var tagId in dto.TagIds)
            {
                _context.TaskTags.Add(new TaskTag { TaskId = task.Id, TagId = tagId });
            }
        }

        await _context.SaveChangesAsync();
        return await GetTaskByIdAsync(id, userId);
    }

    public async Task<bool> DeleteTaskAsync(int id, int userId)
    {
```

```
        var task = await _context.Tasks
            .FirstOrDefaultAsync(t => t.Id == id && t.UserId == userId);

        if (task == null) return false;

        _context.Tasks.Remove(task);
        await _context.SaveChangesAsync();
        return true;
    }
}
```

Шаг 2. Регистрация сервиса в Program.cs

```
builder.Services.AddScoped<ITaskService, TaskService>();
builder.Services.AddScoped<IProjectService, ProjectService>();
builder.Services.AddScoped<ITagService, TagService>();
```

Шаг 3. Создание DTO (Data Transfer Objects)

DTOs/TaskDto.cs:

```
public class TaskDto
{
    public int Id { get; set; }
    public string Title { get; set; } = string.Empty;
    public string? Description { get; set; }
    public DateTime CreatedAt { get; set; }
    public DateTime? Deadline { get; set; }
    public string Status { get; set; } = string.Empty;
    public string Priority { get; set; } = string.Empty;
    public ProjectDto? Project { get; set; }
    public List<TagDto> Tags { get; set; } = new();
}

public class CreateTaskDto
{
    public string Title { get; set; } = string.Empty;
    public string? Description { get; set; }
    public DateTime? Deadline { get; set; }
    public string Status { get; set; } = "New";
    public string Priority { get; set; } = "Medium";
    public int? ProjectId { get; set; }
    public List<int> TagIds { get; set; } = new();
}

public class UpdateTaskDto
{
    public string Title { get; set; } = string.Empty;
```

```
    public string? Description { get; set; }
    public DateTime? Deadline { get; set; }
    public string Status { get; set; } = string.Empty;
    public string Priority { get; set; } = string.Empty;
    public int? ProjectId { get; set; }
    public List<int> TagIds { get; set; } = new();
}

public class TaskFilterDto
{
    public string? Status { get; set; }
    public string? Priority { get; set; }
    public int? ProjectId { get; set; }
    public string? Search { get; set; }
}

public class PagedResult<T>
{
    public List<T> Items { get; set; } = new();
    public int Total { get; set; }
    public int Page { get; set; }
    public int PageSize { get; set; }
    public int TotalPages { get; set; }
}
```

Шаг 4. Создание API контроллера

Controllers/TaskController.cs:

```
using Microsoft.AspNetCore.Mvc;

[ApiController]
[Route("api/[controller]")]
public class TaskController : ControllerBase
{
    private readonly ITaskService _taskService;

    public TaskController(ITaskService taskService)
    {
        _taskService = taskService;
    }

    // GET: api/task?page=1&pageSize=10&status=New
    [HttpGet]
    public async Task<ActionResult<PagedResult<TaskDto>>> GetTasks(
        [FromQuery] int page = 1,
        [FromQuery] int pageSize = 10,
        [FromQuery] string? status = null,
        [FromQuery] string? priority = null,
        [FromQuery] int? projectId = null,
```

```
    [FromQuery] string? search = null)
{
    // Временный ID пользователя (пока без аутентификации)
    int userId = 1;

    var filter = new TaskFilterDto
    {
        Status = status,
        Priority = priority,
        ProjectId = projectId,
        Search = search
    };

    var result = await _taskService.GetUserTasksAsync(userId, filter, page, pageSize);
    return Ok(result);
}

// GET: api/task/5
[HttpGet("{id}")]
public async Task<ActionResult<TaskDto>> GetTask(int id)
{
    int userId = 1;
    var task = await _taskService.GetTaskByIdAsync(id, userId);

    if (task == null)
        return NotFound();

    return Ok(task);
}

// POST: api/task
[HttpPost]
public async Task<ActionResult<TaskDto>> CreateTask(CreateTaskDto dto)
{
    int userId = 1;
    var task = await _taskService.CreateTaskAsync(dto, userId);
    return CreatedAtAction(nameof(GetTask), new { id = task.Id }, task);
}

// PUT: api/task/5
[HttpPut("{id}")]
public async Task<ActionResult<TaskDto>> UpdateTask(int id, UpdateTaskDto dto)
{
    int userId = 1;
    var task = await _taskService.UpdateTaskAsync(id, dto, userId);

    if (task == null)
        return NotFound();

    return Ok(task);
```

```
    }

    // DELETE: api/task/5
    [HttpDelete("{id}")]
    public async Task<IActionResult> DeleteTask(int id)
    {
        int userId = 1;
        var success = await _taskService.DeleteTaskAsync(id, userId);

        if (!success)
            return NotFound();

        return NoContent();
    }
}
```

Контрольные вопросы

1. Что такое CRUD и какие операции к нему относятся?
2. Для чего нужен сервисный слой? Какие преимущества он даёт?
3. Что такое пагинация и зачем она нужна?
4. Какие параметры пагинации обычно используются?
5. Как реализовать фильтрацию данных на бэкенде?
6. В чем разница между DTO и Model?

Критерии оценки

Критерий	Баллы
Реализованы CRUD-операции для задач	3
Реализованы CRUD-операции для проектов и тегов	2
Реализован сервисный слой	2
Реализована фильтрация и пагинация	2
Код работает без ошибок	1

Критерий	Баллы
Итого	10

Дополнительные задания (для продвинутых)

1. Сортировка: Добавить возможность сортировки списка задач по дате создания, приоритету и дедлайну
2. Расширенная фильтрация: Добавить фильтрацию по диапазону дат (дедлайн с ... по ...)
3. Мягкое удаление: Реализовать soft delete (поле is_deleted вместо физического удаления)
4. Кэширование: Добавить кэширование списка проектов и тегов (так как они меняются редко)
5. Логирование: Добавить логирование всех операций в сервисном слое

Результат выполнения

После выполнения лабораторной работы №3 должно быть:

1. Полноценный REST API для управления задачами, проектами и тегами
2. Сервисный слой с отделённой бизнес-логикой
3. Фильтрацию и пагинацию для списка задач
4. Валидацию входных данных на уровне API

На данном этапе аутентификация отсутствует, все запросы работают с тестовым пользователем (ID = 1). Аутентификация будет добавлена в следующей лабораторной работе (ЛР №4).

Лабораторная работа №4

Интеграция аутентификации. Frontend + Backend (JWT)

Цель работы: Интегрировать систему аутентификации, созданную в ЛР №3, с React-фронтендом: реализовать формы входа и регистрации, хранение JWT-токенов, защиту маршрутов и автоматическое добавление токена к запросам.

Задачи

№	Задача
1	Реализовать формы регистрации и входа с валидацией (Formik + Yup)
2	Настроить хранение JWT-токена (access + refresh) в localStorage / cookies
3	Реализовать защищенные маршруты (PrivateRoute) в React Router
4	Настроить автоматическое добавление Authorization header к API-запросам (axios interceptor)
5	Реализовать автоматическое обновление access-токена через refresh-токен
6	Реализовать выход из системы (logout)

Теоретическая часть

JWT (JSON Web Token)

JWT - компактный URL-безопасный токен, используемый для передачи информации между сторонами.

Структура JWT:

Header.Payload.Signature

Типы токенов:

Access Token - короткоживущий токен для доступа к API (обычно 15-60 минут)

Refresh Token - долгоживущий токен для получения нового access-токена (обычно 7-30 дней)

Стратегии хранения токенов на клиенте

Способ	Преимущества	Недостатки
localStorage	Простота, доступность из JS	Уязвим к XSS-атакам
sessionStorage	Ограничен сессией вкладки	Уязвим к XSS
HttpOnly cookies	Защита от XSS	Уязвим к CSRF, сложнее для SPA

Axios Interceptors

Interceptors позволяют перехватывать и обрабатывать запросы до их отправки или ответы до их обработки.

```
// Request interceptor
axios.interceptors.request.use(config => {
  const token = localStorage.getItem('access_token');
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});

// Response interceptor
axios.interceptors.response.use(
  response => response,
  async error => {
    if (error.response?.status === 401) {
      // Обновление токена
    }
    return Promise.reject(error);
```

```
  }
);
```

Ход выполнения работы

Общие требования

1. Фронтенд должен корректно обрабатывать JWT-токены, полученные от бэкенда
2. При входе пользователя токены должны сохраняться на клиенте
3. При выполнении запросов к защищенным API токен должен автоматически подставляться в заголовок Authorization: Bearer <token>
4. При истечении access-токена должен автоматически запрашиваться новый через refresh-токен
5. При выходе из системы токены должны удаляться

Траектория A (Django + React)

Шаг 1. Подготовка бэкенда (проверка JWT эндпоинтов)

Убедитесь, что в вашем Django-проекте настроены эндпоинты JWT:

```
# tasks/urls.py
from rest_framework_simplejwt.views import (
    TokenObtainPairView,
    TokenRefreshView,
    TokenVerifyView
)

urlpatterns = [
    # ... другие маршруты
    path('api/token/', TokenObtainPairView.as_view(), name='token_obtain_pair'),
    path('api/token/refresh/', TokenRefreshView.as_view(), name='token_refresh'),
    path('api/token/verify/', TokenVerifyView.as_view(), name='token_verify'),
]
```

Шаг 2. Создание API-сервисов для аутентификации

frontend/src/services/authService.js:

```
import axios from 'axios';

const API_URL = 'http://localhost:8000/api';

const authService = {
    // Регистрация
    async register(username, email, password, password2) {
        const response = await axios.post( `${API_URL}/auth/register/ `, {
            username,
            email,
            password,
            password2
        });
        return response.data;
    },

    // Вход
    async login(username, password) {
        const response = await axios.post( `${API_URL}/auth/login/ `, {
            username,
            password
        });

        if (response.data.user) {
            // Сохраняем данные пользователя
            localStorage.setItem('user', JSON.stringify(response.data.user));
        }

        return response.data;
    },

    // Получение JWT токенов
    async getTokens(username, password) {
        const response = await axios.post( `${API_URL}/token/ `, {
            username,
            password
        });

        if (response.data.access) {
            localStorage.setItem('access_token', response.data.access);
            localStorage.setItem('refresh_token', response.data.refresh);
        }

        return response.data;
    },

    // Обновление access токена
    async refreshToken() {
        const refresh = localStorage.getItem('refresh_token');
        if (!refresh) return null;
```

```
    try {
      const response = await axios.post( `${API_URL}/token/refresh/`, {
        refresh
      });

      if (response.data.access) {
        localStorage.setItem('access_token', response.data.access);
      }

      return response.data;
    } catch (error) {
      this.logout();
      return null;
    }
  },

  // Выход
  logout() {
    localStorage.removeItem('access_token');
    localStorage.removeItem('refresh_token');
    localStorage.removeItem('user');
  },

  // Получение текущего пользователя
  getCurrentUser() {
    const userStr = localStorage.getItem('user');
    if (userStr) {
      return JSON.parse(userStr);
    }
    return null;
  },

  // Проверка авторизации
  isAuthenticated() {
    return !!localStorage.getItem('access_token');
  }
};

export default authService;
```

Шаг 3. Настройка axios с перехватчиками

frontend/src/services/api.js:

```
import axios from 'axios';
import authService from './authService';

const api = axios.create({
    baseURL: 'http://localhost:8000/api',
    headers: {
        'Content-Type': 'application/json',
    },
});

// Request interceptor — добавляем токен к каждому запросу
api.interceptors.request.use(
    (config) => {
        const token = localStorage.getItem('access_token');
        if (token) {
            config.headers.Authorization = `Bearer ${token}`;
        }
        return config;
    },
    (error) => {
        return Promise.reject(error);
    }
);

// Response interceptor — обрабатываем ошибку 401 (неавторизован)
api.interceptors.response.use(
    (response) => response,
    async (error) => {
        const originalRequest = error.config;

        if (error.response?.status === 401 && !originalRequest._retry) {
            originalRequest._retry = true;

            try {
                const newTokens = await authService.refreshToken();
                if (newTokens) {
                    originalRequest.headers.Authorization = `Bearer ${newTokens.access}`;
                    return api(originalRequest);
                }
            } catch (refreshError) {
                authService.logout();
                window.location.href = '/login';
                return Promise.reject(refreshError);
            }
        }

        return Promise.reject(error);
    }
);
```

```
export default api;
```

Шаг 4. Создание контекста аутентификации

frontend/src/context/AuthContext.js:

```
import React, { createContext, useState, useContext, useEffect } from 'react';
import authService from '../services/authService';
import api from '../services/api';

const AuthContext = createContext();

export const useAuth = () => useContext(AuthContext);

export const AuthProvider = ({ children }) => {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    // Проверяем сохраненного пользователя при загрузке
    const currentUser = authService.getCurrentUser();
    const isAuthenticated = authService.isAuthenticated();

    if (currentUser && isAuthenticated) {
      setUser(currentUser);
    }
    setLoading(false);
  }, []);

  const register = async (username, email, password, password2) => {
    try {
      await authService.register(username, email, password, password2);
      // После регистрации выполняем вход
      return await login(username, password);
    } catch (error) {
      throw error;
    }
  };

  const login = async (username, password) => {
    try {
      // Сначала получаем JWT токены
      await authService.getTokens(username, password);
      // Затем получаем данные пользователя
      const response = await api.post('/auth/login/', { username, password });

      if (response.data.user) {
        setUser(response.data.user);
        return { success: true };
```

```
        }
        return { success: false, error: 'Ошибка входа' };
      } catch (error) {
        return {
          success: false,
          error: error.response?.data?.detail || 'Ошибка входа'
        };
      }
    };

    const logout = () => {
      authService.logout();
      setUser(null);
    };

    const value = {
      user,
      loading,
      register,
      login,
      logout,
      isAuthenticated: !!user && authService.isAuthenticated()
    };

    return (
      <AuthContext.Provider value={value}>
        {children}
      </AuthContext.Provider>
    );
};
```

Шаг 5. Создание компонента регистрации

frontend/src/components/RegisterForm.js:

```
import React, { useState } from 'react';
import { useFormik } from 'formik';
import * as Yup from 'yup';
import { useAuth } from '../context/AuthContext';
import { useNavigate, Link } from 'react-router-dom';
import {
  TextField,
  Button,
  Container,
  Typography,
  Box,
  Alert,
  Paper
} from '@mui/material';
```

```
const validationSchema = Yup.object({
  username: Yup.string()
    .min(3, 'Имя пользователя должно содержать минимум 3 символа')
    .max(100, 'Имя пользователя не может превышать 100 символов')
    .required('Имя пользователя обязательно'),
  email: Yup.string()
    .email('Некорректный формат email')
    .required('Email обязателен'),
  password: Yup.string()
    .min(6, 'Пароль должен содержать минимум 6 символов')
    .required('Пароль обязателен'),
  password2: Yup.string()
    .oneOf([Yup.ref('password'), null], 'Пароли не совпадают')
    .required('Подтверждение пароля обязательно')
});

const RegisterForm = () => {
  const { register } = useAuth();
  const navigate = useNavigate();
  const [error, setError] = useState('');

  const formik = useFormik({
    initialValues: {
      username: '',
      email: '',
      password: '',
      password2: ''
    },
    validationSchema: validationSchema,
    onSubmit: async (values) => {
      try {
        const result = await register(
          values.username,
          values.email,
          values.password,
          values.password2
        );
        if (result?.success) {
          navigate('/tasks');
        }
      } catch (err) {
        setError(err.response?.data?.detail || 'Ошибка регистрации');
      }
    }
  });

  return (
    <Container maxWidth="sm">
      <Paper elevation={3} sx={{ p: 4, mt: 8 }}>
        <Typography variant="h4" component="h1" gutterBottom align="center">
```

```
        Регистрация
      </Typography>

      {error && (
        <Alert severity="error" sx={{ mb: 2 }}>
          {error}
        </Alert>
      )}

      <form onSubmit={formik.handleSubmit}>
        <TextField
          fullWidth
          id="username"
          name="username"
          label="Имя пользователя"
          value={formik.values.username}
          onChange={formik.handleChange}
          error={formik.touched.username && Boolean(formik.errors.username)}
          helperText={formik.touched.username && formik.errors.username}
          margin="normal"
        />

        <TextField
          fullWidth
          id="email"
          name="email"
          label="Email"
          type="email"
          value={formik.values.email}
          onChange={formik.handleChange}
          error={formik.touched.email && Boolean(formik.errors.email)}
          helperText={formik.touched.email && formik.errors.email}
          margin="normal"
        />

        <TextField
          fullWidth
          id="password"
          name="password"
          label="Пароль"
          type="password"
          value={formik.values.password}
          onChange={formik.handleChange}
          error={formik.touched.password && Boolean(formik.errors.password)}
          helperText={formik.touched.password && formik.errors.password}
          margin="normal"
        />

        <TextField
          fullWidth
```

```
            id="password2"
            name="password2"
            label="Подтверждение пароля"
            type="password"
            value={formik.values.password2}
            onChange={formik.handleChange}
            error={formik.touched.password2 && Boolean(formik.errors.password2)}
            helperText={formik.touched.password2 && formik.errors.password2}
            margin="normal"
          />

          <Button
            type="submit"
            fullWidth
            variant="contained"
            color="primary"
            disabled={formik.isSubmitting}
            sx={{ mt: 3, mb: 2 }}
          >
            Зарегистрироваться
          </Button>

          <Box textAlign="center">
            <Link to="/login">
              <Typography variant="body2">
                Уже есть аккаунт? Войти
              </Typography>
            </Link>
          </Box>
        </form>
      </Paper>
    </Container>
  );
};

export default RegisterForm;
```

Шаг 6. Создание компонента входа

frontend/src/components/LoginForm.js:

```
import React, { useState } from 'react';
import { useFormik } from 'formik';
import * as Yup from 'yup';
import { useAuth } from '../context/AuthContext';
import { useNavigate, Link } from 'react-router-dom';
import {
  TextField,
  Button,
  Container,
```

```
  Typography,
  Box,
  Alert,
  Paper
} from '@mui/material';

const validationSchema = Yup.object({
  username: Yup.string().required('Имя пользователя обязательно'),
  password: Yup.string().required('Пароль обязателен')
});

const LoginForm = () => {
  const { login } = useAuth();
  const navigate = useNavigate();
  const [error, setError] = useState('');

  const formik = useFormik({
    initialValues: {
      username: '',
      password: ''
    },
    validationSchema: validationSchema,
    onSubmit: async (values) => {
      const result = await login(values.username, values.password);
      if (result.success) {
        navigate('/tasks');
      } else {
        setError(result.error);
      }
    }
  });

  return (
    <Container maxWidth="sm">
      <Paper elevation={3} sx={{ p: 4, mt: 8 }}>
        <Typography variant="h4" component="h1" gutterBottom align="center">
          Вход
        </Typography>

        {error && (
          <Alert severity="error" sx={{ mb: 2 }}>
            {error}
          </Alert>
        )}

        <form onSubmit={formik.handleSubmit}>
          <TextField
            fullWidth
            id="username"
            name="username"
```

```
            label="Имя пользователя"
            value={formik.values.username}
            onChange={formik.handleChange}
            error={formik.touched.username && Boolean(formik.errors.username)}
            helperText={formik.touched.username && formik.errors.username}
            margin="normal"
          />

          <TextField
            fullWidth
            id="password"
            name="password"
            label="Пароль"
            type="password"
            value={formik.values.password}
            onChange={formik.handleChange}
            error={formik.touched.password && Boolean(formik.errors.password)}
            helperText={formik.touched.password && formik.errors.password}
            margin="normal"
          />

          <Button
            type="submit"
            fullWidth
            variant="contained"
            color="primary"
            disabled={formik.isSubmitting}
            sx={{ mt: 3, mb: 2 }}
          >
            Войти
          </Button>

          <Box textAlign="center">
            <Link to="/register">
              <Typography variant="body2">
                Нет аккаунта? Зарегистрироваться
              </Typography>
            </Link>
          </Box>
        </form>
      </Paper>
    </Container>
  );
};

export default LoginForm;
```

Шаг 7. Создание защищенных маршрутов

frontend/src/components/PrivateRoute.js:

```
import { Navigate } from 'react-router-dom';
import { useAuth } from '../context/AuthContext';

const PrivateRoute = ({ children }) => {
  const { isAuthenticated, loading } = useAuth();

  if (loading) {
    return <div>Загрузка...</div>;
  }

  return isAuthenticated ? children : <Navigate to="/login" />;
};

export default PrivateRoute;
```

frontend/src/components/PublicRoute.js:

```
import { Navigate } from 'react-router-dom';
import { useAuth } from '../context/AuthContext';

const PublicRoute = ({ children }) => {
  const { isAuthenticated, loading } = useAuth();

  if (loading) {
    return <div>Загрузка...</div>;
  }

  return !isAuthenticated ? children : <Navigate to="/tasks" />;
};

export default PublicRoute;
```

Шаг 8. Настройка маршрутизации

frontend/src/App.js:

```
import React from 'react';
import { BrowserRouter, Routes, Route } from 'react-router-dom';
import { AuthProvider } from './context/AuthContext';
import PrivateRoute from './components/PrivateRoute';
import PublicRoute from './components/PublicRoute';
import LoginForm from './components/LoginForm';
import RegisterForm from './components/RegisterForm';
import TaskList from './components/TaskList';
import Navbar from './components/Navbar';

function App() {
  return (
```

```
      <BrowserRouter>
        <AuthProvider>
          <Navbar />
          <Routes>
            <Route path="/login" element={
              <PublicRoute>
                <LoginForm />
              </PublicRoute>
            } />
            <Route path="/register" element={
              <PublicRoute>
                <RegisterForm />
              </PublicRoute>
            } />
            <Route path="/tasks" element={
              <PrivateRoute>
                <TaskList />
              </PrivateRoute>
            } />
            <Route path="/" element={
              <PrivateRoute>
                <TaskList />
              </PrivateRoute>
            } />
          </Routes>
        </AuthProvider>
      </BrowserRouter>
    );
}

export default App;
```

Шаг 9. Компонент навигации

frontend/src/components/Navbar.js:

```
import React from 'react';
import { AppBar, Toolbar, Typography, Button, Box } from '@mui/material';
import { Link, useNavigate } from 'react-router-dom';
import { useAuth } from '../context/AuthContext';

const Navbar = () => {
  const { isAuthenticated, user, logout } = useAuth();
  const navigate = useNavigate();

  const handleLogout = () => {
    logout();
    navigate('/login');
  };
```

```
  return (
    <AppBar position="static">
      <Toolbar>
        <Typography variant="h6" component="div" sx={{ flexGrow: 1 }}>
          <Link to="/" style={{ color: 'white', textDecoration: 'none' }}>
            Умный ежедневник
          </Link>
        </Typography>

        {isAuthenticated ?(
          <Box>
            <Typography variant="body1" component="span" sx={{ mr: 2 }}>
              {user?.username}
            </Typography>
            <Button color="inherit" onClick={handleLogout}>
              Выйти
            </Button>
          </Box>
        ) : (
          <Box>
            <Button color="inherit" component={Link} to="/login">
              Вход
            </Button>
            <Button color="inherit" component={Link} to="/register">
              Регистрация
            </Button>
          </Box>
        )}
      </Toolbar>
    </AppBar>
  );
};

export default Navbar;
```

Траектория Б (ASP.NET Core + React)

Для студентов, переходящих на ASP.NET в 4 семестре. Основные компоненты (React часть) будут идентичны траектории А, отличаются только бэкенд-эндпоинты и обработка JWT.

Шаг 1. Настройка бэкенда для JWT (проверка)

Убедитесь, что в ASP.NET Core приложении настроены эндпоинты:

```
// Controllers/AuthController.cs
[ApiController]
[Route("api/[controller]")]
public class AuthController : ControllerBase
{
    private readonly UserManager<User> _userManager;
    private readonly SignInManager<User> _signInManager;
    private readonly IConfiguration _configuration;

    [HttpPost("login")]
    public async Task<IActionResult> Login(LoginModel model)
    {
        var user = await _userManager.FindByNameAsync(model.Username);
        if (user != null && await _userManager.CheckPasswordAsync(user, model.Password))
        {
            var token = await GenerateJwtToken(user);
            return Ok(new { token });
        }
        return Unauthorized();
    }
}
```

Шаг 2. Адаптация authService.js для ASP.NET

```
// frontend/src/services/authService.js (для ASP.NET)
const authService = {
    async login(username, password) {
        const response = await axios.post( `${API_URL}/auth/login `, {
            username,
            password
        });

        if (response.data.token) {
            localStorage.setItem('access_token', response.data.token);
        }

        return response.data;
    },

    // Остальные методы аналогичны траектории A
    // ...
};
```

Остальные компоненты (RegisterForm, LoginForm, AuthContext, PrivateRoute) остаются практически идентичными, меняется только логика вызовов API.

Контрольные вопросы

1. Какие существуют стратегии хранения JWT-токенов на клиенте? В чём их преимущества и недостатки?
2. Как работает механизм refresh-токена?
3. Для чего нужны axios interceptors?
4. Как защитить маршруты в React Router?
5. В чём разница между авторизацией на бэкенде и на фронтенде?
6. Что такое XSS-атака и как она связана с хранением токенов?
7. Как работает автоматическое обновление access-токена при его истечении?

Критерии оценки

Критерий	Баллы
Реализованы формы регистрации и входа (Formik + Yup)	2
Настроено хранение токенов (localStorage)	1
Реализованы axios interceptors для подстановки токена	2
Реализована защита маршрутов (PrivateRoute)	2
Реализовано автоматическое обновление access-токена	2
Реализован выход из системы	1
Итого	10

Дополнительные задания (для продвинутых)

1. HttpOnly cookies: Реализовать хранение refresh-токена в HttpOnly cookie для повышения безопасности
2. Биометрия: Добавить возможность сохранения сессии (Remember me)

3. Двухфакторная аутентификация: Реализовать 2FA на бэкенде и фронтенде
4. Обработка ошибок: Расширенная обработка ошибок (сеть, сервер, неверные данные)
5. Индикация загрузки: Добавить спиннеры при выполнении асинхронных операций

Результат выполнения

После выполнения лабораторной работы №4 должны получить:

- Полноценную систему аутентификации в SPA:
  - Регистрация новых пользователей
  - Вход существующих пользователей
  - Выход из системы
- Защищённые маршруты:
  - Неавторизованные пользователи перенаправляются на страницу входа
  - Авторизованные пользователи имеют доступ к защищённым страницам
- Автоматическое управление токенами:
  - Access token автоматически добавляется к запросам
  - При истечении access-токена автоматически запрашивается новый через refresh-токен
- Пользовательский интерфейс:
  - Формы с валидацией и отображением ошибок
  - Навигационная панель с именем пользователя и кнопкой выхода

Лабораторная работа №5

Документирование API и тестирование бэкенда

Цель работы: Автоматическое документирование API и написание модульных и интеграционных тестов для бэкенда.

Задачи

№	Задача
1	Настроить автоматическую генерацию документации API (Swagger/OpenAPI)
2	Написать модульные тесты для сервисного слоя
3	Написать интеграционные тесты для API эндпоинтов
4	Настроить покрытие кода тестами (минимальный порог 70%)
5	Реализовать валидацию входных данных и обработку ошибок

Теоретическая часть

Документирование API (OpenAPI/Swagger)

OpenAPI (ранее Swagger) - спецификация для описания REST API в формате JSON или YAML.

Swagger UI - инструмент, который визуализирует OpenAPI спецификацию в интерактивную документацию, позволяющую тестировать API прямо из браузера.

Преимущества автоматической документации:

1. Документация всегда актуальна (генерируется из кода)
2. Интерактивное тестирование API
3. Стандартизированный формат

Пирамида тестирования

Тип тестов	Что тестируют	Количество	Скорость
Модульные (Unit)	Отдельные классы/функции	Много	Быстрые
Интеграционные	Взаимодействие компонентов	Средне	Средние
E2E	Полные пользовательские сценарии	Мало	Медленные

Покрытие кода (Code Coverage)

Метрика, показывающая, какой процент кода выполняется во время тестов.

Рекомендуемые пороги:

1. 80% и выше - отлично
2. 70-80% - хорошо
3. Менее 70% - требует улучшения

Ход выполнения работы

Общие требования

1. API должно быть полностью документировано через Swagger/OpenAPI

2. Модульные тесты должны покрывать не менее 70% сервисного слоя
3. Интеграционные тесты должны проверять все CRUD-операции
4. Тесты должны запускаться автоматически при сборке проекта

Траектория A (Django + React)

Шаг 1. Установка необходимых пакетов

```
pip install drf-spectacular
pip install pytest pytest-django pytest-cov factory-boy
```

Шаг 2. Настройка документации API

settings.py:

```
INSTALLED_APPS = [
    # ...
    'drf_spectacular',
]

REST_FRAMEWORK = {
    'DEFAULT_SCHEMA_CLASS': 'drf_spectacular.openapi.AutoSchema',
    'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.PageNumberPagination',
    'PAGE_SIZE': 10,
    'DEFAULT_FILTER_BACKENDS': [
        'django_filters.rest_framework.DjangoFilterBackend',
        'rest_framework.filters.SearchFilter',
        'rest_framework.filters.OrderingFilter',
    ],
}

SPECTACULAR_SETTINGS = {
    'TITLE': 'Smart Diary API',
```

```
    'DESCRIPTION': 'API для приложения "Умный ежедневник" — система управления задачами и проектами',
    'VERSION': '1.0.0',
    'SERVE_INCLUDE_SCHEMA': False,
    'COMPONENT_SPLIT_REQUEST': True,
    'SWAGGER_UI_SETTINGS': {
        'persistAuthorization': True,
    },
}
```

urls.py:

```
from django urls import path, include
from drf_spectacular views import SpectacularAPIView, SpectacularSwaggerView, SpectacularRedocView

urlpatterns = [
    # Swagger документация
    path('api/schema/', SpectacularAPIView as_view(), name='schema'),
    path('api/schema/swagger-ui/', SpectacularSwaggerView as_view(url_name='schema'), name='swagger-ui'),
    path('api/schema/redoc/', SpectacularRedocView as_view(url_name='schema'), name='redoc'),

    # API маршруты
    path('api/', include('tasks.urls')),
]
```

tasks/views.py (обновленная версия с аннотациями):

```
from drf_spectacular utils import extend_schema, extend_schema_view, OpenApiParameter, OpenApiResponse
from drf_spectacular types import OpenApiTypes
from rest_framework import viewsets, status
```

```
from rest_framework.decorators import action
from rest_framework.response import Response
from .models import Task, Project, Tag
from .serializers import TaskSerializer, ProjectSerializer, TagSerializer
from .services import TaskService, ProjectService, TagService

TEST_USER_ID = 1


@extend_schema_view(
    list=extend_schema(
        summary="Получить список задач",
        description="Возвращает список задач пользователя с фильтрацией и пагинацией",
        parameters=[
            OpenApiParameter(name='page', type=OpenApiTypes.INT, description='Номер страницы', default=1),
            OpenApiParameter(name='page_size', type=OpenApiTypes.INT, description='Размер страницы', default=10),
            OpenApiParameter(name='status', type=OpenApiTypes.STR, description='Статус задачи (New, InProgress, Completed)'),
            OpenApiParameter(name='priority', type=OpenApiTypes.STR, description='Приоритет (Low, Medium, High)'),
            OpenApiParameter(name='project_id', type=OpenApiTypes.INT, description='ID проекта'),
            OpenApiParameter(name='search', type=OpenApiTypes.STR, description='Поиск по заголовку и описанию'),
        ],
        responses={
            200: OpenApiResponse(response=TaskSerializer, description='Успешный ответ'),
        }
    ),
    retrieve=extend_schema(
        summary="Получить задачу по ID",
        description="Возвращает полную информацию о задаче",
        responses={
```

```
            200: OpenApiResponse(response=TaskSerializer, description='Успешный ответ'),
            404: OpenApiResponse(description='Задача не найдена'),
        }
    ),
    create=extend_schema(
        summary="Создать задачу",
        description="Создает новую задачу для текущего пользователя",
        request=TaskSerializer,
        responses={
            201: OpenApiResponse(response=TaskSerializer, description='Задача создана'),
            400: OpenApiResponse(description='Ошибка валидации'),
        }
    ),
    update=extend_schema(
        summary="Обновить задачу",
        description="Полностью обновляет существующую задачу",
        request=TaskSerializer,
        responses={
            200: OpenApiResponse(response=TaskSerializer, description='Задача обновлена'),
            400: OpenApiResponse(description='Ошибка валидации'),
            404: OpenApiResponse(description='Задача не найдена'),
        }
    ),
    destroy=extend_schema(
        summary="Удалить задачу",
        description="Удаляет задачу по ID",
        responses={
            204: OpenApiResponse(description='Задача удалена'),
            404: OpenApiResponse(description='Задача не найдена'),
        }
    )
)
```

```
class TaskViewSet(viewsets.ViewSet):
    # ... код из ЛР №3 (без изменений)
```

Шаг 4. Настройка pytest

pytest.ini (корень проекта):

```
[pytest]
DJANGO_SETTINGS_MODULE = smart_diary.settings
python_files = test_*.py
addopts = --cov=tasks --cov-report=html --cov-report=term --cov-fail-under=70
```

Шаг 5. Написание модульных тестов для сервисного слоя

tasks/tests/test_services.py:

```
import pytest
from django.utils import timezone
from datetime import timedelta
from tasks.models import User, Project, Tag, Task
from tasks.services import TaskService, ProjectService, TagService


@pytest.mark.django_db
class TestTaskService:
    """Модульные тесты для TaskService"""

    @pytest.fixture
    def user(self):
        return User.objects.create_user(
            username='testuser',
            email='test@example.com',
            password='testpass123'
        )

    @pytest.fixture
    def project(self, user):
```

```
    return Project.objects.create(
        name='Тестовый проект',
        description='Описание проекта',
        color='#FF5733',
        owner=user
    )

@pytest.fixture
def tags(self, user):
    return [
        Tag.objects.create(name='Важное', owner=user),
        Tag.objects.create(name='Срочное', owner=user)
    ]

def test_create_task(self, user, project, tags):
    """Тест создания задачи"""
    data = {
        'title': 'Новая задача',
        'description': 'Описание задачи',
        'status': 'New',
        'priority': 'High',
        'project_id': project.id,
        'tag_ids': [tags[0].id, tags[1].id]
    }

    task = TaskService.create_task(data, user.id)

    assert task is not None
    assert task.title == 'Новая задача'
    assert task.user_id == user.id
    assert task.project_id == project.id
    assert task.tags.count() == 2
```

```python
def test_get_all_tasks_with_filters(self, user, project):
    """Тест получения задач с фильтрацией"""
    # Создаем тестовые задачи
    Task.objects.create(
        title='Задача 1',
        status='New',
        priority='High',
        user_id=user.id,
        project=project
    )
    Task.objects.create(
        title='Задача 2',
        status='Completed',
        priority='Low',
        user_id=user.id,
        project=project
    )

    # Фильтрация по статусу
    result = TaskService.get_all_tasks(
        user.id,
        filters={'status': 'New'},
        page=1,
        page_size=10
    )

    assert result['total'] == 1
    assert result['tasks'][0]['title'] == 'Задача 1'

def test_get_all_tasks_with_search(self, user):
    """Тест поиска задач"""
```

```
    Task.objects.create(
        title='Купить продукты',
        description='Молоко, хлеб',
        status='New',
        priority='Medium',
        user_id=user.id
    )
    Task.objects.create(
        title='Сдать отчет',
        description='Подготовить отчет',
        status='New',
        priority='High',
        user_id=user.id
    )

    result = TaskService.get_all_tasks(
        user.id,
        filters={'search': 'продукты'},
        page=1,
        page_size=10
    )

    assert result['total'] == 1
    assert 'продукты' in result['tasks'][0]['title'].lower()

def test_get_task_by_id_not_found(self, user):
    """Тест получения несуществующей задачи"""
    task = TaskService.get_task_by_id(999, user.id)
    assert task is None

def test_update_task(self, user, project):
    """Тест обновления задачи"""
```

```
    task = Task.objects.create(
        title='Старый заголовок',
        description='Старое описание',
        status='New',
        priority='Low',
        user_id=user.id,
        project=project
    )

    updated_task = TaskService.update_task(
        task.id,
        user.id,
        {
            'title': 'Новый заголовок',
            'priority': 'High'
        }
    )

    assert updated_task is not None
    assert updated_task.title == 'Новый заголовок'
    assert updated_task.priority == 'High'

def test_delete_task(self, user):
    """Тест удаления задачи"""
    task = Task.objects.create(
        title='Задача для удаления',
        status='New',
        priority='Medium',
        user_id=user.id
    )

    result = TaskService.delete_task(task.id, user.id)
```

```
        assert result is True
        assert Task objects.filter(id=task.id).exists() is False

    def test_delete_task_not_found(self, user):
        """Тест удаления несуществующей задачи"""
        result = TaskService.delete_task(999, user.id)
        assert result is False


@pytest.mark.django_db
class TestProjectService:
    """Модульные тесты для ProjectService"""

    @pytest.fixture
    def user(self):
        return User.objects.create_user(
            username='testuser',
            email='test@example.com',
            password='testpass123'
        )

    def test_create_project(self, user):
        """Тест создания проекта"""
        data = {
            'name': 'Новый проект',
            'description': 'Описание проекта',
            'color': '#FF5733'
        }

        project = ProjectService.create_project(data, user.id)
```

```
    assert project.name == 'Новый проект'
    assert project.owner_id == user.id

def test_get_all_projects(self, user):
    """Тест получения всех проектов"""
    Project.objects.create(name='Проект 1', owner=user)
    Project.objects.create(name='Проект 2', owner=user)

    projects = ProjectService.get_all_projects(user.id)

    assert projects.count() == 2

def test_update_project(self, user):
    """Тест обновления проекта"""
    project = Project.objects.create(name='Старое имя', owner=user)

    updated = ProjectService.update_project(
        project.id,
        user.id,
        {'name': 'Новое имя'}
    )

    assert updated.name == 'Новое имя'

def test_delete_project(self, user):
    """Тест удаления проекта"""
    project = Project.objects.create(name='Проект для удаления', owner=user)

    result = ProjectService.delete_project(project.id, user.id)

    assert result is True
    assert Project.objects.filter(id=project.id).exists() is False
```

Шаг 6. Написание интеграционных тестов для API

tasks/tests/test_api.py:

```
import pytest
from django urls import reverse
from rest_framework test import APIClient
from rest_framework import status
from tasks models import User, Project, Tag, Task


@pytest.mark.django_db
class TestTaskAPI:
    """Интеграционные тесты для Task API"""

    @pytest.fixture
    def api_client(self):
        return APIClient()

    @pytest.fixture
    def user(self):
        return User.objects.create_user(
            username='testuser',
            email='test@example.com',
            password='testpass123'
        )

    @pytest.fixture
    def project(self, user):
        return Project.objects.create(
            name='Тестовый проект',
            owner=user
        )
```

```python
@pytest.fixture
def auth_client(self, api_client, user):
    # Временно используем force_authenticate (пока без JWT)
    api_client.force_authenticate(user=user)
    return api_client

def test_get_tasks_list(self, auth_client, user):
    """Тест получения списка задач"""
    Task.objects.create(
        title='Задача 1',
        status='New',
        priority='Medium',
        user_id=user.id
    )
    Task.objects.create(
        title='Задача 2',
        status='Completed',
        priority='High',
        user_id=user.id
    )

    response = auth_client.get('/api/tasks/')

    assert response.status_code == status.HTTP_200_OK
    assert response.data['total'] == 2
    assert len(response.data['tasks']) == 2

def test_create_task(self, auth_client, project):
    """Тест создания задачи"""
    data = {
        'title': 'Новая задача через API',
```

```
        'description': 'Описание',
        'status': 'New',
        'priority': 'High',
        'project_id': project.id
    }

    response = auth_client.post('/api/tasks/', data, format='json')

    assert response.status_code == status.HTTP_201_CREATED
    assert response.data['title'] == 'Новая задача через API'

def test_create_task_invalid_data(self, auth_client):
    """Тест создания задачи с некорректными данными"""
    data = {
        'title': '',  # Пустой заголовок
        'status': 'New',
        'priority': 'Medium'
    }

    response = auth_client.post('/api/tasks/', data, format='json')

    assert response.status_code == status.HTTP_400_BAD_REQUEST
    assert 'title' in response.data

def test_get_task_detail(self, auth_client, user):
    """Тест получения деталей задачи"""
    task = Task.objects.create(
        title='Детальная задача',
        description='Подробное описание',
        status='New',
        priority='High',
        user_id=user.id
```

```
    )

    response = auth_client.get(f'/api/tasks/{task.id}/')

    assert response.status_code == status.HTTP_200_OK
    assert response.data['title'] == 'Детальная задача'
    assert response.data['description'] == 'Подробное описание'

def test_get_task_detail_not_found(self, auth_client):
    """Тест получения несуществующей задачи"""
    response = auth_client.get('/api/tasks/999/')

    assert response.status_code == status.HTTP_404_NOT_FOUND

def test_update_task(self, auth_client, user):
    """Тест обновления задачи"""
    task = Task.objects.create(
        title='Старый заголовок',
        description='Старое описание',
        status='New',
        priority='Low',
        user_id=user.id
    )

    data = {
        'title': 'Новый заголовок',
        'description': 'Новое описание',
        'status': 'InProgress',
        'priority': 'High'
    }

    response = auth_client.put(f'/api/tasks/{task.id}/', data, format='json')
```

```
    assert response.status_code == status.HTTP_200_OK
    assert response.data['title'] == 'Новый заголовок'
    assert response.data['status'] == 'InProgress'

def test_delete_task(self, auth_client, user):
    """Тест удаления задачи"""
    task = Task.objects.create(
        title='Задача для удаления',
        status='New',
        priority='Medium',
        user_id=user.id
    )

    response = auth_client.delete(f'/api/tasks/{task.id}/')

    assert response.status_code == status.HTTP_204_NO_CONTENT
    assert Task.objects.filter(id=task.id).exists() is False

def test_filter_tasks_by_status(self, auth_client, user):
    """Тест фильтрации задач по статусу"""
    Task.objects.create(
        title='Новая задача',
        status='New',
        priority='Medium',
        user_id=user.id
    )
    Task.objects.create(
        title='Выполненная задача',
        status='Completed',
        priority='Low',
        user_id=user.id
```

```
    )

    response = auth_client.get('/api/tasks/?status=New')

    assert response.status_code == status.HTTP_200_OK
    assert response.data['total'] == 1
    assert response.data['tasks'][0]['title'] == 'Новая задача'

def test_search_tasks(self, auth_client, user):
    """Тест поиска задач"""
    Task.objects.create(
        title='Купить продукты',
        description='Молоко и хлеб',
        status='New',
        priority='Medium',
        user_id=user.id
    )
    Task.objects.create(
        title='Сделать уборку',
        description='Убраться в комнате',
        status='New',
        priority='Low',
        user_id=user.id
    )

    response = auth_client.get('/api/tasks/?search=продукты')

    assert response.status_code == status.HTTP_200_OK
    assert response.data['total'] == 1


@pytest.mark.django_db
```

```python
class TestProjectAPI:
    """Интеграционные тесты для Project API"""

    @pytest.fixture
    def api_client(self):
        return APIClient()

    @pytest.fixture
    def user(self):
        return User.objects.create_user(
            username='testuser',
            email='test@example.com',
            password='testpass123'
        )

    @pytest.fixture
    def auth_client(self, api_client, user):
        api_client.force_authenticate(user=user)
        return api_client

    def test_create_project(self, auth_client):
        """Тест создания проекта"""
        data = {
            'name': 'Новый проект',
            'description': 'Описание проекта',
            'color': '#FF5733'
        }

        response = auth_client.post('/api/projects/', data, format='json')

        assert response.status_code == status.HTTP_201_CREATED
        assert response.data['name'] == 'Новый проект'
```

```
    def test_get_projects_list(self, auth_client, user):
        """Тест получения списка проектов"""
        Project.objects.create(name='Проект 1', owner=user)
        Project.objects.create(name='Проект 2', owner=user)

        response = auth_client.get('/api/projects/')

        assert response.status_code == status.HTTP_200_OK
        assert len(response.data) == 2
```

Шаг 7. Запуск тестов и проверка покрытия

```
# Запуск всех тестов с отчетом о покрытии
pytest

# Запуск с подробным выводом
pytest -v

# Запуск конкретного тестового файла
pytest tasks/tests/test_services.py -v

# Запуск с генерацией HTML отчета
pytest --cov=tasks --cov-report=html
```

Шаг 8. Просмотр документации

После запуска сервера:

1. Swagger UI: http://localhost:8000/api/schema/swagger-ui/
2. ReDoc: http://localhost:8000/api/schema/redoc/
3. OpenAPI schema (JSON): http://localhost:8000/api/schema/

Траектория Б (ASP.NET Core + React)

Шаг 1. Настройка Swagger/OpenAPI

Program.cs:

```
using Microsoft OpenApi Models;

var builder = WebApplication CreateBuilder(args);

builder Services AddControllers();
builder Services AddEndpointsApiExplorer();

// Настройка Swagger
builder Services AddSwaggerGen(c =>
{
    c SwaggerDoc("v1", new OpenApiInfo
    {
        Title = "Smart Diary API",
        Version = "v1",
        Description = "API для приложения \"Умный ежедневник\" — система управления задачами и проектами",
        Contact = new OpenApiContact
        {
            Name = "Smart Diary Team",
            Email = "support@smartdiary.com"
        }
    });

    // Добавление поддержки JWT (будет использовано позже)
    c AddSecurityDefinition("Bearer", new OpenApiSecurityScheme
    {
        Description = "JWT Authorization header using the Bearer scheme.",
        Name = "Authorization",
        In = ParameterLocation Header,
        Type = SecuritySchemeType Http,
```

```
        Scheme = "bearer"
    });

    c.AddSecurityRequirement(new OpenApiSecurityRequirement
    {
        {
            new OpenApiSecurityScheme
            {
                Reference = new OpenApiReference
                {
                    Type = ReferenceType.SecurityScheme,
                    Id = "Bearer"
                }
            },
            Array.Empty<string>()
        }
    });
});

var app = builder.Build();

// Настройка Swagger UI
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI(c =>
    {
        c.SwaggerEndpoint("/swagger/v1/swagger.json", "Smart Diary API v1");
        c.RoutePrefix = "swagger";
    });
}
```

```
app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();


app.Run();
```

Шаг 2. Аннотирование контроллеров

Controllers/TaskController.cs:

```
using Microsoft.AspNetCore.Mvc;
using Swashbuckle.AspNetCore.Annotations;

[ApiController]
[Route("api/[controller]")]
[SwaggerTag("Управление задачами")]
public class TaskController : ControllerBase
{
    private readonly ITaskService _taskService;

    public TaskController(ITaskService taskService)
    {
        _taskService = taskService;
    }

    /// <summary>
    /// Получить список задач пользователя
    /// </summary>
    /// <remarks>
    /// Возвращает список задач с поддержкой пагинации, фильтрации и поиска.
    ///
    /// Пример запроса:
    ///
    ///     GET /api/task?page=1&amp;pageSize=10&amp;status=New&amp;search=продукты
    ///
```

```
    /// </remarks>
    /// <param name="page">Номер страницы (по умолчанию 1)</param>
    /// <param name="pageSize">Размер страницы (по умолчанию 10)</param>
    /// <param name="status">Фильтр по статусу (New, InProgress, Completed)</param>
    /// <param name="priority">Фильтр по приоритету (Low, Medium, High)</param>
    /// <param name="projectId">Фильтр по проекту</param>
    /// <param name="search">Поиск по заголовку и описанию</param>
    /// <returns>Список задач с метаинформацией пагинации</returns>
    [HttpGet]
    [SwaggerOperation(Summary = "Получить список задач", Description = "Возвращает список задач пользователя с фильтрацией и пагинацией")]
    [SwaggerResponse(200, "Успешный ответ", typeof(PagedResult<TaskDto>))]
    [SwaggerResponse(400, "Некорректные параметры запроса")]
    public async Task<ActionResult<PagedResult<TaskDto>>> GetTasks(
        [FromQuery] int page = 1,
        [FromQuery] int pageSize = 10,
        [FromQuery] string? status = null,
        [FromQuery] string? priority = null,
        [FromQuery] int? projectId = null,
        [FromQuery] string? search = null)
    {
        int userId = 1; // Временный ID (пока без аутентификации)

        var filter = new TaskFilterDto
        {
            Status = status,
            Priority = priority,
            ProjectId = projectId,
            Search = search
        };

        var result = await _taskService.GetUserTasksAsync(userId, filter, page, pageSize);
```

```
        return Ok(result);
    }

    /// <summary>
    /// Получить задачу по ID
    /// </summary>
    /// <param name="id">ID задачи</param>
    /// <returns>Детальная информация о задаче</returns>
    [HttpGet("{id}")]
    [SwaggerOperation(Summary = "Получить задачу по ID", Description = "Возвращает полную информацию о задаче")]
    [SwaggerResponse(200, "Успешный ответ", typeof(TaskDto))]
    [SwaggerResponse(404, "Задача не найдена")]
    public async Task<ActionResult<TaskDto>> GetTask(int id)
    {
        int userId = 1;
        var task = await _taskService.GetTaskByIdAsync(id, userId);

        if (task == null)
            return NotFound(new { message = $"Задача с ID {id} не найдена" });

        return Ok(task);
    }

    /// <summary>
    /// Создать новую задачу
    /// </summary>
    /// <param name="dto">Данные для создания задачи</param>
    /// <returns>Созданная задача</returns>
    [HttpPost]
    [SwaggerOperation(Summary = "Создать задачу", Description = "Создает новую задачу для текущего пользователя")]
    [SwaggerResponse(201, "Задача создана", typeof(TaskDto))]
```

```
    [SwaggerResponse(400, "Ошибка валидации")]
    public async Task<ActionResult<TaskDto>> CreateTask(CreateTaskDto dto)
    {
        if (!ModelState.IsValid)
            return BadRequest(ModelState);

        int userId = 1;
        var task = await _taskService.CreateTaskAsync(dto, userId);
        return CreatedAtAction(nameof(GetTask), new { id = task.Id }, task);
    }

    /// <summary>
    /// Обновить задачу
    /// </summary>
    /// <param name="id">ID задачи</param>
    /// <param name="dto">Данные для обновления</param>
    /// <returns>Обновленная задача</returns>
    [HttpPut("{id}")]
    [SwaggerOperation(Summary = "Обновить задачу", Description = "Полностью обновляет существующую задачу")]
    [SwaggerResponse(200, "Задача обновлена", typeof(TaskDto))]
    [SwaggerResponse(400, "Ошибка валидации")]
    [SwaggerResponse(404, "Задача не найдена")]
    public async Task<ActionResult<TaskDto>> UpdateTask(int id, UpdateTaskDto dto)
    {
        if (!ModelState.IsValid)
            return BadRequest(ModelState);

        int userId = 1;
        var task = await _taskService.UpdateTaskAsync(id, dto, userId);

        if (task == null)
```

```
            return NotFound(new { message = $"Задача с ID {id} не найдена" });

        return Ok(task);
    }

    /// <summary>
    /// Удалить задачу
    /// </summary>
    /// <param name="id">ID задачи</param>
    /// <returns>Статус удаления</returns>
    [HttpDelete("{id}")]
    [SwaggerOperation(Summary = "Удалить задачу", Description = "Удаляет задачу по ID")]
    [SwaggerResponse(204, "Задача удалена")]
    [SwaggerResponse(404, "Задача не найдена")]
    public async Task<IActionResult> DeleteTask(int id)
    {
        int userId = 1;
        var success = await _taskService.DeleteTaskAsync(id, userId);

        if (!success)
            return NotFound(new { message = $"Задача с ID {id} не найдена" });

        return NoContent();
    }
}
```

Шаг 3. Написание модульных тестов (xUnit + Moq)

Tests/Services/TaskServiceTests.cs:

```
using Moq;
using Microsoft.EntityFrameworkCore;
using Xunit;

namespace SmartDiary.Tests.Services
```

```
{
    public class TaskServiceTests
    {
        private readonly Mock<ApplicationDbContext> _mockContext;
        private readonly TaskService _taskService;

        public TaskServiceTests()
        {
            _mockContext = new Mock<ApplicationDbContext>();
            _taskService = new TaskService(_mockContext.Object);
        }

        [Fact]
        public async Task GetTaskByIdAsync_ExistingTask_ReturnsTask()
        {
            // Arrange
            var userId = 1;
            var taskId = 1;
            var expectedTask = new Task
            {
                Id = taskId,
                Title = "Тестовая задача",
                UserId = userId
            };

            var mockSet = new Mock<DbSet<Task>>();
            mockSet.Setup(m => m.FindAsync(taskId)).ReturnsAsync(expectedTask);
            _mockContext.Setup(c => c.Tasks).Returns(mockSet.Object);

            // Act
            var result = await _taskService.GetTaskByIdAsync(taskId, userId);
```

```
    // Assert
    Assert.NotNull(result);
    Assert.Equal(expectedTask.Title, result.Title);
}

[Fact]
public async Task GetTaskByIdAsync_NonExistingTask_ReturnsNull()
{
    // Arrange
    var userId = 1;
    var taskId = 999;

    var mockSet = new Mock<DbSet<Task>>();
    mockSet.Setup(m => m.FindAsync(taskId)).ReturnsAsync((Task)null);
    _mockContext.Setup(c => c.Tasks).Returns(mockSet.Object);

    // Act
    var result = await _taskService.GetTaskByIdAsync(taskId, userId);

    // Assert
    Assert.Null(result);
}

[Fact]
public async Task CreateTaskAsync_ValidData_ReturnsCreatedTask()
{
    // Arrange
    var userId = 1;
    var dto = new CreateTaskDto
    {
        Title = "Новая задача",
        Description = "Описание",
```

```
        Status = "New",
        Priority = "High"
    };

    // Act
    var result = await _taskService.CreateTaskAsync(dto, userId);

    // Assert
    Assert.NotNull(result);
    Assert.Equal(dto.Title, result.Title);
}

[Fact]
public async Task DeleteTaskAsync_ExistingTask_ReturnsTrue()
{
    // Arrange
    var userId = 1;
    var taskId = 1;
    var taskToDelete = new Task { Id = taskId, UserId = userId };

    var mockSet = new Mock<DbSet<Task>>();
    mockSet.Setup(m => m.FindAsync(taskId)).ReturnsAsync(taskToDelete);
    _mockContext.Setup(c => c.Tasks).Returns(mockSet.Object);

    // Act
    var result = await _taskService.DeleteTaskAsync(taskId, userId);

    // Assert
    Assert.True(result);
}

[Fact]
```

```
    public async Task DeleteTaskAsync_NonExistingTask_ReturnsFalse()
    {
        // Arrange
        var userId = 1;
        var taskId = 999;

        var mockSet = new Mock<DbSet<Task>>();
        mockSet.Setup(m => m.FindAsync(taskId)).ReturnsAsync((Task)null);
        _mockContext.Setup(c => c.Tasks).Returns(mockSet.Object);

        // Act
        var result = await _taskService.DeleteTaskAsync(taskId, userId);

        // Assert
        Assert.False(result);
    }
  }
}
```

Шаг 4. Написание интеграционных тестов

Tests/Integration/TaskControllerTests.cs:

```
using Microsoft.AspNetCore.Mvc.Testing;
using System.Net.Http.Json;
using Xunit;

namespace SmartDiary.Tests.Integration
{
  public class TaskControllerTests : IClassFixture<WebApplicationFactory<Program>>
  {
    private readonly WebApplicationFactory<Program> _factory;
    private readonly HttpClient _client;
```

```
public TaskControllerTests(WebApplicationFactory<Program> factory)
{
    _factory = factory;
    _client = _factory.CreateClient();
}

[Fact]
public async Task GetTasks_ReturnsSuccessStatusCode()
{
    // Act
    var response = await _client.GetAsync("/api/task");

    // Assert
    response.EnsureSuccessStatusCode();
}

[Fact]
public async Task CreateTask_ValidData_ReturnsCreatedTask()
{
    // Arrange
    var newTask = new
    {
        title = "Интеграционный тест",
        description = "Создано через API",
        status = "New",
        priority = "High"
    };

    // Act
    var response = await _client.PostAsJsonAsync("/api/task", newTask);

    // Assert
```

```
    Assert.Equal(HttpStatusCode.Created, response.StatusCode);

    var createdTask = await response.Content.ReadFromJsonAsync<TaskDto>();
    Assert.NotNull(createdTask);
    Assert.Equal(newTask.title, createdTask.Title);
}

[Fact]
public async Task GetTaskById_NonExisting_ReturnsNotFound()
{
    // Act
    var response = await _client.GetAsync("/api/task/999");

    // Assert
    Assert.Equal(HttpStatusCode.NotFound, response.StatusCode);
}

[Fact]
public async Task UpdateTask_ValidData_ReturnsUpdatedTask()
{
    // Arrange
    var updateData = new
    {
        title = "Обновленная задача",
        description = "Обновленное описание",
        status = "InProgress",
        priority = "High"
    };

    // Сначала создаем задачу
    var createResponse = await _client.PostAsJsonAsync("/api/task", new
    {
```

```
            title = "Исходная задача",
            status = "New",
            priority = "Medium"
        });

        var createdTask = await createResponse.Content.ReadFromJsonAsync<TaskDto>();

        // Act
        var response = await _client.PutAsJsonAsync($"/api/task/{createdTask.Id}", updateData);

        // Assert
        Assert.Equal(HttpStatusCode.OK, response.StatusCode);

        var updatedTask = await response.Content.ReadFromJsonAsync<TaskDto>();
        Assert.Equal(updateData.title, updatedTask.Title);
    }
  }
}
```

Шаг 5. Настройка покрытия кода

SmartDiary.Tests.csproj:

```
<Project Sdk="Microsoft.NET.Sdk">
 <PropertyGroup>
  <TargetFramework>net8.0</TargetFramework>
  <ImplicitUsings>enable</ImplicitUsings>
  <Nullable>enable</Nullable>
  <IsPackable>false</IsPackable>
 </PropertyGroup>

 <ItemGroup>
  <PackageReference Include="coverlet.collector" Version="6.0.0" />
  <PackageReference Include="Microsoft.NET.Test.Sdk" Version="17.8.0" />
  <PackageReference Include="xunit" Version="2.5.3" />
```

```
    <PackageReference Include="xunit.runner.visualstudio" Version="2.5.3" />
    <PackageReference Include="Moq" Version="4.20.70" />
    <PackageReference Include="Microsoft.AspNetCore.Mvc.Testing" Version="8.0.0" />
  </ItemGroup>
</Project>
```

Шаг 6. Запуск тестов

```
# Запуск всех тестов
dotnet test

# Запуск с отчетом о покрытии
dotnet test --collect:"XPlat Code Coverage"

# Запуск с подробным выводом
dotnet test -v detailed
```

Контрольные вопросы

1. Что такое OpenAPI/Swagger и для чего он нужен?
2. Какие существуют типы тестов и чем они отличаются?
3. Что такое покрытие кода и какой порог рекомендуется?
4. В чем разница между модульными и интеграционными тестами?
5. Как автоматическая документация API помогает в разработке?
6. Что такое mock-объекты и для чего они используются?

Критерии оценки

Критерий	Баллы
Настроена автоматическая документация API (Swagger)	2
Написаны модульные тесты для сервисного слоя	2
Написаны интеграционные тесты для API	2
Покрытие кода не менее 70%	2

Критерий	Баллы
Настроена валидация и обработка ошибок	1
Код работает без ошибок	1
Итого	10

Дополнительные задания (для продвинутых)

1. Тестирование производительности: Написать нагрузочные тесты с использованием k6 или JMeter
2. Мутационное тестирование: Внедрить мутационное тестирование для оценки качества тестов (Stryker для .NET или Cosmic Ray для Python)
3. Тестирование безопасности: Добавить тесты для проверки защиты от SQL-инъекций
4. E2E тестирование: Добавить сквозные тесты с использованием Playwright или Cypress

Результат выполнения

1. После выполнения лабораторной работы №5 студенты должны получить:
2. Интерактивную документацию API (Swagger UI)
3. Набор модульных тестов для сервисного слоя (покрытие ≥70%)
4. Набор интеграционных тестов для API эндпоинтов
5. Отчет о покрытии кода
6. Обработку ошибок и валидацию входных данных

Лабораторная работа №6

Продвинутая бизнес-логика. Статистика и отчеты

Цель работы: Реализация сложных бизнес-сценариев, аналитических функций и генерации отчетов на основе накопленных данных.

Задачи

№	Задача
1	Реализовать подсчет статистики по задачам (по статусам, просроченные, продуктивность)
2	Реализовать экспорт данных в форматы CSV, JSON, PDF
3	Реализовать полнотекстовый поиск задач
4	Реализовать иерархию задач (подзадачи)
5	Реализовать систему напоминаний о дедлайнах

Теоретическая часть

Агрегация данных

Агрегация - процесс сбора и обобщения данных для получения сводной информации.

Типичные агрегатные функции:

1. COUNT - количество записей
2. SUM - сумма значений
3. AVG - среднее значение
4. MIN / MAX - минимальное/максимальное значение

Экспорт данных

Формат	**Описание**	**Применение**
CSV	Простой текстовый формат с	Импорт в Excel,

Формат	Описание	Применение
	разделителями	Google Sheets
JSON	Структурированный формат данных	Обмен данными между системами
PDF	Документ для печати	Отчеты, документы

Полнотекстовый поиск

Полнотекстовый поиск позволяет искать по текстовым полям с учетом словоформ, игнорированием стоп-слов и ранжированием результатов.

Иерархия задач

Подзадачи — механизм для детализации крупных задач:

- Задача может иметь несколько подзадач
- Подзадача всегда принадлежит одной родительской задаче
- Статус родительской задачи может зависеть от статусов подзадач

Ход выполнения работы

Общие требования

1. Аналитическая панель должна отображать ключевые метрики
2. Экспорт данных должен поддерживать фильтрацию по периоду
3. Полнотекстовый поиск должен работать по заголовку и описанию
4. Подзадачи должны поддерживать неограниченную вложенность

Траектория A (Django + React)

Шаг 1. Обновление модели Task (добавление parent для подзадач)

tasks/models.py (дополнение):

```python
class Task(models.Model):
    # ... существующие поля

    # Новое поле для иерархии задач
    parent = models.ForeignKey(
        'self',
        on_delete=models.CASCADE,
        null=True,
        blank=True,
        related_name='subtasks'
    )

    # Поле для времени напоминания
    reminder = models.DateTimeField(null=True, blank=True)

    # Поле для архивации (для статистики)
    completed_at = models.DateTimeField(null=True, blank=True)

    class Meta:
        indexes = [
            models.Index(fields=['user', 'status']),
            models.Index(fields=['user', 'deadline']),
            models.Index(fields=['user', 'created_at']),
        ]
```

Шаг 2. Создание миграции

python manage.py makemigrations

python manage.py migrate

Шаг 3. Реализация статистики

tasks/services/statistics_service.py:

```python
from django.db.models import Count, Q, Sum, Avg, F, Case, When, Value, IntegerField
from django.db.models.functions import TruncDate, TruncWeek, TruncMonth
from django.utils import timezone
from datetime import timedelta
from ..models import Task, Project

class StatisticsService:
    """Сервис для сбора статистики по задачам"""

    @staticmethod
    def get_task_status_stats(user_id):
        """Статистика по статусам задач"""
```

```
        total = Task.objects.filter(user_id=user_id).count()

        status_stats = Task.objects.filter(user_id=user_id).values('status').annotate(
            count=Count('id')
        )

        result = {item['status']: item['count'] for item in status_stats}

        # Добавляем проценты
        for status in ['New', 'InProgress', 'Completed']:
            result[f'{status}_percent'] = round((result.get(status, 0) / total * 100), 1) if total > 0 else 0

        return result

    @staticmethod
    def get_overdue_stats(user_id):
        """Статистика по просроченным задачам"""
        now = timezone.now()

        overdue = Task.objects.filter(
            user_id=user_id,
            deadline__lt=now,
            status__in=['New', 'InProgress']
        ).count()

        due_today = Task.objects.filter(
            user_id=user_id,
            deadline__date=now.date(),
            status__in=['New', 'InProgress']
        ).count()

        due_this_week = Task.objects.filter(
            user_id=user_id,
            deadline__date__lte=now.date() + timedelta(days=7),
            deadline__date__gte=now.date(),
            status__in=['New', 'InProgress']
        ).count()

        return {
            'overdue': overdue,
            'due_today': due_today,
            'due_this_week': due_this_week,
            'has_overdue': overdue > 0,
            'has_due_today': due_today > 0
        }

    @staticmethod
    def get_productivity_stats(user_id, days=30):
        """Продуктивность пользователя за период"""
        start_date = timezone.now() - timedelta(days=days)
```

```
    completed = Task.objects.filter(
        user_id=user_id,
        status='Completed',
        completed_at__gte=start_date
    ).count()

    created = Task.objects.filter(
        user_id=user_id,
        created_at__gte=start_date
    ).count()

    # Ежедневная статистика
    daily_stats = Task.objects.filter(
        user_id=user_id,
        created_at__gte=start_date
    ).annotate(
        date=TruncDate('created_at')
    ).values('date').annotate(
        created=Count('id', filter=Q(status__isnull=False)),
        completed=Count('id', filter=Q(status='Completed'))
    ).order_by('date')

    return {
        'period_days': days,
        'tasks_completed': completed,
        'tasks_created': created,
        'completion_rate': round((completed / created * 100), 1) if created > 0 else 0,
        'daily_stats': list(daily_stats)
    }

@staticmethod
def get_priority_distribution(user_id):
    """Распределение задач по приоритетам"""
    total = Task.objects.filter(user_id=user_id, status__in=['New', 'InProgress']).count()

    priorities = Task.objects.filter(
        user_id=user_id,
        status__in=['New', 'InProgress']
    ).values('priority').annotate(
        count=Count('id')
    )

    result = {}
    for item in priorities:
        result[item['priority']] = {
            'count': item['count'],
            'percent': round((item['count'] / total * 100), 1) if total > 0 else 0
        }
```

```python
        return result

    @staticmethod
    def get_project_stats(user_id):
        """Статистика по проектам"""
        projects = Project.objects.filter(owner_id=user_id).prefetch_related('tasks')

        result = []
        for project in projects:
            tasks = project.tasks.all()
            total = tasks.count()
            completed = tasks.filter(status='Completed').count()

            result.append({
                'id': project.id,
                'name': project.name,
                'color': project.color,
                'total_tasks': total,
                'completed_tasks': completed,
                'completion_rate': round((completed / total * 100), 1) if total > 0 else 0
            })

        return result

    @staticmethod
    def get_weekly_heatmap(user_id, weeks=8):
        """Тепловая карта активности по дням недели"""
        start_date = timezone.now() - timedelta(days=weeks * 7)

        # Создаем все дни периода
        days = []
        current = start_date
        while current <= timezone.now():
            days.append(current.date())
            current += timedelta(days=1)

        # Получаем выполненные задачи по дням
        completed_tasks = Task.objects.filter(
            user_id=user_id,
            status='Completed',
            completed_at__gte=start_date
        ).annotate(
            date=TruncDate('completed_at')
        ).values('date').annotate(
            count=Count('id')
        )

        completed_dict = {item['date']: item['count'] for item in completed_tasks}

        # Формируем результат
```

```
        heatmap = []
        for date in days:
            heatmap.append({
                'date': date.isoformat(),
                'day_of_week': date.weekday(),
                'week': date.isocalendar()[1],
                'count': completed_dict.get(date, 0)
            })

        return heatmap
```

Шаг 4. Реализация экспорта данных

tasks/services/export_service.py:

```
import csv
import json
import io
from datetime import datetime
from django.http import HttpResponse
from django.utils import timezone
from reportlab.lib import colors
from reportlab.lib.pagesizes import A4
from reportlab.lib.styles import getSampleStyleSheet, ParagraphStyle
from reportlab.lib.units import inch
from reportlab.platypus import SimpleDocTemplate, Table, TableStyle, Paragraph, Spacer
from ..models import Task, Project

class ExportService:
    """Сервис для экспорта данных в различные форматы"""

    @staticmethod
    def export_to_csv(user_id, format_type='tasks', filters=None):
        """Экспорт в CSV"""
        response = HttpResponse(content_type='text/csv')
        response['Content-Disposition'] = f'attachment;
filename="{format_type}_{datetime.now().strftime("%Y%m%d_%H%M%S")}.csv"'

        writer = csv.writer(response)

        if format_type == 'tasks':
            tasks = ExportService._get_filtered_tasks(user_id, filters)
            writer.writerow(['ID', 'Заголовок', 'Описание', 'Статус', 'Приоритет', 'Дедлайн',
'Создана', 'Проект'])
            for task in tasks:
                writer.writerow([
                    task.id,
```

```
                task.title,
                task.description or '',
                task.get_status_display(),
                task.get_priority_display(),
                task.deadline.strftime('%Y-%m-%d %H:%M') if task.deadline else '',
                task.created_at.strftime('%Y-%m-%d %H:%M'),
                task.project.name if task.project else ''
            ])

    elif format_type == 'projects':
        projects = Project.objects.filter(owner_id=user_id)
        writer.writerow(['ID', 'Название', 'Описание', 'Цвет', 'Создан', 'Кол-во задач'])
        for project in projects:
            writer.writerow([
                project.id,
                project.name,
                project.description or '',
                project.color,
                project.created_at.strftime('%Y-%m-%d'),
                project.tasks.count()
            ])

    elif format_type == 'statistics':
        stats = ExportService._get_statistics(user_id, filters)
        writer.writerow(['Метрика', 'Значение'])
        for key, value in stats.items():
            writer.writerow([key, value])

    return response

@staticmethod
def export_to_json(user_id, format_type='tasks', filters=None):
    """Экспорт в JSON"""
    if format_type == 'tasks':
        tasks = ExportService._get_filtered_tasks(user_id, filters)
        data = [
            {
                'id': task.id,
                'title': task.title,
                'description': task.description,
                'status': task.status,
                'status_display': task.get_status_display(),
                'priority': task.priority,
                'priority_display': task.get_priority_display(),
                'deadline': task.deadline.isoformat() if task.deadline else None,
                'created_at': task.created_at.isoformat(),
                'completed_at': task.completed_at.isoformat() if task.completed_at else None,
                'project': {
                    'id': task.project.id,
                    'name': task.project.name
```

```
                } if task.project else None,
                'tags': [{'id': tt.tag.id, 'name': tt.tag.name} for tt in task.task_tags.all()]
            }
            for task in tasks
        ]

    elif format_type == 'projects':
        projects = Project.objects.filter(owner_id=user_id)
        data = [
            {
                'id': project.id,
                'name': project.name,
                'description': project.description,
                'color': project.color,
                'created_at': project.created_at.isoformat(),
                'tasks_count': project.tasks.count()
            }
            for project in projects
        ]

    else:
        data = ExportService._get_statistics(user_id, filters)

    response = HttpResponse(
        json.dumps(data, indent=2, ensure_ascii=False),
        content_type='application/json'
    )
    response['Content-Disposition'] = f'attachment;
filename="{format_type}_{datetime.now().strftime("%Y%m%d_%H%M%S")}.json"'
    return response

@staticmethod
def export_to_pdf(user_id, period_days=30):
    """Экспорт отчета в PDF"""
    response = HttpResponse(content_type='application/pdf')
    response['Content-Disposition'] = f'attachment;
filename="report_{datetime.now().strftime("%Y%m%d_%H%M%S")}.pdf"'

    # Создаем PDF документ
    buffer = io.BytesIO()
    doc = SimpleDocTemplate(buffer, pagesize=A4)
    styles = getSampleStyleSheet()

    # Стили
    title_style = ParagraphStyle(
        'CustomTitle',
        parent=styles['Heading1'],
        fontSize=24,
        textColor=colors.HexColor('#2c3e50'),
        spaceAfter=30
```

```
        )

        heading_style = ParagraphStyle(
            'CustomHeading',
            parent=styles['Heading2'],
            fontSize=16,
            textColor=colors.HexColor('#34495e'),
            spaceAfter=15
        )

        # Собираем данные для отчета
        status_stats = StatisticsService.get_task_status_stats(user_id)
        overdue_stats = StatisticsService.get_overdue_stats(user_id)
        productivity = StatisticsService.get_productivity_stats(user_id, period_days)
        priority_dist = StatisticsService.get_priority_distribution(user_id)

        # Содержимое PDF
        story = []

        # Заголовок
        story.append(Paragraph("Отчет по задачам", title_style))
        story.append(Paragraph(f"Период: последние {period_days} дней", styles['Normal']))
        story.append(Paragraph(f"Дата генерации:
{timezone.now().strftime('%d.%m.%Y %H:%M')}", styles['Normal']))
        story.append(Spacer(1, 20))

        # Статистика по статусам
        story.append(Paragraph("Статистика по статусам", heading_style))
        status_data = [
            ['Статус', 'Количество', 'Процент'],
            ['Новые', status_stats.get('New', 0), f"{status_stats.get('New_percent', 0)}%"],
            ['В работе', status_stats.get('InProgress', 0), f"{status_stats.get('InProgress_percent',
0)}%"],
            ['Выполнены', status_stats.get('Completed', 0), f"{status_stats.get('Completed_percent',
0)}%"],
        ]
        status_table = Table(status_data)
        status_table.setStyle(TableStyle([
            ('BACKGROUND', (0, 0), (-1, 0), colors.grey),
            ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
            ('ALIGN', (0, 0), (-1, -1), 'CENTER'),
            ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
            ('FONTSIZE', (0, 0), (-1, 0), 12),
            ('BOTTOMPADDING', (0, 0), (-1, 0), 12),
            ('BACKGROUND', (0, 1), (-1, -1), colors.beige),
            ('GRID', (0, 0), (-1, -1), 1, colors.black),
        ]))
        story.append(status_table)
        story.append(Spacer(1, 20))
```

```
        # Просроченные задачи
        story.append(Paragraph("Срочность", heading_style))
        urgency_data = [
            ['Тип', 'Количество'],
            ['Просроченные', overdue_stats['overdue']],
            ['На сегодня', overdue_stats['due_today']],
            ['На эту неделю', overdue_stats['due_this_week']],
        ]
        urgency_table = Table(urgency_data)
        urgency_table.setStyle(TableStyle([
            ('BACKGROUND', (0, 0), (-1, 0), colors.grey),
            ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
            ('ALIGN', (0, 0), (-1, -1), 'CENTER'),
            ('GRID', (0, 0), (-1, -1), 1, colors.black),
        ]))
        story.append(urgency_table)
        story.append(Spacer(1, 20))

        # Продуктивность
        story.append(Paragraph("Продуктивность", heading_style))
        productivity_data = [
            ['Метрика', 'Значение'],
            ['Выполнено задач', productivity['tasks_completed']],
            ['Создано задач', productivity['tasks_created']],
            ['Процент выполнения', f"{productivity['completion_rate']}%"],
        ]
        productivity_table = Table(productivity_data)
        productivity_table.setStyle(TableStyle([
            ('BACKGROUND', (0, 0), (-1, 0), colors.grey),
            ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
            ('GRID', (0, 0), (-1, -1), 1, colors.black),
        ]))
        story.append(productivity_table)

        # Сборка PDF
        doc.build(story)

        pdf = buffer.getvalue()
        buffer.close()
        response.write(pdf)
        return response

    @staticmethod
    def _get_filtered_tasks(user_id, filters=None):
        """Получение отфильтрованных задач"""
        queryset = Task.objects.filter(user_id=user_id).select_related('project')

        if filters:
            if filters.get('status'):
                queryset = queryset.filter(status=filters['status'])
```

```
        if filters.get('priority'):
            queryset = queryset.filter(priority=filters['priority'])
        if filters.get('project_id'):
            queryset = queryset.filter(project_id=filters['project_id'])
        if filters.get('date_from'):
            queryset = queryset.filter(created_at__gte=filters['date_from'])
        if filters.get('date_to'):
            queryset = queryset.filter(created_at__lte=filters['date_to'])

    return queryset

@staticmethod
def _get_statistics(user_id, filters=None):
    """Получение статистики для экспорта"""
    status_stats = StatisticsService.get_task_status_stats(user_id)
    overdue_stats = StatisticsService.get_overdue_stats(user_id)

    return {
        'Всего задач': Task.objects.filter(user_id=user_id).count(),
        'Новые задачи': status_stats.get('New', 0),
        'Задачи в работе': status_stats.get('InProgress', 0),
        'Выполненные задачи': status_stats.get('Completed', 0),
        'Просроченные задачи': overdue_stats['overdue'],
        'Задачи на сегодня': overdue_stats['due_today'],
        'Задачи на эту неделю': overdue_stats['due_this_week'],
    }
```

Шаг 5. Реализация поиска

tasks/services/search_service.py:

```
from django.db.models import Q
from ..models import Task

class SearchService:
    """Сервис для полнотекстового поиска"""

    @staticmethod
    def search_tasks(user_id, query, filters=None, page=1, page_size=20):
        """
        Полнотекстовый поиск задач
        """
        if not query or len(query.strip()) < 2:
            return {
                'tasks': [],
                'total': 0,
                'query': query
            }

        queryset = Task.objects.filter(user_id=user_id)
```

```
        # Полнотекстовый поиск по заголовку и описанию
        queryset = queryset.filter(
            Q(title__icontains=query) |
            Q(description__icontains=query)
        )

        # Дополнительная фильтрация
        if filters:
            if filters.get('status'):
                queryset = queryset.filter(status=filters['status'])
            if filters.get('priority'):
                queryset = queryset.filter(priority=filters['priority'])
            if filters.get('project_id'):
                queryset = queryset.filter(project_id=filters['project_id'])

        # Ранжирование результатов (сначала точные совпадения по заголовку)
        queryset = queryset.extra(
            select={
                'relevance': """
                    CASE
                        WHEN title ILIKE '%%%s%%' THEN 2
                        WHEN description ILIKE '%%%s%%' THEN 1
                        ELSE 0
                    END
                """ % (query, query)
            },
            order_by=['-relevance', '-created_at']
        )

        total = queryset.count()

        # Пагинация
        start = (page - 1) * page_size
        tasks = queryset[start:start + page_size]

        return {
            'tasks': tasks,
            'total': total,
            'query': query,
            'page': page,
            'page_size': page_size,
            'total_pages': (total + page_size - 1) // page_size
        }

    @staticmethod
    def get_search_suggestions(user_id, query, limit=5):
        """Получение поисковых подсказок"""
        if not query or len(query.strip()) < 2:
            return []
```

```
    # Популярные слова из заголовков задач
    suggestions = Task.objects.filter(
        user_id=user_id,
        title__icontains=query
    ).values_list('title', flat=True).distinct()[:limit]

    return list(suggestions)
```

Шаг 6. Реализация иерархии задач (подзадачи)

tasks/services/subtask_service.py:

```
from django.db import transaction
from django.utils import timezone
from ..models import Task

class SubtaskService:
    """Сервис для работы с подзадачами"""

    @staticmethod
    def get_subtasks(task_id, user_id):
        """Получение всех подзадач задачи"""
        try:
            parent = Task.objects.get(id=task_id, user_id=user_id)
            return parent.subtasks.all()
        except Task.DoesNotExist:
            return []

    @staticmethod
    def create_subtask(parent_id, user_id, data):
        """Создание подзадачи"""
        try:
            parent = Task.objects.get(id=parent_id, user_id=user_id)
        except Task.DoesNotExist:
            return None

        data['user_id'] = user_id
        data['parent_id'] = parent_id

        subtask = Task.objects.create(**data)
        return subtask

    @staticmethod
    def update_parent_status(parent_id, user_id):
        """Обновление статуса родительской задачи на основе подзадач"""
        try:
            parent = Task.objects.get(id=parent_id, user_id=user_id)
        except Task.DoesNotExist:
            return None
```

```
        subtasks = parent.subtasks.all()

        if not subtasks.exists():
            return parent

        total = subtasks.count()
        completed = subtasks.filter(status='Completed').count()

        if completed == 0:
            parent.status = 'New'
        elif completed == total:
            parent.status = 'Completed'
        else:
            parent.status = 'InProgress'

        parent.save()
        return parent

    @staticmethod
    def get_task_tree(user_id, root_task_id=None):
        """Получение дерева задач"""
        if root_task_id:
            tasks = Task.objects.filter(id=root_task_id, user_id=user_id)
        else:
            tasks = Task.objects.filter(user_id=user_id, parent__isnull=True)

        def build_tree(task):
            return {
                'id': task.id,
                'title': task.title,
                'status': task.status,
                'priority': task.priority,
                'deadline': task.deadline,
                'subtasks': [build_tree(st) for st in task.subtasks.all()]
            }

        return [build_tree(task) for task in tasks]
```

Шаг 7. Создание API эндпоинтов

tasks/views/statistics_views.py:

```
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework.permissions import IsAuthenticated
from drf_spectacular.utils import extend_schema
from ..services.statistics_service import StatisticsService
from ..services.export_service import ExportService
from ..services.search_service import SearchService
```

```python
from ..services.subtask_service import SubtaskService

class StatisticsView(APIView):
    """API для получения статистики"""

    @extend_schema(summary="Получить статистику по статусам")
    def get(self, request):
        user_id = request.user.id

        status_stats = StatisticsService.get_task_status_stats(user_id)
        overdue_stats = StatisticsService.get_overdue_stats(user_id)
        productivity = StatisticsService.get_productivity_stats(user_id)
        priority_dist = StatisticsService.get_priority_distribution(user_id)
        project_stats = StatisticsService.get_project_stats(user_id)
        heatmap = StatisticsService.get_weekly_heatmap(user_id)

        return Response({
            'status_stats': status_stats,
            'overdue_stats': overdue_stats,
            'productivity': productivity,
            'priority_distribution': priority_dist,
            'project_stats': project_stats,
            'weekly_heatmap': heatmap
        })

class ExportView(APIView):
    """API для экспорта данных"""

    @extend_schema(summary="Экспорт задач в CSV")
    def get(self, request):
        format_type = request.query_params.get('format', 'tasks')
        file_type = request.query_params.get('type', 'csv')

        filters = {
            'status': request.query_params.get('status'),
            'priority': request.query_params.get('priority'),
            'project_id': request.query_params.get('project_id'),
            'date_from': request.query_params.get('date_from'),
            'date_to': request.query_params.get('date_to'),
        }
        filters = {k: v for k, v in filters.items() if v}

        if file_type == 'csv':
            return ExportService.export_to_csv(request.user.id, format_type, filters)
        elif file_type == 'json':
            return ExportService.export_to_json(request.user.id, format_type, filters)
        elif file_type == 'pdf':
            days = int(request.query_params.get('days', 30))
            return ExportService.export_to_pdf(request.user.id, days)
```

```
        return Response({'error': 'Unsupported format'}, status=400)

class SearchView(APIView):
    """API для поиска задач"""

    @extend_schema(summary="Поиск задач")
    def get(self, request):
        query = request.query_params.get('q', '')
        page = int(request.query_params.get('page', 1))
        page_size = int(request.query_params.get('page_size', 20))

        filters = {
            'status': request.query_params.get('status'),
            'priority': request.query_params.get('priority'),
            'project_id': request.query_params.get('project_id'),
        }
        filters = {k: v for k, v in filters.items() if v}

        result = SearchService.search_tasks(request.user.id, query, filters, page, page_size)
        return Response(result)

class SubtaskView(APIView):
    """API для работы с подзадачами"""

    @extend_schema(summary="Получить подзадачи")
    def get(self, request, task_id):
        subtasks = SubtaskService.get_subtasks(task_id, request.user.id)
        return Response([{'id': t.id, 'title': t.title} for t in subtasks])

    @extend_schema(summary="Создать подзадачу")
    def post(self, request, task_id):
        data = {
            'title': request.data.get('title'),
            'description': request.data.get('description'),
            'status': 'New',
            'priority': request.data.get('priority', 'Medium')
        }

        subtask = SubtaskService.create_subtask(task_id, request.user.id, data)
        if subtask:
            return Response({'id': subtask.id, 'title': subtask.title}, status=201)
        return Response({'error': 'Parent task not found'}, status=404)

    @extend_schema(summary="Получить дерево задач")
    def get_tree(self, request):
        tree = SubtaskService.get_task_tree(request.user.id)
        return Response(tree)
```

Шаг 8. Добавление маршрутов

tasks/urls.py (дополнение):

```
from django.urls import path
from .views.statistics_views import StatisticsView, ExportView, SearchView, SubtaskView

urlpatterns += [
    path('api/statistics/', StatisticsView.as_view(), name='statistics'),
    path('api/export/', ExportView.as_view(), name='export'),
    path('api/search/', SearchView.as_view(), name='search'),
    path('api/tasks/<int:task_id>/subtasks/', SubtaskView.as_view(), name='subtasks'),
    path('api/tasks/tree/', SubtaskView.as_view(), {'action': 'tree'}, name='task-tree'),
]
```

-*- Траектория Б (ASP.NET Core + React)

Шаг 1. Сервис статистики

```
using Microsoft.EntityFrameworkCore;
using SmartDiary.Web.Data;
using SmartDiary.Web.Models;

namespace SmartDiary.Web.Services
{
    public interface IStatisticsService
    {
        Task<StatusStatsDto> GetTaskStatusStatsAsync(int userId);
        Task<OverdueStatsDto> GetOverdueStatsAsync(int userId);
        Task<ProductivityStatsDto> GetProductivityStatsAsync(int userId, int days);
        Task<PriorityDistributionDto> GetPriorityDistributionAsync(int userId);
        Task<List<ProjectStatsDto>> GetProjectStatsAsync(int userId);
        Task<List<WeeklyHeatmapDto>> GetWeeklyHeatmapAsync(int userId, int weeks);
    }

    public class StatisticsService : IStatisticsService
    {
        private readonly ApplicationDbContext _context;

        public StatisticsService(ApplicationDbContext context)
        {
            _context = context;
        }

        public async Task<StatusStatsDto> GetTaskStatusStatsAsync(int userId)
        {
            var total = await _context.Tasks.CountAsync(t => t.UserId == userId);

            var statusGroups = await _context.Tasks
                .Where(t => t.UserId == userId)
                .GroupBy(t => t.Status)
                .Select(g => new { Status = g.Key, Count = g.Count() })
```

```
            .ToDictionaryAsync(x => x.Status, x => x.Count);

        return new StatusStatsDto
        {
            New = statusGroups.GetValueOrDefault("New", 0),
            InProgress = statusGroups.GetValueOrDefault("InProgress", 0),
            Completed = statusGroups.GetValueOrDefault("Completed", 0),
            NewPercent = total > 0 ? Math.Round((double)statusGroups.GetValueOrDefault("New",
0) / total * 100, 1) : 0,
            InProgressPercent = total > 0 ?
Math.Round((double)statusGroups.GetValueOrDefault("InProgress", 0) / total * 100, 1) : 0,
            CompletedPercent = total > 0 ?
Math.Round((double)statusGroups.GetValueOrDefault("Completed", 0) / total * 100, 1) : 0
        };
    }

    public async Task<OverdueStatsDto> GetOverdueStatsAsync(int userId)
    {
        var now = DateTime.UtcNow;
        var today = now.Date;

        var overdue = await _context.Tasks
            .CountAsync(t => t.UserId == userId &&
                        t.Deadline < now &&
                        (t.Status == "New" || t.Status == "InProgress"));

        var dueToday = await _context.Tasks
            .CountAsync(t => t.UserId == userId &&
                        t.Deadline != null &&
                        t.Deadline.Value.Date == today &&
                        (t.Status == "New" || t.Status == "InProgress"));

        var dueThisWeek = await _context.Tasks
            .CountAsync(t => t.UserId == userId &&
                        t.Deadline != null &&
                        t.Deadline.Value.Date <= today.AddDays(7) &&
                        t.Deadline.Value.Date >= today &&
                        (t.Status == "New" || t.Status == "InProgress"));

        return new OverdueStatsDto
        {
            Overdue = overdue,
            DueToday = dueToday,
            DueThisWeek = dueThisWeek,
            HasOverdue = overdue > 0,
            HasDueToday = dueToday > 0
        };
    }

    public async Task<ProductivityStatsDto> GetProductivityStatsAsync(int userId, int days)
```

```
    {
        var startDate = DateTime.UtcNow.AddDays(-days);

        var completed = await _context.Tasks
            .CountAsync(t => t.UserId == userId &&
                             t.Status == "Completed" &&
                             t.CompletedAt >= startDate);

        var created = await _context.Tasks
            .CountAsync(t => t.UserId == userId && t.CreatedAt >= startDate);

        var dailyStats = await _context.Tasks
            .Where(t => t.UserId == userId && t.CreatedAt >= startDate)
            .GroupBy(t => t.CreatedAt.Date)
            .Select(g => new DailyStatsDto
            {
                Date = g.Key,
                Created = g.Count(),
                Completed = _context.Tasks.Count(t => t.UserId == userId &&
                                                      t.Status == "Completed" &&
                                                      t.CompletedAt.HasValue &&
                                                      t.CompletedAt.Value.Date == g.Key)
            })
            .OrderBy(x => x.Date)
            .ToListAsync();

        return new ProductivityStatsDto
        {
            PeriodDays = days,
            TasksCompleted = completed,
            TasksCreated = created,
            CompletionRate = created > 0 ? Math.Round((double)completed / created * 100, 1) : 0,
            DailyStats = dailyStats
        };
    }

    public async Task<PriorityDistributionDto> GetPriorityDistributionAsync(int userId)
    {
        var total = await _context.Tasks
            .CountAsync(t => t.UserId == userId && (t.Status == "New" || t.Status == "InProgress"));

        var distribution = await _context.Tasks
            .Where(t => t.UserId == userId && (t.Status == "New" || t.Status == "InProgress"))
            .GroupBy(t => t.Priority)
            .Select(g => new PriorityItemDto
            {
                Priority = g.Key,
                Count = g.Count(),
                Percent = total > 0 ? Math.Round((double)g.Count() / total * 100, 1) : 0
            })
```

```
            .ToDictionaryAsync(x => x.Priority, x => x);

        return new PriorityDistributionDto
        {
            Low = distribution.GetValueOrDefault("Low"),
            Medium = distribution.GetValueOrDefault("Medium"),
            High = distribution.GetValueOrDefault("High")
        };
    }

    public async Task<List<ProjectStatsDto>> GetProjectStatsAsync(int userId)
    {
        var projects = await _context.Projects
            .Include(p => p.Tasks)
            .Where(p => p.OwnerId == userId)
            .ToListAsync();

        return projects.Select(p => new ProjectStatsDto
        {
            Id = p.Id,
            Name = p.Name,
            Color = p.Color,
            TotalTasks = p.Tasks.Count,
            CompletedTasks = p.Tasks.Count(t => t.Status == "Completed"),
            CompletionRate = p.Tasks.Count > 0 ?
                Math.Round((double)p.Tasks.Count(t => t.Status == "Completed") / p.Tasks.Count *
100, 1) : 0
        }).ToList();
    }

    public async Task<List<WeeklyHeatmapDto>> GetWeeklyHeatmapAsync(int userId, int
weeks)
    {
        var startDate = DateTime.UtcNow.AddDays(-weeks * 7);
        var endDate = DateTime.UtcNow;

        // Получаем все дни периода
        var allDays = Enumerable.Range(0, (endDate - startDate).Days + 1)
            .Select(offset => startDate.AddDays(offset))
            .ToList();

        // Получаем выполненные задачи по дням
        var completedByDate = await _context.Tasks
            .Where(t => t.UserId == userId &&
                        t.Status == "Completed" &&
                        t.CompletedAt >= startDate)
            .GroupBy(t => t.CompletedAt!.Value.Date)
            .Select(g => new { Date = g.Key, Count = g.Count() })
            .ToDictionaryAsync(x => x.Date, x => x.Count);
```

```
        return allDays.Select(date => new WeeklyHeatmapDto
        {
            Date = date,
            DayOfWeek = (int)date.DayOfWeek,
            Week = ISOWeek.GetWeekOfYear(date),
            Count = completedByDate.GetValueOrDefault(date.Date, 0)
        }).ToList();
    }
}

// DTO классы
public class StatusStatsDto
{
    public int New { get; set; }
    public int InProgress { get; set; }
    public int Completed { get; set; }
    public double NewPercent { get; set; }
    public double InProgressPercent { get; set; }
    public double CompletedPercent { get; set; }
}

public class OverdueStatsDto
{
    public int Overdue { get; set; }
    public int DueToday { get; set; }
    public int DueThisWeek { get; set; }
    public bool HasOverdue { get; set; }
    public bool HasDueToday { get; set; }
}

public class ProductivityStatsDto
{
    public int PeriodDays { get; set; }
    public int TasksCompleted { get; set; }
    public int TasksCreated { get; set; }
    public double CompletionRate { get; set; }
    public List<DailyStatsDto> DailyStats { get; set; } = new();
}

public class DailyStatsDto
{
    public DateTime Date { get; set; }
    public int Created { get; set; }
    public int Completed { get; set; }
}

public class PriorityItemDto
{
    public string Priority { get; set; } = string.Empty;
    public int Count { get; set; }
```

```
    public double Percent { get; set; }
  }

  public class PriorityDistributionDto
  {
    public PriorityItemDto? Low { get; set; }
    public PriorityItemDto? Medium { get; set; }
    public PriorityItemDto? High { get; set; }
  }

  public class ProjectStatsDto
  {
    public int Id { get; set; }
    public string Name { get; set; } = string.Empty;
    public string Color { get; set; } = string.Empty;
    public int TotalTasks { get; set; }
    public int CompletedTasks { get; set; }
    public double CompletionRate { get; set; }
  }

  public class WeeklyHeatmapDto
  {
    public DateTime Date { get; set; }
    public int DayOfWeek { get; set; }
    public int Week { get; set; }
    public int Count { get; set; }
  }
}
```

Шаг 2. Сервис экспорта

Services/ExportService.cs:

```
using System.Text;
using System.Text.Json;
using Microsoft.EntityFrameworkCore;

namespace SmartDiary.Web.Services
{
  public interface IExportService
  {
    Task<byte[]> ExportToCsvAsync(int userId, string format, Dictionary<string, string>? filters);
    Task<byte[]> ExportToJsonAsync(int userId, string format, Dictionary<string, string>?
filters);
    Task<byte[]> ExportToPdfAsync(int userId, int days);
  }

  public class ExportService : IExportService
```

```
{
    private readonly ApplicationDbContext _context;
    private readonly IStatisticsService _statisticsService;

    public ExportService(ApplicationDbContext context, IStatisticsService statisticsService)
    {
        _context = context;
        _statisticsService = statisticsService;
    }

    public async Task<byte[]> ExportToCsvAsync(int userId, string format, Dictionary<string,
string>? filters)
    {
        var sb = new StringBuilder();

        switch (format)
        {
            case "tasks":
                var tasks = await GetFilteredTasks(userId, filters);
                sb.AppendLine("ID,Заголовок,Описание,Статус,Приоритет,Дедлайн,Создана,Про
ект");
                foreach (var task in tasks)
                {
                    sb.AppendLine($"\"{task.Id}\",\"{EscapeCsv(task.Title)}\",\"{EscapeCsv(task.Descript
ion)}\",\"{task.Status}\",\"{task.Priority}\",\"{task.Deadline:yyyy-MM-dd
HH:mm}\",\"{task.CreatedAt:yyyy-MM-dd HH:mm}\",\"{task.Project?.Name}\"");
                }
                break;

            case "projects":
                var projects = await _context.Projects
                    .Where(p => p.OwnerId == userId)
                    .ToListAsync();
                sb.AppendLine("ID,Название,Описание,Цвет,Создан,Кол-во задач");
                foreach (var project in projects)
                {
                    var taskCount = await _context.Tasks.CountAsync(t => t.ProjectId == project.Id);
                    sb.AppendLine($"\"{project.Id}\",\"{EscapeCsv(project.Name)}\",\"{EscapeCsv(proje
ct.Description)}\",\"{project.Color}\",\"{project.CreatedAt:yyyy-MM-dd}\",\"{taskCount}\"");
                }
                break;
        }

        return Encoding.UTF8.GetBytes(sb.ToString());
    }

    public async Task<byte[]> ExportToJsonAsync(int userId, string format, Dictionary<string,
string>? filters)
    {
        object data;
```

```
        switch (format)
        {
            case "tasks":
                var tasks = await GetFilteredTasks(userId, filters);
                data = tasks.Select(t => new
                {
                    t.Id,
                    t.Title,
                    t.Description,
                    t.Status,
                    t.Priority,
                    Deadline = t.Deadline?.ToString("yyyy-MM-dd HH:mm"),
                    CreatedAt = t.CreatedAt.ToString("yyyy-MM-dd HH:mm"),
                    Project = t.Project != null ? new { t.Project.Id, t.Project.Name } : null
                });
                break;

            default:
                data = await _statisticsService.GetTaskStatusStatsAsync(userId);
                break;
        }

        var json = JsonSerializer.Serialize(data, new JsonSerializerOptions
        {
            WriteIndented = true,
            Encoder =
System.Text.Encodings.Web.JavaScriptEncoder.Create(System.Text.Unicode.UnicodeRanges.All)
        });

        return Encoding.UTF8.GetBytes(json);
    }

    public async Task<byte[]> ExportToPdfAsync(int userId, int days)
    {
        // Для PDF используется сторонняя библиотека (например, iTextSharp или QuestPDF)
        // Здесь приведена упрощенная структура
        var statusStats = await _statisticsService.GetTaskStatusStatsAsync(userId);
        var overdueStats = await _statisticsService.GetOverdueStatsAsync(userId);
        var productivity = await _statisticsService.GetProductivityStatsAsync(userId, days);

        var html = $@"
        <html>
        <head>
            <meta charset='utf-8'>
            <title>Отчет по задачам</title>
            <style>
                body {{ font-family: Arial, sans-serif; margin: 40px; }}
                h1 {{ color: #2c3e50; }}
                h2 {{ color: #34495e; margin-top: 30px; }}
```

```
            table {{ border-collapse: collapse; width: 100%; margin: 20px 0; }}
            th, td {{ border: 1px solid #ddd; padding: 8px; text-align: left; }}
            th {{ background-color: #34495e; color: white; }}
            .metric {{ font-size: 24px; font-weight: bold; color: #27ae60; }}
        </style>
    </head>
    <body>
        <h1>Отчет по задачам</h1>
        <p>Период: последние {days} дней</p>
        <p>Дата: {DateTime.Now:dd.MM.yyyy HH:mm}</p>

        <h2>Статистика по статусам</h2>
        <table>
            <tr><th>Статус</th><th>Количество</th><th>Процент</th></tr>
            <tr><td>Новые</td><td>{statusStats.New}</td><td>{statusStats.NewPercent}%</td
></tr>
            <tr><td>В
работе</td><td>{statusStats.InProgress}</td><td>{statusStats.InProgressPercent}%</td></tr>
            <tr><td>Выполнены</td><td>{statusStats.Completed}</td><td>{statusStats.Compl
etedPercent}%</td></tr>
        </table>

        <h2>Срочность</h2>
        <table>
            <tr><th>Тип</th><th>Количество</th></tr>
            <tr><td>Просроченные</td><td>{overdueStats.Overdue}</td></tr>
            <tr><td>На сегодня</td><td>{overdueStats.DueToday}</td></tr>
            <tr><td>На эту неделю</td><td>{overdueStats.DueThisWeek}</td></tr>
        </table>

        <h2>Продуктивность</h2>
        <table>
            <tr><th>Метрика</th><th>Значение</th></tr>
            <tr><td>Выполнено задач</td><td>{productivity.TasksCompleted}</td></tr>
            <tr><td>Создано задач</td><td>{productivity.TasksCreated}</td></tr>
            <tr><td>Процент выполнения</td><td>{productivity.CompletionRate}%</td></tr>
        </table>
    </body>
    </html>";

    // Конвертация HTML в PDF (требуется установка пакета)
    // Для примера возвращаем HTML как текст
    return Encoding.UTF8.GetBytes(html);
}

private async Task<List<Task>> GetFilteredTasks(int userId, Dictionary<string, string>?
filters)
{
    var query = _context.Tasks
        .Include(t => t.Project)
```

```
            .Where(t => t.UserId == userId)
            .AsQueryable();

        if (filters != null)
        {
            if (filters.ContainsKey("status"))
                query = query.Where(t => t.Status == filters["status"]);
            if (filters.ContainsKey("priority"))
                query = query.Where(t => t.Priority == filters["priority"]);
            if (filters.ContainsKey("project_id") && int.TryParse(filters["project_id"], out var
projectId))
                query = query.Where(t => t.ProjectId == projectId);
            if (filters.ContainsKey("date_from") && DateTime.TryParse(filters["date_from"], out var
dateFrom))
                query = query.Where(t => t.CreatedAt >= dateFrom);
            if (filters.ContainsKey("date_to") && DateTime.TryParse(filters["date_to"], out var
dateTo))
                query = query.Where(t => t.CreatedAt <= dateTo);
        }

        return await query.ToListAsync();
    }

    private static string EscapeCsv(string?value)
    {
        if (string.IsNullOrEmpty(value)) return "";
        return value.Replace("\"", "\"\"");
    }
  }
}
```

Контрольные вопросы

1. Какие агрегатные функции используются для подсчета статистики?
2. В чем разница между экспортом в CSV и JSON?
3. Как работает полнотекстовый поиск в БД?
4. Какие есть стратегии реализации иерархии задач?
5. Как рассчитать процент выполнения задач?
6. Что такое тепловая карта активности?

Критерии оценки

Критерий	Баллы
Реализована статистика по статусам, приоритетам,	2

Критерий	Баллы
проектам	
Реализован экспорт в CSV, JSON, PDF	2
Реализован полнотекстовый поиск	2
Реализована иерархия задач (подзадачи)	2
Реализованы API эндпоинты для статистики и экспорта	1
Код работает без ошибок	1
Итого	10

Результат выполнения

После выполнения лабораторной работы №6 студенты должны получить:

1. Аналитическую панель с ключевыми метриками
2. Функции экспорта данных в CSV, JSON, PDF
3. Полнотекстовый поиск по задачам
4. Иерархию задач (подзадачи)
5. API эндпоинты для всех новых функций

Лабораторная работа №7

Введение в React. Компонентный подход

Цель работы

Создание клиентского приложения на React с нуля: освоение компонентного подхода, маршрутизации, создания базовых компонентов страниц и настройки HTTP-клиента для взаимодействия с бэкендом.

Задачи

№	Задача
1	Создать React-проект с использованием Vite
2	Настроить маршрутизацию (React Router)
3	Создать основные компоненты страниц (Dashboard, TaskList, TaskForm)
4	Реализовать переиспользуемые компоненты (TaskCard, TaskFilter, Navbar, Sidebar)
5	Настроить HTTP-клиент (axios) для связи с API бэкенда

Теоретическая часть

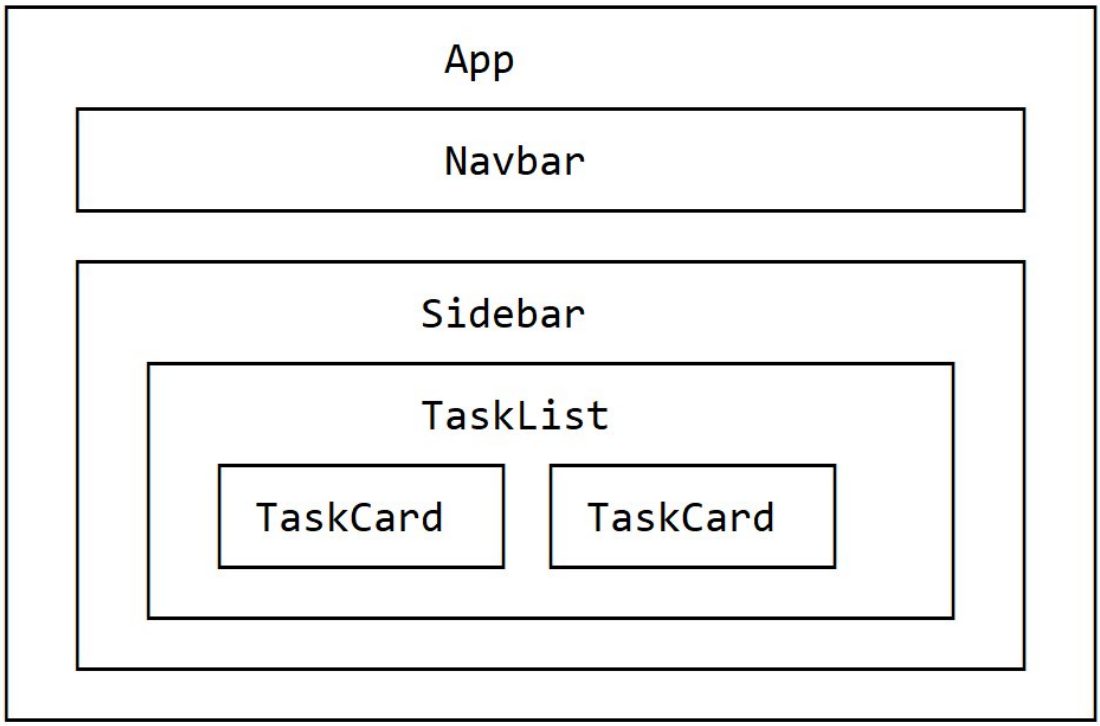
React

React - JavaScript-библиотека для создания пользовательских интерфейсов, разработанная Facebook.

Ключевые концепции:

- Компоненты - строительные блоки интерфейса
- JSX - расширение JavaScript, позволяющее писать HTML-подобный код
- State — внутреннее состояние компонента
- Props — свойства, передаваемые компоненту
- Hooks — функции для работы с состоянием и побочными эффектами

Компонентный подход



Маршрутизация (React Router)

React Router позволяет реализовать навигацию между страницами без перезагрузки.

Основные компоненты:

- BrowserRouter - обёртка для всего приложения
- Routes - контейнер для маршрутов
- Route - отдельный маршрут (путь → компонент)
- Link / NavLink - компоненты для навигации
- HTTP-клиент (Axios)
- Axios - библиотека для выполнения HTTP-запросов из браузера.

Преимущества:

- Поддержка Promise API
- Автоматическая трансформация JSON
- Перехватчики (interceptors)
- Настройка базового URL и заголовков

Ход выполнения работы

Общие требования

1. Приложение должно быть одностраничным (SPA)

2. Все компоненты должны быть функциональными с использованием хуков
3. Интерфейс должен быть адаптивным (использовать CSS-модули или styled-components)

Траектория A (Django + React)

Шаг 1. Создание React-проекта

bash

```
# Создание проекта с Vite
npm create vite@latest smart-diary-frontend -- --template react
# Переход в папку проекта
cd smart-diary-frontend
# Установка зависимостей
npm install
# Установка дополнительных пакетов
npm install axios react-router-dom
npm install @mui/material @emotion/react @emotion/styled @mui/icons-material
# Запуск проекта
npm run dev
```

Шаг 2. Структура проекта

```
src/
├── api/
│   └── client.js           # Настройка axios
├── components/
│   ├── common/
│   │   ├── Navbar.jsx
│   │   ├── Sidebar.jsx
│   │   └── LoadingSpinner.jsx
│   ├── tasks/
│   │   ├── TaskList.jsx
│   │   ├── TaskCard.jsx
│   │   ├── TaskForm.jsx
│   │   └── TaskFilter.jsx
│   └── layout/
│       └── Layout.jsx
├── pages/
│   ├── Dashboard.jsx
│   ├── TaskDetail.jsx
│   ├── TaskCreate.jsx
│   ├── TaskEdit.jsx
│   └── Profile.jsx
├── hooks/
│   └── useTasks.js
├── styles/
│   └── global.css
├── App.jsx
├── App.css
└── main.jsx
```

Шаг 3. Настройка HTTP-клиента

src/api/client.js:

javascript

```
import axios from 'axios';

const API_BASE_URL = import.meta.env.VITE_API_URL || 'http://localhost:8000/api';

const apiClient = axios.create({
  baseURL: API_BASE_URL,
  headers: {
    'Content-Type': 'application/json',
  },
});

// Request interceptor для добавления токена
apiClient.interceptors.request.use(
  (config) => {
    const token = localStorage.getItem('access_token');
    if (token) {
      config.headers.Authorization = `Bearer ${token}`;
    }
    return config
  },
  (error) => Promise.reject(error)
);

// Response interceptor для обработки ошибок
apiClient.interceptors.response.use(
  (response) => response,
  (error) => {
    if (error.response?.status === 401) {
      localStorage.removeItem('access_token');
      localStorage.removeItem('refresh_token');
      window.location.href = '/login';
    }
    return Promise.reject(error);
  }
);
```

```
export default apiClient;
```

Шаг 4. API сервисы

src/api/taskService.js:

```
import apiClient from './client';

export const taskService = {
  // Получение списка задач
  getTasks: async (page = 1, pageSize = 10, filters = {}) => {
    const params = { page, page_size: pageSize, ...filters };
    const response = await apiClient.get('/tasks/', { params });
    return response.data;
  },

  // Получение задачи по ID
  getTaskById: async (id) => {
    const response = await apiClient.get(`/tasks/${id}/`);
    return response.data;
  },

  // Создание задачи
  createTask: async (taskData) => {
    const response = await apiClient.post('/tasks/', taskData);
    return response.data;
  },

  // Обновление задачи
  updateTask: async (id, taskData) => {
    const response = await apiClient.put(`/tasks/${id}/`, taskData);
    return response.data;
  },

  // Удаление задачи
  deleteTask: async (id) => {
```

```
    await apiClient.delete(`/tasks/${id}/`);
  },

  // Изменение статуса задачи
  updateTaskStatus: async (id, status) => {
    const response = await apiClient.patch(`/tasks/${id}/`, { status });
    return response.data;
  },
};

export const projectService = {
  getProjects: async () => {
    const response = await apiClient.get('/projects/');
    return response.data;
  },
};

export const tagService = {
  getTags: async () => {
    const response = await apiClient.get('/tags/');
    return response.data;
  },
};
```

Шаг 5. Кастомный хук для работы с задачами

src/hooks/useTasks.js:

```
import { useState, useEffect, useCallback } from 'react';
import { taskService } from '../api/taskService';

export const useTasks = (initialFilters = {}) => {
  const [tasks, setTasks] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  const [pagination, setPagination] = useState({
```

```
    page: 1,
    pageSize: 10,
    total: 0,
    totalPages: 0,
  });
  const [filters, setFilters] = useState(initialFilters);

  const loadTasks = useCallback(async () => {
    setLoading(true);
    setError(null);
    try {
      const data = await taskService.getTasks(
        pagination.page,
        pagination.pageSize,
        filters
      );
      setTasks(data.tasks || []);
      setPagination(prev => ({
        ...prev,
        total: data.total || 0,
        totalPages: data.total_pages || 0,
      }));
    } catch (err) {
      setError(err.response?.data?.detail || 'Ошибка загрузки задач');
      console.error('Error loading tasks:', err);
    } finally {
      setLoading(false);
    }
  }, [pagination.page, pagination.pageSize, filters]);

  useEffect(() => {
    loadTasks();
  }, [loadTasks]);

  const createTask = async (taskData) => {
```

```
    try {
      const newTask = await taskService.createTask(taskData);
      await loadTasks();
      return { success: true, data: newTask };
    } catch (err) {
      return {
        success: false,
        error: err.response?.data || 'Ошибка создания задачи'
      };
    }
  };

  const updateTask = async (id, taskData) => {
    try {
      const updatedTask = await taskService.updateTask(id, taskData);
      await loadTasks();
      return { success: true, data: updatedTask };
    } catch (err) {
      return {
        success: false,
        error: err.response?.data || 'Ошибка обновления задачи'
      };
    }
  };

  const deleteTask = async (id) => {
    try {
      await taskService.deleteTask(id);
      await loadTasks();
      return { success: true };
    } catch (err) {
      return {
        success: false,
        error: err.response?.data || 'Ошибка удаления задачи'
      };
```

```
    }
  };

  const changePage = (newPage) => {
    setPagination(prev => ({ ...prev, page: newPage }));
  };

  const changeFilters = (newFilters) => {
    setFilters(prev => ({ ...prev, ...newFilters }));
    setPagination(prev => ({ ...prev, page: 1 }));
  };

  return {
    tasks,
    loading,
    error,
    pagination,
    filters,
    createTask,
    updateTask,
    deleteTask,
    changePage,
    changeFilters,
    reloadTasks: loadTasks,
  };
};
```

Шаг 6. Создание компонента TaskCard

src/components/tasks/TaskCard.jsx:

```
import React, { useState } from 'react';
import {
  Card,
  CardContent
  CardActions
  Typography,
```

```
    Chip,
    IconButton,
    Box,
    Menu,
    MenuItem,
    Checkbox
} from '@mui/material';
import {
    MoreVert as MoreVertIcon,
    Edit as EditIcon,
    Delete as DeleteIcon,
    AccessTime as AccessTimeIcon,
    Flag as FlagIcon,
} from '@mui/icons-material';
import { useNavigate } from 'react-router-dom';
import './TaskCard.css';

const priorityColors = {
    Low: { color: '#2ecc71', label: 'Низкий' },
    Medium: { color: '#f39c12', label: 'Средний' },
    High: { color: '#e74c3c', label: 'Высокий' },
};

const statusColors = {
    New: { color: '#3498db', label: 'Новая' },
    InProgress: { color: '#f39c12', label: 'В работе' },
    Completed: { color: '#2ecc71', label: 'Выполнена' },
};

const TaskCard = ({ task, onStatusChange, onDelete, onEdit }) => {
    const navigate = useNavigate();
    const [anchorEl, setAnchorEl] = useState(null);
    const open = Boolean(anchorEl);

    const handleMenuClick = (event) => {
```

```
    setAnchorEl(event.currentTarget);
  };

  const handleMenuClose = () => {
    setAnchorEl(null);
  };

  const handleEdit = () => {
    handleMenuClose();
    onEdit?.(task.id);
    navigate(`/tasks/${task.id}/edit`);
  };

  const handleDelete = () => {
    handleMenuClose();
    if (window.confirm('Вы уверены, что хотите удалить эту задачу?')) {
      onDelete?.(task.id);
    }
  };

  const formatDate = (dateString) => {
    if (!dateString) return 'Без дедлайна';
    const date = new Date(dateString);
    const today = new Date();
    const tomorrow = new Date(today);
    tomorrow.setDate(tomorrow.getDate() + 1);

    if (date.toDateString() === today.toDateString()) {
      return 'Сегодня';
    }
    if (date.toDateString() === tomorrow.toDateString()) {
      return 'Завтра';
    }
    return date.toLocaleDateString('ru-RU', {
      day: 'numeric',
```

```
                month: 'long',
                hour: '2-digit',
                minute: '2-digit',
              });
            };

            const isOverdue = task deadline && new Date(task deadline) < new Date() &&
task status !== 'Completed';

            return (
              <Card className={`task-card ${task status === 'Completed' ? 'completed' : ''}`}>
                <CardContent>
                  <Box display="flex" alignItems="flex-start" justifyContent="space-between">
                    <Box display="flex" alignItems="center" gap={1} flex={1}>
                      <Checkbox
                        checked={task status === 'Completed'}
                        onChange={() => onStatusChange?.(task id, task status === 'Completed' ?
'New' : 'Completed')}
                        color="success"
                      />
                      <Typography
                        variant="h6"
                        component="h3"
                        sx={{
                          textDecoration: task status === 'Completed' ? 'line-through' : 'none',
                          opacity: task status === 'Completed' ? 0.7 : 1,
                          flex: 1,
                        }}
                      >
                        {task title}
                      </Typography>
                      <IconButton size="small" onClick={handleMenuClick}>
                        <MoreVertIcon />
                      </IconButton>
                    </Box>
```

```
</Box>

{task description && (
  <Typography
    variant="body2"
    color="text.secondary"
    sx={{ mt: 1, ml: 5, mb: 2 }}
  >
    {task description.length > 100
      ? `${task description.substring(0, 100)}...`
      : task description}
  </Typography>
)}

<Box display="flex" flexWrap="wrap" gap={1} sx={{ mt: 2, ml: 5 }}>
  <Chip
    size="small"
    icon={<FlagIcon />}
    label={priorityColors[task priority]?.label || task priority}
    sx={{
      bgcolor: priorityColors[task priority]?.color,
      color: 'white',
    }}
  />
  <Chip
    size="small"
    label={statusColors[task status]?.label || task status}
    sx={{
      bgcolor: statusColors[task status]?.color,
      color: 'white',
    }}
  />
  {task deadline && (
    <Chip
      size="small"
```

```
            icon={<AccessTimeIcon />}
            label={formatDate(task.deadline)}
            sx={{
              bgcolor: isOverdue ? '#e74c3c' : '#95a5a6',
              color: 'white',
            }}
          />
        )}
        {task.project && (
          <Chip
            size="small"
            label={task.project.name}
            sx={{ bgcolor: task.project.color || '#9b59b6', color: 'white' }}
          />
        )}
        {task.tags?.map((tag) => (
          <Chip
            key={tag.id}
            size="small"
            label={tag.name}
            variant="outlined"
          />
        ))}
      </Box>
    </CardContent>

    <Menu anchorEl={anchorEl} open={open} onClose={handleMenuClose}>
      <MenuItem onClick={handleEdit}>
        <EditIcon fontSize="small" sx={{ mr: 1 }} />
        Редактировать
      </MenuItem>
      <MenuItem onClick={handleDelete} sx={{ color: '#e74c3c' }}>
        <DeleteIcon fontSize="small" sx={{ mr: 1 }} />
        Удалить
      </MenuItem>
```

```
        </Menu>
      </Card>
    );
};

export default TaskCard;
```

src/components/tasks/TaskCard.css:

```
.task-card {
  margin-bottom: 16px;
  transition: transform 0.2s, box-shadow 0.2s;
}

.task-card:hover {
  transform: translateY(-2px);
  box-shadow: 0 6px 12px rgba(0, 0, 0, 0.1);
}

.task-card.completed {
  opacity: 0.8;
  background-color: #f8f9fa;
}
```

Шаг 7. Создание компонента TaskList

src/components/tasks/TaskList.jsx:

```
import React from 'react';
import { Box, Typography, CircularProgress, Alert, Pagination } from '@mui/material';
import TaskCard from './TaskCard';
import TaskFilter from './TaskFilter';

const TaskList = ({
  tasks
  loading
  error
```

```
  pagination,
  filters,
  onPageChange,
  onFilterChange,
  onStatusChange,
  onDelete,
}) => {
  if (loading) {
    return (
      <Box display="flex" justifyContent="center" alignItems="center" minHeight="200px">
        <CircularProgress />
      </Box>
    );
  }

  if (error) {
    return (
      <Alert severity="error" sx={{ m: 2 }}>
        {error}
      </Alert>
    );
  }

  if (!tasks || tasks.length === 0) {
    return (
      <Box textAlign="center" py={4}>
        <Typography variant="h6" color="text.secondary">
          Нет задач
        </Typography>
        <Typography variant="body2" color="text.secondary">
          Создайте первую задачу  нажав кнопку «Новая задача»
        </Typography>
      </Box>
    );
  }
```

```
  return (
    <Box>
      <TaskFilter filters={filters} onFilterChange={onFilterChange} />

      <Box>
        {tasks.map((task) => (
          <TaskCard
            key={task.id}
            task={task}
            onStatusChange={onStatusChange}
            onDelete={onDelete}
          />
        ))}
      </Box>

      {pagination.totalPages > 1 && (
        <Box display="flex" justifyContent="center" mt={3}>
          <Pagination
            count={pagination.totalPages}
            page={pagination.page}
            onChange={(_, page) => onPageChange(page)}
            color="primary"
          />
        </Box>
      )}
    </Box>
  );
};

export default TaskList;
```

Шаг 8. Создание компонента TaskFilter

src/components/tasks/TaskFilter.jsx:

```jsx
import React, { useState, useEffect } from 'react';
import {
  Paper,
  Box,
  TextField,
  Select,
  MenuItem,
  FormControl,
  InputLabel,
  Button,
  Chip,
  IconButton,
} from '@mui/material';
import { Search as SearchIcon, Clear as ClearIcon } from '@mui/icons-material';
import { projectService, tagService } from '../../api/taskService';

const TaskFilter = ({ filters, onFilterChange }) => {
  const [localFilters, setLocalFilters] = useState(filters);
  const [projects, setProjects] = useState([]);
  const [tags, setTags] = useState([]);

  useEffect(() => {
    const loadData = async () => {
      try {
        const [projectsData, tagsData] = await Promise.all([
          projectService.getProjects(),
          tagService.getTags(),
        ]);
        setProjects(projectsData);
        setTags(tagsData);
      } catch (error) {
        console.error('Error loading filter data:', error);
      }
    };
    loadData();
```

```
}, []);

const handleChange = (field, value) => {
  setLocalFilters(prev => ({ ...prev, [field]: value }));
};

const handleApply = () => {
  onFilterChange(localFilters);
};

const handleClear = () => {
  const emptyFilters = {
    status: '',
    priority: '',
    project_id: '',
    search: '',
  };
  setLocalFilters(emptyFilters);
  onFilterChange(emptyFilters);
};

const handleKeyPress = (e) => {
  if (e.key === 'Enter') {
    handleApply();
  }
};

return (
  <Paper elevation={2} sx={{ p: 2, mb: 3 }}>
    <Box display="flex" flexWrap="wrap" gap={2} alignItems="flex-end">
      <TextField
        label="Поиск"
        variant="outlined"
        size="small"
        value={localFilters.search || ''}
```

```
      onChange={(e) => handleChange('search', e.target.value)}
      onKeyPress={handleKeyPress}
      placeholder="Поиск по задачам..."
      InputProps={{
        startAdornment: <SearchIcon sx={{ mr: 1, color: 'text.secondary' }} />,
      }}
      sx={{ minWidth: 200, flex: 1 }}
    />

    <FormControl size="small" sx={{ minWidth: 120 }}>
      <InputLabel>Статус</InputLabel>
      <Select
        value={localFilters.status || ''}
        onChange={(e) => handleChange('status', e.target.value)}
        label="Статус"
      >
        <MenuItem value="">Все</MenuItem>
        <MenuItem value="New">Новые</MenuItem>
        <MenuItem value="InProgress">В работе</MenuItem>
        <MenuItem value="Completed">Выполненные</MenuItem>
      </Select>
    </FormControl>

    <FormControl size="small" sx={{ minWidth: 120 }}>
      <InputLabel>Приоритет</InputLabel>
      <Select
        value={localFilters.priority || ''}
        onChange={(e) => handleChange('priority', e.target.value)}
        label="Приоритет"
      >
        <MenuItem value="">Все</MenuItem>
        <MenuItem value="Low">Низкий</MenuItem>
        <MenuItem value="Medium">Средний</MenuItem>
        <MenuItem value="High">Высокий</MenuItem>
      </Select>
```

```
        </FormControl>

        <FormControl size="small" sx={{ minWidth: 150 }}>
          <InputLabel>Проект</InputLabel>
          <Select
            value={localFilters.project_id || ''}
            onChange={(e) => handleChange('project_id', e.target.value)}
            label="Проект"
          >
            <MenuItem value="">Все проекты</MenuItem>
            {projects.map((project) => (
              <MenuItem key={project.id} value={project.id}>
                {project.name}
              </MenuItem>
            ))}
          </Select>
        </FormControl>

        <Button variant="contained" onClick={handleApply}>
          Применить
        </Button>

        <IconButton onClick={handleClear} size="small">
          <ClearIcon />
        </IconButton>
      </Box>

      {/* Активные фильтры */}
      {(localFilters.status || localFilters.priority || localFilters.project_id ||
localFilters.search) && (
        <Box display="flex" flexWrap="wrap" gap={1} mt={2}>
          {localFilters.search && (
            <Chip
              label={`Поиск: ${localFilters.search}`}
              onDelete={() => handleChange('search', '')}
```

```
                size="small"
              />
            )}
            {localFilters.status && (
              <Chip
                label={`Статус: ${
                  { New: 'Новые', InProgress: 'В работе', Completed:
'Выполненные' }[localFilters.status]
                }`}
                onDelete={() => handleChange('status', '')}
                size="small"
              />
            )}
            {localFilters.priority && (
              <Chip
                label={`Приоритет: ${
                  { Low: 'Низкий', Medium: 'Средний', High: 'Высокий' }[localFilters.priority]
                }`}
                onDelete={() => handleChange('priority', '')}
                size="small"
              />
            )}
            {localFilters.project_id && (
              <Chip
                label={`Проект: ${projects.find(p => p.id === localFilters.project_id)?.name || ''}`}
                onDelete={() => handleChange('project_id', '')}
                size="small"
              />
            )}
          </Box>
        )}
      </Paper>
    );
  };
```

```
export default TaskFilter;
```

Шаг 9. Создание компонента Navbar

src/components/common/Navbar.jsx:

javascript

```
import React, { useState } from 'react';
import {
  AppBar,
  Toolbar,
  Typography,
  Button,
  IconButton,
  Box,
  Menu,
  MenuItem,
  Avatar,
  useMediaQuery,
  Drawer,
  List,
  ListItem,
  ListItemIcon,
  ListItemText,
} from '@mui/material';
import {
  Menu as MenuIcon,
  Dashboard as DashboardIcon,
  Assignment as TasksIcon,
  BarChart as StatsIcon,
  Person as ProfileIcon,
  Logout as LogoutIcon,
} from '@mui/icons-material';
import { Link, useNavigate } from 'react-router-dom';

const Navbar = ({ user, onLogout }) => {
```

```
const navigate = useNavigate();
const isMobile = useMediaQuery('(max-width:600px)');
const [drawerOpen, setDrawerOpen] = useState(false);
const [anchorEl, setAnchorEl] = useState(null);

const menuItems = [
  { text: 'Панель управления', icon: <DashboardIcon />, path: '/' },
  { text: 'Задачи', icon: <TasksIcon />, path: '/tasks' },
  { text: 'Статистика', icon: <StatsIcon />, path: '/statistics' },
];

const handleMenuOpen = (event) => {
  setAnchorEl(event.currentTarget);
};

const handleMenuClose = () => {
  setAnchorEl(null);
};

const handleLogout = () => {
  handleMenuClose();
  onLogout?.();
};

const handleProfile = () => {
  handleMenuClose();
  navigate('/profile');
};

const drawer = (
  <Box sx={{ width: 250 }} role="presentation" onClick={() => setDrawerOpen(false)}>
    <List>
      {menuItems.map((item) => (
        <ListItem component={Link} to={item.path} key={item.text}>
          <ListItemIcon>{item.icon}</ListItemIcon>
```

```
                        <ListItemText primary={item.text} />
                    </ListItem>
                ))}
            </List>
        </Box>
    );

    return (
        <>
            <AppBar position="static">
                <Toolbar>
                    {isMobile && (
                        <IconButton
                            edge="start"
                            color="inherit"
                            onClick={() => setDrawerOpen(true)}
                            sx={{ mr: 2 }}
                        >
                            <MenuIcon />
                        </IconButton>
                    )}

                    <Typography variant="h6" component={Link} to="/" sx={{ flexGrow: 1,
textDecoration: 'none', color: 'white' }}>
                        Умный ежедневник
                    </Typography>

                    {!isMobile && (
                        <Box sx={{ display: 'flex', gap: 2 }}>
                            {menuItems.map((item) => (
                                <Button color="inherit" component={Link} to={item.path} key={item.text}>
                                    {item.text}
                                </Button>
                            ))}
                        </Box>
```

```
                    )}

                    {user && (
                      <Box>
                        <IconButton onClick={handleMenuOpen} sx={{ ml: 2 }}>
                          <Avatar sx={{ bgcolor: '#2ecc71' }}>
                            {user.username?.charAt(0).toUpperCase()}
                          </Avatar>
                        </IconButton>
                        <Menu anchorEl={anchorEl} open={Boolean(anchorEl)}
onClose={handleMenuClose}>
                          <MenuItem onClick={handleProfile}>
                            <ListItemIcon><ProfileIcon fontSize="small" /></ListItemIcon>
                            <ListItemText>Профиль</ListItemText>
                          </MenuItem>
                          <MenuItem onClick={handleLogout}>
                            <ListItemIcon><LogoutIcon fontSize="small" /></ListItemIcon>
                            <ListItemText>Выйти</ListItemText>
                          </MenuItem>
                        </Menu>
                      </Box>
                    )}
                  </Toolbar>
                </AppBar>

                <Drawer anchor="left" open={drawerOpen} onClose={() => setDrawerOpen(false)}>
                  {drawer}
                </Drawer>
              </>
            );
          };

          export default Navbar;
```

Шаг 10. Создание главной страницы Dashboard

src/pages/Dashboard.jsx:

```jsx
import React from 'react';
import { Container, Grid, Paper, Typography, Box } from '@mui/material';
import { useTasks } from '../hooks/useTasks';
import TaskList from '../components/tasks/TaskList';

const Dashboard = () => {
  const {
    tasks,
    loading,
    error,
    pagination,
    filters,
    changePage,
    changeFilters,
    updateTask,
    deleteTask
  } = useTasks({ status: 'InProgress,New' });

  const handleStatusChange = async (taskId, newStatus) => {
    await updateTask(taskId, { status: newStatus });
  };

  return (
    <Container maxWidth="lg" sx={{ mt: 4, mb: 4 }}>
      <Typography variant="h4" gutterBottom>
        Панель управления
      </Typography>

      <Grid container spacing={3}>
        {/* Статистика (будет добавлена позже) */}
        <Grid item xs={12} md={4}>
          <Paper sx={{ p: 2 }}>
            <Typography variant="h6">Активные задачи</Typography>
            <Typography variant="h3">
              {tasks.filter(t => t.status !== 'Completed').length}
```

```
            </Typography>
          </Paper>
        </Grid>

        {/* Список задач */}
        <Grid item xs={12}>
          <Paper sx={{ p: 2 }}>
            <Typography variant="h6" gutterBottom>
              Мои задачи
            </Typography>
            <TaskList
              tasks={tasks}
              loading={loading}
              error={error}
              pagination={pagination}
              filters={filters}
              onPageChange={changePage}
              onFilterChange={changeFilters}
              onStatusChange={handleStatusChange}
              onDelete={deleteTask}
            />
          </Paper>
        </Grid>
      </Grid>
    </Container>
  );
};

export default Dashboard;
```

Шаг 11. Настройка маршрутизации в App.jsx

src/App.jsx:

```
import React from 'react';
import { BrowserRouter, Routes, Route, Navigate } from 'react-router-dom';
import { ThemeProvider, createTheme } from '@mui/material/styles';
```

```
import CssBaseline from '@mui/material/CssBaseline';
import { AuthProvider, useAuth } from './context/AuthContext';
import Navbar from './components/common/Navbar';
import Dashboard from './pages/Dashboard';
import TaskListPage from './pages/TaskListPage';
import TaskCreatePage from './pages/TaskCreatePage';
import TaskEditPage from './pages/TaskEditPage';
import TaskDetailPage from './pages/TaskDetailPage';
import StatisticsPage from './pages/StatisticsPage';
import ProfilePage from './pages/ProfilePage';
import LoginForm from './components/LoginForm';
import RegisterForm from './components/RegisterForm';

const theme = createTheme({
  palette: {
    primary: {
      main: '#2c3e50',
    },
    secondary: {
      main: '#3498db',
    },
    success: {
      main: '#2ecc71',
    },
    warning: {
      main: '#f39c12',
    },
    error: {
      main: '#e74c3c',
    },
  },
});

const PrivateRoute = ({ children }) => {
  const { isAuthenticated, loading } = useAuth();
```

```
  if (loading) return <div>Загрузка...</div>;
  return isAuthenticated ? children : <Navigate to="/login" />;
};

const PublicRoute = ({ children }) => {
  const { isAuthenticated, loading } = useAuth();
  if (loading) return <div>Загрузка...</div>;
  return !isAuthenticated ? children : <Navigate to="/" />;
};

const AppContent = () => {
  const { user, logout, isAuthenticated } = useAuth();

  return (
    <>
      {isAuthenticated && <Navbar user={user} onLogout={logout} />}
      <Routes>
        <Route path="/login" element={
          <PublicRoute>
            <LoginForm />
          </PublicRoute>
        } />
        <Route path="/register" element={
          <PublicRoute>
            <RegisterForm />
          </PublicRoute>
        } />
        <Route path="/" element={
          <PrivateRoute>
            <Dashboard />
          </PrivateRoute>
        } />
        <Route path="/tasks" element={
          <PrivateRoute>
            <TaskListPage />
```

```
            </PrivateRoute>
          } />
          <Route path="/tasks/create" element={
            <PrivateRoute>
              <TaskCreatePage />
            </PrivateRoute>
          } />
          <Route path="/tasks/:id/edit" element={
            <PrivateRoute>
              <TaskEditPage />
            </PrivateRoute>
          } />
          <Route path="/tasks/:id" element={
            <PrivateRoute>
              <TaskDetailPage />
            </PrivateRoute>
          } />
          <Route path="/statistics" element={
            <PrivateRoute>
              <StatisticsPage />
            </PrivateRoute>
          } />
          <Route path="/profile" element={
            <PrivateRoute>
              <ProfilePage />
            </PrivateRoute>
          } />
        </Routes>
      </>
    );
};

function App() {
  return (
    <ThemeProvider theme={theme}>
```

```
            <CssBaseline />
            <BrowserRouter>
              <AuthProvider>
                <AppContent />
              </AuthProvider>
            </BrowserRouter>
          </ThemeProvider>
    );
}

export default App
```

Контрольные вопросы

1. Что такое React и в чем его основные преимущества?
2. Чем отличаются функциональные компоненты от классовых?
3. Что такое JSX и какие у него особенности?
4. Как работает маршрутизация в React Router?
5. Для чего нужны хуки useState и useEffect?
6. В чем разница между props и state?
7. Как настроить axios для работы с API?

Критерии оценки

Критерий	Баллы
Создан React-проект с правильной структурой	1
Настроена маршрутизация (React Router)	2
Реализованы основные компоненты страниц	2
Реализованы переиспользуемые компоненты (TaskCard, TaskFilter)	2

Критерий	Баллы
Настроен HTTP-клиент (axios)	2
Кастомный хук для работы с задачами	1
Итого	10

Результат выполнения

После выполнения лабораторной работы №7 должны получить:

1. Работающее React-приложение с настроенной маршрутизацией
2. Набор компонентов для отображения и управления задачами
3. Настроенный HTTP-клиент для взаимодействия с бэкендом
4. Адаптивный интерфейс с использованием Material-UI

Лабораторная работа №8

Управление состоянием и интеграция с API

Цель работы: Реализация полноценного взаимодействия React-приложения с бэкенд-API через управление состоянием с использованием Redux Toolkit.

Задачи

№	Задача
1	Настроить Redux Toolkit для управления глобальным состоянием
2	Реализовать хранилище (store) и слайсы (slices) для задач, проектов, тегов и аутентификации
3	Реализовать CRUD операции на фронтенде с вызовом API через

№	Задача
	асинхронные действия (createAsyncThunk)
4	Реализовать фильтрацию и поиск задач на фронтенде с синхронизацией с URL
5	Реализовать отображение уведомлений (toast notifications) об успехе/ошибке операций

Теоретическая часть

Управление состоянием в React

Проблема: В больших приложениях состояние может быть разбросано по множеству компонентов, что затрудняет его отслеживание и синхронизацию.

Решения:

1. Context API - для простых случаев
2. Redux / Redux Toolkit - для сложных приложений
3. Zustand - современная альтернатива Redux

Redux Toolkit (RTK)

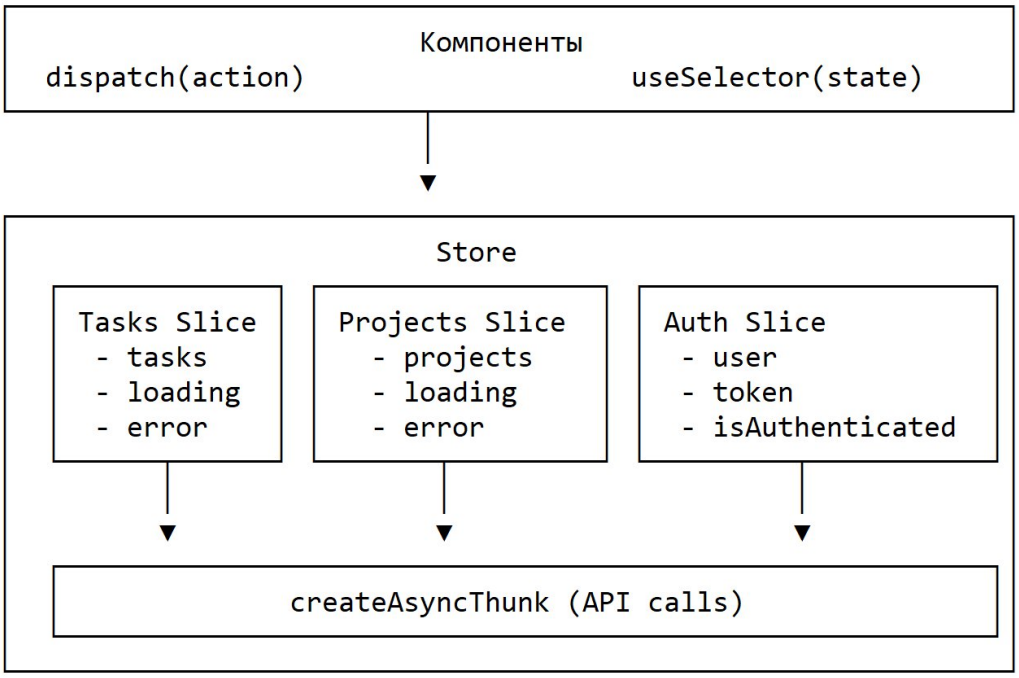
Redux Toolkit - официальный, рекомендуемый способ написания Redux-логики.

Основные понятия:

Понятие	Описание
Store	Единый объект, содержащий всё состояние приложения
Slice	Часть состояния со своими редьюсерами и действиями

Понятие	Описание
Reducer	Функция, которая принимает текущее состояние и действие, возвращает новое состояние
Action	Объект, описывающий изменение состояния
createAsyncThunk	Функция для обработки асинхронных операций (API-запросов)

Архитектура Redux в приложении



Ход выполнения работы

Шаг 1. Установка необходимых пакетов

```
# Установка Redux Toolkit и React-Redux
npm install @reduxjs/toolkit react-redux

# Установка для уведомлений
npm install react-hot-toast
```

```
# Установка для работы с query параметрами URL
npm install react-router-dom
```

Шаг 2. Создание хранилища (store)

src/store/store.js:

```
import { configureStore } from '@reduxjs/toolkit';
import tasksReducer from './slices/tasksSlice';
import projectsReducer from './slices/projectsSlice';
import tagsReducer from './slices/tagsSlice';
import authReducer from './slices/authSlice';
import uiReducer from './slices/uiSlice';

export const store = configureStore({
  reducer: {
    tasks: tasksReducer,
    projects: projectsReducer,
    tags: tagsReducer,
    auth: authReducer,
    ui: uiSlice
  },
  middleware: (getDefaultMiddleware) =>
    getDefaultMiddleware({
      serializableCheck: false,
    }),
});

export default store;
```

Шаг 3. Создание слайса для аутентификации

src/store/slices/authSlice.js:

```
import { createSlice, createAsyncThunk } from '@reduxjs/toolkit';
import apiClient from '../../api/client';

// Асинхронные действия
export const login = createAsyncThunk(
  'auth/login',
  async ({ username, password }, { rejectWithValue }) => {
    try {
      const response = await apiClient.post('/auth/login/', { username, password });
      const { access, refresh, user } = response.data;
      localStorage.setItem('access_token', access);
      localStorage.setItem('refresh_token', refresh);
      localStorage.setItem('user', JSON.stringify(user));
      return { user, access, refresh };
    } catch (error) {
      return rejectWithValue(error.response?.data?.detail || 'Ошибка входа');
    }
  }
);

export const register = createAsyncThunk(
  'auth/register',
  async ({ username, email, password, password2 }, { rejectWithValue }) => {
    try {
      const response = await apiClient.post('/auth/register/', {
        username,
        email,
        password,
        password2,
      });
      return response.data;
    } catch (error) {
```

```
            return rejectWithValue(error.response?.data || 'Ошибка регистрации');
        }
    }
);

export const logout = createAsyncThunk('auth/logout', async () => {
    localStorage.removeItem('access_token');
    localStorage.removeItem('refresh_token');
    localStorage.removeItem('user');
    return null;
});

export const checkAuth = createAsyncThunk('auth/checkAuth', async () => {
    const token = localStorage.getItem('access_token');
    const userStr = localStorage.getItem('user');

    if (token && userStr) {
        try {
            const user = JSON.parse(userStr);
            return { user, token };
        } catch {
            return null;
        }
    }
    return null;
});

// Начальное состояние
const initialState = {
    user: null,
    accessToken: null,
    refreshToken: null,
```

```
  isAuthenticated: false,
  isLoading: false,
  error: null,
};

// Слайс
const authSlice = createSlice({
  name: 'auth',
  initialState
  reducers: {
    clearError: (state) => {
      state error = null;
    },
  },
  extraReducers: (builder) => {
    builder
      // Login
      .addCase(login pending, (state) => {
        state isLoading = true;
        state error = null;
      })
      .addCase(login fulfilled, (state, action) => {
        state isLoading = false;
        state isAuthenticated = true;
        state user = action payload user;
        state accessToken = action payload access;
        state refreshToken = action payload refresh;
      })
      .addCase(login rejected, (state, action) => {
        state isLoading = false;
        state error = action payload
      })
```

```
      // Register
      .addCase(register.pending, (state) => {
        state.isLoading = true;
        state.error = null;
      })
      .addCase(register.fulfilled, (state) => {
        state.isLoading = false;
      })
      .addCase(register.rejected, (state, action) => {
        state.isLoading = false;
        state.error = action.payload;
      })
      // Logout
      .addCase(logout.fulfilled, (state) => {
        state.user = null;
        state.accessToken = null;
        state.refreshToken = null;
        state.isAuthenticated = false;
      })
      // Check Auth
      .addCase(checkAuth.fulfilled, (state, action) => {
        if (action.payload) {
          state.user = action.payload.user;
          state.accessToken = action.payload.token;
          state.isAuthenticated = true;
        }
      });
  },
});

export const { clearError } = authSlice.actions;
export default authSlice.reducer;
```

Шаг 4. Создание слайса для задач

src/store/slices/tasksSlice.js:

```
import { createSlice, createAsyncThunk } from '@reduxjs/toolkit';
import apiClient from '../../api/client';

// Асинхронные действия
export const fetchTasks = createAsyncThunk(
  'tasks/fetchTasks',
  async ({ page = 1, pageSize = 10, filters = {} }, { rejectWithValue }) => {
    try {
      const params = { page, page_size: pageSize, ...filters };
      const response = await apiClient.get('/tasks/', { params });
      return response.data;
    } catch (error) {
      return rejectWithValue(error.response?.data?.detail || 'Ошибка загрузки задач');
    }
  }
);

export const fetchTaskById = createAsyncThunk(
  'tasks/fetchTaskById',
  async (taskId, { rejectWithValue }) => {
    try {
      const response = await apiClient.get(`/tasks/${taskId}/`);
      return response.data;
    } catch (error) {
      return rejectWithValue(error.response?.data?.detail || 'Задача не найдена');
    }
  }
);
```

```
export const createTask = createAsyncThunk(
  'tasks/createTask',
  async (taskData, { rejectWithValue }) => {
    try {
      const response = await apiClient.post('/tasks/', taskData);
      return response.data;
    } catch (error) {
      return rejectWithValue(error.response?.data || 'Ошибка создания задачи');
    }
  }
);

export const updateTask = createAsyncThunk(
  'tasks/updateTask',
  async ({ id, data }, { rejectWithValue }) => {
    try {
      const response = await apiClient.put(`/tasks/${id}/`, data);
      return response.data;
    } catch (error) {
      return rejectWithValue(error.response?.data || 'Ошибка обновления задачи');
    }
  }
);

export const deleteTask = createAsyncThunk(
  'tasks/deleteTask',
  async (id, { rejectWithValue }) => {
    try {
      await apiClient.delete(`/tasks/${id}/`);
      return id;
    } catch (error) {
```

```
            return rejectWithValue(error.response?.data?.detail || 'Ошибка удаления задачи');
        }
    }
);

export const updateTaskStatus = createAsyncThunk(
    'tasks/updateTaskStatus',
    async ({ id, status }, { rejectWithValue }) => {
        try {
            const response = await apiClient.patch(`/tasks/${id}/`, { status });
            return response.data;
        } catch (error) {
            return rejectWithValue(error.response?.data?.detail || 'Ошибка обновления статуса');
        }
    }
);

// Начальное состояние
const initialState = {
    items: [],
    currentTask: null,
    pagination: {
        page: 1,
        pageSize: 10,
        total: 0,
        totalPages: 0,
    },
    filters: {
        status: '',
        priority: '',
        project_id: '',
        search: '',
```

```
  },
  isLoading: false,
  error: null,
};

// Слайс
const tasksSlice = createSlice({
  name: 'tasks',
  initialState
  reducers: {
    setPage: (state, action) => {
      state pagination page = action payload
    },
    setPageSize: (state, action) => {
      state pagination pageSize = action payload;
      state pagination page = 1;
    },
    setFilters: (state, action) => {
      state filters = { ...state filters, ...action payload };
      state pagination page = 1;
    },
    clearFilters: (state) => {
      state filters = initialState filters;
      state pagination page = 1;
    },
    clearError: (state) => {
      state error = null;
    },
    clearCurrentTask: (state) => {
      state currentTask = null;
    },
  },
```

```
extraReducers: (builder) => {
  builder
    // Fetch Tasks
    .addCase(fetchTasks.pending, (state) => {
      state.isLoading = true;
      state.error = null;
    })
    .addCase(fetchTasks.fulfilled, (state, action) => {
      state.isLoading = false;
      state.items = action.payload.tasks || [];
      state.pagination = {
        page: action.payload.page || 1,
        pageSize: action.payload.page_size || 10,
        total: action.payload.total || 0,
        totalPages: action.payload.total_pages || 0,
      };
    })
    .addCase(fetchTasks.rejected, (state, action) => {
      state.isLoading = false;
      state.error = action.payload;
    })
    // Fetch Task By ID
    .addCase(fetchTaskById.pending, (state) => {
      state.isLoading = true;
      state.error = null;
    })
    .addCase(fetchTaskById.fulfilled, (state, action) => {
      state.isLoading = false;
      state.currentTask = action.payload;
    })
    .addCase(fetchTaskById.rejected, (state, action) => {
      state.isLoading = false;
```

```
    state.error = action.payload;
  })
  // Create Task
  .addCase(createTask.fulfilled, (state, action) => {
    state.items.unshift(action.payload);
    state.pagination.total += 1;
  })
  // Update Task
  .addCase(updateTask.fulfilled, (state, action) => {
    const index = state.items.findIndex(t => t.id === action.payload.id);
    if (index !== -1) {
      state.items[index] = action.payload;
    }
    if (state.currentTask?.id === action.payload.id) {
      state.currentTask = action.payload
    }
  })
  // Delete Task
  .addCase(deleteTask.fulfilled, (state, action) => {
    state.items = state.items.filter(t => t.id !== action.payload);
    state.pagination.total -= 1;
    if (state.currentTask?.id === action.payload) {
      state.currentTask = null;
    }
  })
  // Update Task Status
  .addCase(updateTaskStatus.fulfilled, (state, action) => {
    const index = state.items.findIndex(t => t.id === action.payload.id);
    if (index !== -1) {
      state.items[index] = action.payload;
    }
  });
```

```
  },
});

export const { setPage, setPageSize, setFilters, clearFilters, clearError, clearCurrentTask } =
tasksSlice actions
export default tasksSlice reducer;
```

Шаг 5. Создание слайса для проектов

src/store/slices/projectsSlice.js:

```
import { createSlice, createAsyncThunk } from '@reduxjs/toolkit';
import apiClient from '../../api/client';

export const fetchProjects = createAsyncThunk(
  'projects/fetchProjects',
  async (_, { rejectWithValue }) => {
    try {
      const response = await apiClient.get('/projects/');
      return response.data;
    } catch (error) {
      return rejectWithValue(error.response?.data?.detail || 'Ошибка загрузки проектов');
    }
  }
);

export const createProject = createAsyncThunk(
  'projects/createProject',
  async (projectData, { rejectWithValue }) => {
    try {
      const response = await apiClient.post('/projects/', projectData);
      return response.data;
    } catch (error) {
      return rejectWithValue(error.response?.data || 'Ошибка создания проекта');
```

```
    }
  }
);

export const updateProject = createAsyncThunk(
  'projects/updateProject',
  async ({ id, data }, { rejectWithValue }) => {
    try {
      const response = await apiClient.put(`/projects/${id}/`, data);
      return response.data;
    } catch (error) {
      return rejectWithValue(error.response?.data || 'Ошибка обновления проекта');
    }
  }
);

export const deleteProject = createAsyncThunk(
  'projects/deleteProject',
  async (id, { rejectWithValue }) => {
    try {
      await apiClient.delete(`/projects/${id}/`);
      return id;
    } catch (error) {
      return rejectWithValue(error.response?.data?.detail || 'Ошибка удаления проекта');
    }
  }
);

const initialState = {
  items: [],
  isLoading: false,
  error: null,
```

```
};

const projectsSlice = createSlice({
  name: 'projects',
  initialState,
  reducers: {
    clearError: (state) => {
      state.error = null;
    },
  },
  extraReducers: (builder) => {
    builder
      .addCase(fetchProjects.pending, (state) => {
        state.isLoading = true;
        state.error = null;
      })
      .addCase(fetchProjects.fulfilled, (state, action) => {
        state.isLoading = false;
        state.items = action.payload;
      })
      .addCase(fetchProjects.rejected, (state, action) => {
        state.isLoading = false;
        state.error = action.payload;
      })
      .addCase(createProject.fulfilled, (state, action) => {
        state.items.push(action.payload);
      })
      .addCase(updateProject.fulfilled, (state, action) => {
        const index = state.items.findIndex(p => p.id === action.payload.id);
        if (index !== -1) {
          state.items[index] = action.payload;
        }
```

```
      })
      .addCase(deleteProject.fulfilled, (state, action) => {
        state.items = state.items.filter(p => p.id !== action.payload);
      });
  },
});

export const { clearError } = projectsSlice.actions;
export default projectsSlice.reducer;
```

Шаг 6. Создание слайса для тегов

src/store/slices/tagsSlice.js:

```
import { createSlice, createAsyncThunk } from '@reduxjs/toolkit';
import apiClient from '../../api/client';

export const fetchTags = createAsyncThunk(
  'tags/fetchTags',
  async (_, { rejectWithValue }) => {
    try {
      const response = await apiClient.get('/tags/');
      return response.data;
    } catch (error) {
      return rejectWithValue(error.response?.data?.detail || 'Ошибка загрузки тегов');
    }
  }
);

export const createTag = createAsyncThunk(
  'tags/createTag',
  async (tagData, { rejectWithValue }) => {
    try {
```

```
            const response = await apiClient.post('/tags/', tagData);
            return response.data;
        } catch (error) {
            return rejectWithValue(error.response?.data || 'Ошибка создания тега');
        }
    }
);

export const deleteTag = createAsyncThunk(
    'tags/deleteTag',
    async (id, { rejectWithValue }) => {
        try {
            await apiClient.delete(`/tags/${id}/`);
            return id;
        } catch (error) {
            return rejectWithValue(error.response?.data?.detail || 'Ошибка удаления тега');
        }
    }
);

const initialState = {
    items: [],
    isLoading: false,
    error: null,
};

const tagsSlice = createSlice({
    name: 'tags',
    initialState,
    reducers: {
        clearError: (state) => {
            state.error = null;
```

```
    },
  },
  extraReducers: (builder) => {
    builder
      .addCase(fetchTags.pending, (state) => {
        state.isLoading = true;
        state.error = null;
      })
      .addCase(fetchTags.fulfilled, (state, action) => {
        state.isLoading = false;
        state.items = action.payload;
      })
      .addCase(fetchTags.rejected, (state, action) => {
        state.isLoading = false;
        state.error = action.payload;
      })
      .addCase(createTag.fulfilled, (state, action) => {
        state.items.push(action.payload);
      })
      .addCase(deleteTag.fulfilled, (state, action) => {
        state.items = state.items.filter(t => t.id !== action.payload);
      });
  },
});

export const { clearError } = tagsSlice.actions;
export default tagsSlice.reducer;
```

Шаг 7. Создание слайса для UI (уведомления)

src/store/slices/uiSlice.js:

```
import { createSlice } from '@reduxjs/toolkit';
```

```
const initialState = {
  notifications: [],
  sidebarOpen: true,
  theme: 'light',
};

const uiSlice = createSlice({
  name: 'ui',
  initialState,
  reducers: {
    addNotification: (state, action) => {
      const notification = {
        id: Date.now(),
        type: action.payload.type || 'info',
        message: action.payload.message,
        timestamp: new Date().toISOString(),
      };
      state.notifications.push(notification);

      // Автоматическое удаление через 5 секунд
      setTimeout(() => {
        state.notifications = state.notifications.filter(n => n.id !== notification.id);
      }, 5000);
    },
    removeNotification: (state, action) => {
      state.notifications = state.notifications.filter(n => n.id !== action.payload);
    },
    clearNotifications: (state) => {
      state.notifications = [];
    },
    toggleSidebar: (state) => {
```

```
      state.sidebarOpen = !state.sidebarOpen;
    },
    setTheme: (state, action) => {
      state.theme = action.payload;
      localStorage.setItem('theme', action.payload);
    },
  },
});

export const { addNotification, removeNotification, clearNotifications, toggleSidebar, setTheme } = uiSlice.actions;
export default uiSlice.reducer;
```

Шаг 8. Создание хуков для удобного использования Redux

src/store/hooks.js:

```
import { useDispatch, useSelector } from 'react-redux';

export const useAppDispatch = () => useDispatch();
export const useAppSelector = useSelector;
```

Шаг 9. Подключение Store к приложению

src/main.jsx:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import { Provider } from 'react-redux';
import { Toaster } from 'react-hot-toast';
import { store } from './store/store';
import App from './App';
import './styles/global.css';

ReactDOM.createRoot(document.getElementById('root')).render(
```

```
    <React StrictMode>
      <Provider store={store}>
        <Toaster position="top-right" />
        <App />
      </Provider>
    </React StrictMode>
);
```

Шаг 10. Компонент TaskList с использованием Redux

src/components/tasks/TaskListRedux.jsx:

```
import React, { useEffect } from 'react';
import { useAppDispatch, useAppSelector } from '../../store/hooks';
import {
  fetchTasks,
  updateTaskStatus,
  deleteTask,
  setPage,
  setFilters,
} from '../../store/slices/tasksSlice';
import { addNotification } from '../../store/slices/uiSlice';
import TaskCard from './TaskCard';
import TaskFilter from './TaskFilter';
import { Box, Typography, CircularProgress, Alert, Pagination } from '@mui/material';

const TaskListRedux = () => {
  const dispatch = useAppDispatch();
  const { items: tasks, pagination, filters, isLoading, error } = useAppSelector(
    (state) => state.tasks
  );

  useEffect(() => {
```

```
    dispatch(fetchTasks({
      page: pagination.page,
      pageSize: pagination.pageSize,
      filters,
    }));
  }, [dispatch, pagination.page, pagination.pageSize, filters]);

  const handleStatusChange = async (taskId, newStatus) => {
    const result = await dispatch(updateTaskStatus({ id: taskId, status: newStatus }));
    if (result.meta.requestStatus === 'fulfilled') {
      dispatch(addNotification({
        type: 'success',
        message: 'Статус задачи обновлен',
      }));
      dispatch(fetchTasks({
        page: pagination.page,
        pageSize: pagination.pageSize,
        filters,
      }));
    } else {
      dispatch(addNotification({
        type: 'error',
        message: result.payload || 'Ошибка обновления статуса',
      }));
    }
  };

  const handleDelete = async (taskId) => {
    const result = await dispatch(deleteTask(taskId));
    if (result.meta.requestStatus === 'fulfilled') {
      dispatch(addNotification({
        type: 'success',
```

```
            message: 'Задача удалена',
        }));
    } else {
        dispatch(addNotification({
            type: 'error',
            message: result.payload || 'Ошибка удаления задачи',
        }));
    }
};

const handlePageChange = (newPage) => {
    dispatch(setPage(newPage));
};

const handleFilterChange = (newFilters) => {
    dispatch(setFilters(newFilters));
};

if (isLoading && tasks.length === 0) {
    return (
        <Box display="flex" justifyContent="center" alignItems="center" minHeight="200px">
            <CircularProgress />
        </Box>
    );
}

if (error) {
    return (
        <Alert severity="error" sx={{ m: 2 }}>
            {error}
        </Alert>
    );
```

```
}

if (tasks.length === 0) {
  return (
    <Box textAlign="center" py={4}>
      <Typography variant="h6" color="text.secondary">
        Нет задач
      </Typography>
      <Typography variant="body2" color="text.secondary">
        Создайте первую задачу, нажав кнопку «Новая задача»
      </Typography>
    </Box>
  );
}

return (
  <Box>
    <TaskFilter filters={filters} onFilterChange={handleFilterChange} />

    <Box>
      {tasks.map((task) => (
        <TaskCard
          key={task.id}
          task={task}
          onStatusChange={handleStatusChange}
          onDelete={handleDelete}
        />
      ))}
    </Box>

    {pagination.totalPages > 1 && (
      <Box display="flex" justifyContent="center" mt={3}>
```

```
            <Pagination
              count={pagination.totalPages}
              page={pagination.page}
              onChange={(_, page) => handlePageChange(page)}
              color="primary"
            />
          </Box>
        )}
      </Box>
    );
};

export default TaskListRedux;
```

Шаг 11. Компонент TaskForm с использованием Redux

src/components/tasks/TaskForm.jsx:

```
import React, { useState, useEffect } from 'react';
import { useNavigate, useParams } from 'react-router-dom';
import { useFormik } from 'formik';
import * as Yup from 'yup';
import { useAppDispatch, useAppSelector } from '../../store/hooks';
import {
  createTask,
  updateTask,
  fetchTaskById,
  clearCurrentTask,
} from '../../store/slices/tasksSlice';
import { fetchProjects } from '../../store/slices/projectsSlice';
import { fetchTags } from '../../store/slices/tagsSlice';
import { addNotification } from '../../store/slices/uiSlice';
import {
```

```
    TextField,
    Button,
    Box,
    Paper,
    Typography,
    MenuItem,
    FormControl,
    InputLabel,
    Select,
    OutlinedInput,
    Chip,
    Grid,
} from '@mui/material';

const validationSchema = Yup.object({
    title: Yup.string()
        .min(3, 'Заголовок должен содержать минимум 3 символа')
        .max(200, 'Заголовок не может превышать 200 символов')
        .required('Заголовок обязателен'),
    description: Yup.string(),
    deadline: Yup.date()
        .nullable()
        .min(new Date(), 'Дедлайн не может быть в прошлом'),
    status: Yup.string().required('Статус обязателен'),
    priority: Yup.string().required('Приоритет обязателен'),
    project_id: Yup.number().nullable(),
    tag_ids: Yup.array(),
});

const statusOptions = [
    { value: 'New', label: 'Новая' },
    { value: 'InProgress', label: 'В работе' },
```

```
  { value: 'Completed', label: 'Выполнена' },
];

const priorityOptions = [
  { value: 'Low', label: 'Низкий' },
  { value: 'Medium', label: 'Средний' },
  { value: 'High', label: 'Высокий' },
];

const TaskForm = () => {
  const { id } = useParams();
  const navigate = useNavigate();
  const dispatch = useAppDispatch();
  const isEditMode = Boolean(id);

  const { currentTask, isLoading: taskLoading } = useAppSelector((state) => state.tasks);
  const { items: projects, isLoading: projectsLoading } = useAppSelector((state) => state.projects);
  const { items: tags, isLoading: tagsLoading } = useAppSelector((state) => state.tags);

  const [initialLoading, setInitialLoading] = useState(true);

  useEffect(() => {
    Promise.all([
      dispatch(fetchProjects()),
      dispatch(fetchTags()),
    ]).finally(() => {
      setInitialLoading(false);
    });

    if (isEditMode && id) {
      dispatch(fetchTaskById(parseInt(id)));
    }
```

```
    return () => {
      if (isEditMode) {
        dispatch(clearCurrentTask());
      }
    };
  }, [dispatch, id, isEditMode]);

  const formik = useFormik({
    initialValues: {
      title: currentTask?.title || '',
      description: currentTask?.description || '',
      deadline: currentTask?.deadline || '',
      status: currentTask?.status || 'New',
      priority: currentTask?.priority || 'Medium',
      project_id: currentTask?.project?.id || '',
      tag_ids: currentTask?.tags?.map(t => t.id) || [],
    },
    validationSchema: validationSchema,
    enableReinitialize: true,
    onSubmit: async (values) => {
      // Преобразуем пустые значения в null
      const submitData = {
        ...values,
        deadline: values.deadline || null,
        project_id: values.project_id || null,
      };

      let result;
      if (isEditMode) {
        result = await dispatch(updateTask({ id: parseInt(id), data: submitData }));
      } else {
```

```
            result = await dispatch(createTask(submitData));
        }

        if (result.meta.requestStatus === 'fulfilled') {
            dispatch(addNotification({
                type: 'success',
                message: isEditMode ? 'Задача обновлена' : 'Задача создана',
            }));
            navigate('/tasks');
        } else {
            dispatch(addNotification({
                type: 'error',
                message: result.payload || 'Ошибка сохранения задачи',
            }));
        }
    },
});

if (initialLoading || (isEditMode && taskLoading)) {
    return (
        <Box display="flex" justifyContent="center" alignItems="center" minHeight="400px">
            <CircularProgress />
        </Box>
    );
}

return (
    <Box maxWidth="md" mx="auto" mt={4}>
        <Paper elevation={3} sx={{ p: 4 }}>
            <Typography variant="h5" gutterBottom>
                {isEditMode ? 'Редактирование задачи' : 'Новая задача'}
            </Typography>
```

```
<form onSubmit={formik.handleSubmit}>
  <Grid container spacing={3}>
    <Grid item xs={12}>
      <TextField
        fullWidth
        name="title"
        label="Заголовок"
        value={formik.values.title}
        onChange={formik.handleChange}
        error={formik.touched.title && Boolean(formik.errors.title)}
        helperText={formik.touched.title && formik.errors.title}
        disabled={formik.isSubmitting}
      />
    </Grid>

    <Grid item xs={12}>
      <TextField
        fullWidth
        name="description"
        label="Описание"
        multiline
        rows={4}
        value={formik.values.description}
        onChange={formik.handleChange}
        disabled={formik.isSubmitting}
      />
    </Grid>

    <Grid item xs={12} sm={6}>
      <TextField
        fullWidth
```

```
        name="deadline"
        label="Дедлайн"
        type="datetime-local"
        value={formik.values.deadline}
        onChange={formik.handleChange}
        InputLabelProps={{ shrink: true }}
        error={formik.touched.deadline && Boolean(formik.errors.deadline)}
        helperText={formik.touched.deadline && formik.errors.deadline}
        disabled={formik.isSubmitting}
      />
    </Grid>

    <Grid item xs={12} sm={6}>
      <FormControl fullWidth disabled={formik.isSubmitting}>
        <InputLabel>Статус</InputLabel>
        <Select
          name="status"
          value={formik.values.status}
          onChange={formik.handleChange}
          label="Статус"
        >
          {statusOptions.map(option => (
            <MenuItem key={option.value} value={option.value}>
              {option.label}
            </MenuItem>
          ))}
        </Select>
      </FormControl>
    </Grid>

    <Grid item xs={12} sm={6}>
      <FormControl fullWidth disabled={formik.isSubmitting}>
```

```
        <InputLabel>Приоритет</InputLabel>
        <Select
          name="priority"
          value={formik.values.priority}
          onChange={formik.handleChange}
          label="Приоритет"
        >
          {priorityOptions.map(option => (
            <MenuItem key={option.value} value={option.value}>
              {option.label}
            </MenuItem>
          ))}
        </Select>
      </FormControl>
    </Grid>

    <Grid item xs={12} sm={6}>
      <FormControl fullWidth disabled={projectsLoading || formik.isSubmitting}>
        <InputLabel>Проект</InputLabel>
        <Select
          name="project_id"
          value={formik.values.project_id}
          onChange={formik.handleChange}
          label="Проект"
        >
          <MenuItem value="">Без проекта</MenuItem>
          {projects.map(project => (
            <MenuItem key={project.id} value={project.id}>
              {project.name}
            </MenuItem>
          ))}
        </Select>
```

```
    </FormControl>
  </Grid>

  <Grid item xs={12}>
    <FormControl fullWidth disabled={tagsLoading || formik isSubmitting}>
      <InputLabel>Теги</InputLabel>
      <Select
        multiple
        name="tag_ids"
        value={formik values tag_ids}
        onChange={formik handleChange}
        input={<OutlinedInput label="Теги" />}
        renderValue={(selected) => (
          <Box sx={{ display: 'flex', flexWrap: 'wrap', gap: 0.5 }}>
            {selected.map((value) => {
              const tag = tags.find(t => t.id === value);
              return (
                <Chip key={value} label={tag?.name || value} size="small" />
              );
            })}
          </Box>
        )}
      >
        {tags.map(tag => (
          <MenuItem key={tag.id} value={tag.id}>
            {tag.name}
          </MenuItem>
        ))}
      </Select>
    </FormControl>
  </Grid>
```

```
            <Grid item xs={12}>
              <Box display="flex" gap={2} justifyContent="flex-end">
                <Button
                  variant="outlined"
                  onClick={() => navigate('/tasks')}
                  disabled={formik.isSubmitting}
                >
                  Отмена
                </Button>
                <Button
                  type="submit"
                  variant="contained"
                  color="primary"
                  disabled={formik.isSubmitting}
                >
                  {formik.isSubmitting ? 'Сохранение...' : (isEditMode ? 'Сохранить' : 'Создать')}
                </Button>
              </Box>
            </Grid>
          </Grid>
        </form>
      </Paper>
    </Box>
  );
};

export default TaskForm;
```

Шаг 12. Обновление AuthContext для работы с Redux

src/context/AuthContext.jsx (обновленный):

```
import React, { createContext, useContext, useEffect } from 'react';
```

```
import { useAppDispatch, useAppSelector } from '../store/hooks';
import { login, register, logout, checkAuth, clearError } from '../store/slices/authSlice';
import { addNotification } from '../store/slices/uiSlice';

const AuthContext = createContext();

export const useAuth = () => useContext(AuthContext);

export const AuthProvider = ({ children }) => {
  const dispatch = useAppDispatch();
  const { user, isAuthenticated, isLoading, error } = useAppSelector((state) => state.auth);

  useEffect(() => {
    dispatch(checkAuth());
  }, [dispatch]);

  useEffect(() => {
    if (error) {
      dispatch(addNotification({ type: 'error', message: error }));
      dispatch(clearError());
    }
  }, [error, dispatch]);

  const handleLogin = async (username, password) => {
    const result = await dispatch(login({ username, password }));
    if (result.meta.requestStatus === 'fulfilled') {
      dispatch(addNotification({ type: 'success', message: 'Добро пожаловать!' }));
      return { success: true };
    }
    return { success: false, error: result.payload };
  };
```

```
  const handleRegister = async (username, email, password, password2) => {
    const result = await dispatch(register({ username, email, password, password2 }));
    if (result.meta.requestStatus === 'fulfilled') {
      dispatch(addNotification({ type: 'success', message: 'Регистрация прошла успешно! Теперь войдите в систему.' }));
      return { success: true };
    }
    return { success: false, error: result.payload };
  };

  const handleLogout = async () => {
    await dispatch(logout());
    dispatch(addNotification({ type: 'info', message: 'Вы вышли из системы' }));
  };

  const value = {
    user,
    isAuthenticated,
    isLoading,
    login: handleLogin,
    register: handleRegister,
    logout: handleLogout,
  };

  return (
    <AuthContext.Provider value={value}>
      {children}
    </AuthContext.Provider>
  );
};
```

Контрольные вопросы

1. Что такое Redux Toolkit и для чего он нужен?
2. Чем отличается Redux от Context API?
3. Что такое slice в Redux Toolkit?
4. Для чего используется createAsyncThunk?
5. Как работает интеграция Redux с React через react-redux?
6. Что такое селекторы и для чего они нужны?
7. Как организовать уведомления в Redux?

Критерии оценки

Критерий	Баллы
Настроен Redux Toolkit store	1
Реализованы слайсы для задач, проектов, тегов, аутентификации	2
Реализованы асинхронные действия (createAsyncThunk)	2
Компоненты используют Redux для получения данных	2
Реализована фильтрация с синхронизацией с Redux	1
Реализованы уведомления (toast)	1
Код работает без ошибок	1
Итого	10

Результат выполнения

После выполнения лабораторной работы №8 студенты должны получить:

Централизованное хранилище состояния (Redux store)

Асинхронные действия для работы с API

Полноценную интеграцию React-компонентов с Redux

Систему уведомлений об успехе/ошибке операций

Фильтрацию и пагинацию задач с хранением состояния в Redux

Лабораторная работа №9

Презентация и защита проекта

Цель работы: Систематизация и демонстрация результатов, полученных в первом семестре, защита технической документации, презентация работающего прототипа веб-приложения и планирование работ на второй семестр.

Задачи

№	Задача
1	Подготовить презентацию проекта (10-15 слайдов)
2	Подготовить техническую документацию (README, API, архитектура)
3	Продемонстрировать работающее приложение (бэкенд + базовый фронтенд)
4	Защитить принятые архитектурные решения
5	Спланировать работы на второй семестр с использованием инструментов управления задачами

Теоретическая часть

Структура презентации проекта

Раздел	Содержание	Рекомендуемое количество слайдов
Титул	Название проекта, команда, дата	1
Актуальность	Проблема, которую решает проект	1
Цели и задачи	Что сделано за семестр	1
Функциональные требования	Основные возможности	1
Архитектура	Диаграммы, технологический стек	2-3
Модель данных	ER-диаграмма, основные сущности	1
API	Основные эндпоинты, примеры запросов	1-2
Демонстрация	Скриншоты или живая демонстрация	2-3
Проблемы и решения	Трудности и как их преодолели	1
Планы на 2 семестр	Что будет добавлено	1
Заключение	Итоги, выводы	1

Раздел	Содержание	Рекомендуемое количество слайдов
Вопросы		1

Техническая документация

1. Минимальный состав документации:
2. README.md - общее описание проекта, установка, запуск
3. API документация - Swagger/OpenAPI (автоматическая)
4. Архитектурная документация - диаграммы, описание слоёв
5. Руководство пользователя - базовые сценарии работы

План работ на 2 семестр

Должен включать:

1. Функциональные возможности, которые будут добавлены
2. Технологии, которые будут освоены
3. Распределение задач между участниками команды
4. Оценка трудоёмкости (в часах или story points)
5. Ключевые вехи (deadline)

Ход выполнения работы

Шаг 1. Подготовка презентации проекта

Структура презентации (детализация)

Слайд 1: Титульный лист

УМНЫЙ ЕЖЕДНЕВНИК

Smart Task Manager

Междисциплинарный проектный практикум-2

Студент: [ФИО]

Группа: [номер]

Преподаватель: [ФИО]

Дата: [дата]

Слайд 2: Актуальность проекта

Проблема: Современный человек сталкивается с большим количеством задач,

которые необходимо планировать и контролировать.

Решение: Создание веб-приложения для управления задачами, проектами и

временем с возможностью:

- Категоризации задач по проектам и тегам
- Установки приоритетов и дедлайнов
- Отслеживания прогресса и статистики

Слайд 3: Цели и задачи 1 семестра

Цель: Разработать прототип веб-приложения «Умный ежедневник»

Выполненные задачи:

1. Спроектирована архитектура приложения (MVC)
2. Разработана база данных (5 сущностей, связи)
3. Реализовано REST API (CRUD для задач, проектов, тегов)
4. Реализована JWT-аутентификация
5. Создан базовый React-фронтенд
6. Написаны тесты (покрытие >70%)
7. Подготовлена документация API (Swagger)

Слайд 4: Функциональные требования

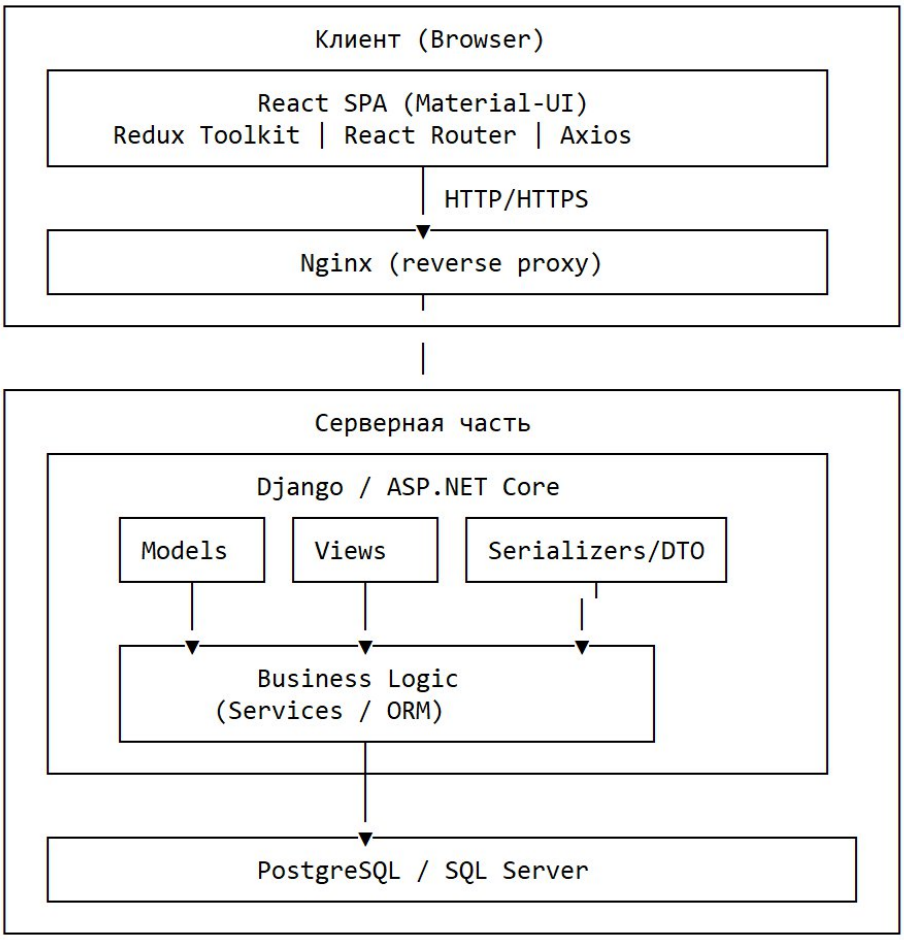
Пользовательские роли:

- Аноним: регистрация, вход
- Авторизованный: управление задачами, проектами, тегами
- Администратор: управление пользователями

Основной функционал:

1. CRUD операции с задачами
2. Фильтрация по статусу, приоритету, проекту
3. Полнотекстовый поиск
4. Статистика по задачам
5. Экспорт в CSV/JSON/PDF

Слайд 5: Архитектура приложения



Слайд 6: Технологический стек (пример для Django)

Backend:

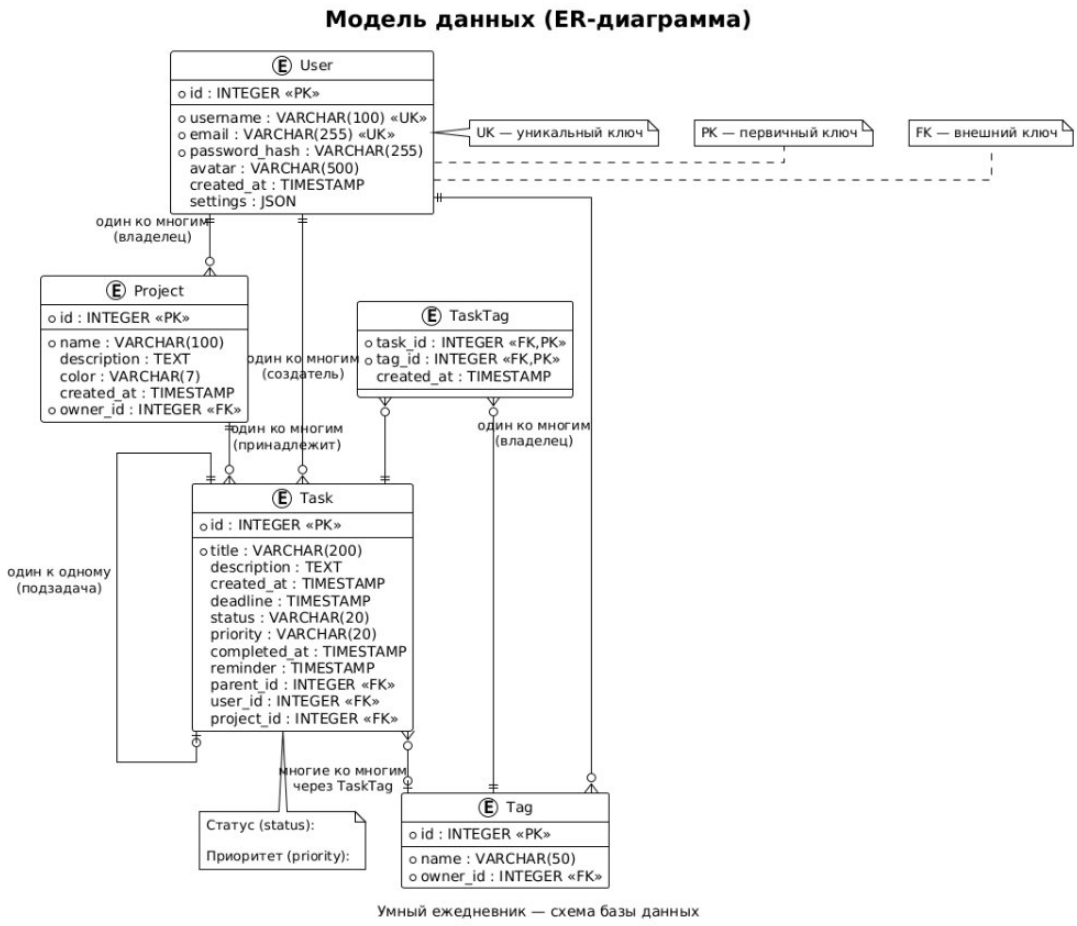
- Python 3.11
- Django 5.0
- Django REST Framework
- PostgreSQL

- JWT (djangorestframework-simplejwt)
- drf-spectacular (Swagger)

Frontend:
- React 18
- Vite
- Redux Toolkit
- React Router v6
- Axios
- Material-UI (MUI)
- Formik + Yup

DevOps:
- Git / GitHub
- Docker / Docker Compose
- GitHub Actions (CI/CD)

Слайд 7: Модель данных (ER-диаграмма)



Слайд 8: API эндпоинты

Метод	URL	Описание
POST	/api/auth/register/	Регистрация
POST	/api/auth/login/	Вход (JWT)
POST	/api/token/refresh/	Обновление токена
GET	/api/tasks/	Список задач (пагинация)
GET	/api/tasks/{id}/	Детали задачи
POST	/api/tasks/	Создание задачи
PUT	/api/tasks/{id}/	Обновление задачи
DELETE	/api/tasks/{id}/	Удаление задачи

PATCH	/api/tasks/{id}/	Частичное обновление
GET	/api/projects/	Список проектов
GET	/api/tags/	Список тегов
GET	/api/statistics/	Статистика
GET	/api/export/?format=csv	Экспорт задач

Слайд 9: Демонстрация (скриншоты)

```
[Скриншот 1: Страница входа]

  [Скриншот 2: Главная страница со списком задач]

[Скриншот 3: Форма создания задачи]
```

Слайд 10: Трудности и их решение

Трудность	Решение
Настройка CORS	Использование django-cors-headers
Обновление JWT-токена	Axios interceptors
N+1 проблема в ORM	select_related / prefetch_related
Пагинация на фронтенде	React Router + query params

Слайд 11: Планы развития

Функциональность:

- WebSocket уведомления о дедлайнах
- Совместная работа (шаринг задач)
- Интеграция с Google Calendar
- PWA (офлайн-режим)

- AI-рекомендации приоритетов
- Мобильная адаптация

Технологии:
- WebSockets (Django Channels / SignalR)
- Redis (кэш, очереди)
- Celery (фоновые задачи)
- Docker (контейнеризация)
- CI/CD (GitHub Actions)
- Prometheus + Grafana (мониторинг)

Слайд 12: Заключение

Достигнутые результаты:
- Разработан полностью функционирующий прототип
- Освоены технологии: Django/ASP.NET, React, Redux, Docker
- Сформированы навыки командной разработки
- Создана база для дальнейшего развития

Ключевые метрики:
- Количество строк кода: ~5000
- Количество коммитов: ~50
- Покрытие тестами: ~75%

Шаг 2. Оформление технической документации

README.md (пример):

markdown

Умный ежедневник (Smart Diary)

О проекте

Smart Diary — это веб-приложение для управления задачами, проектами и личной продуктивностью.

Основные возможности

- ✔ Управление задачами (CRUD)
- ✔ Категоризация по проектам и тегам
- ✔ Приоритеты и дедлайны
- ✔ Полнотекстовый поиск
- ✔ Статистика продуктивности
- ✔ Экспорт данных (CSV, JSON, PDF)
- ✔ JWT-аутентификация

Технологии

Backend
- Python 3.11 / Django 5.0 / DRF
- PostgreSQL
- JWT аутентификация
- Swagger/OpenAPI

Frontend
- React 18 / Vite
- Redux Toolkit
- Material-UI
- Axios

DevOps
- Git / GitHub

- Docker / Docker Compose

Установка и запуск

Требования

- Python 3.11+
- Node.js 18+
- PostgreSQL 15+

Backend

```bash
# Клонирование репозитория
git clone https://github.com/username/smart-diary.git
cd smart-diary/backend

# Создание виртуального окружения
python -m venv venv
source venv/bin/activate  # Linux/Mac
venv\Scripts\activate     # Windows

# Установка зависимостей
pip install -r requirements.txt

# Применение миграций
python manage.py migrate

# Создание суперпользователя
python manage.py createsuperuser
```

```
# Запуск сервера
python manage.py runserver
```

Frontend

bash

```
cd ./frontend
npm install
npm run dev
```

API Документация

После запуска бэкенда документация доступна по адресу:

Swagger UI: http://localhost:8000/api/schema/swagger-ui/

ReDoc: http://localhost:8000/api/schema/redoc/

Тестирование

bash

```
# Backend
pytest --cov=tasks --cov-report=html

# Frontend
npm test
```

Структура проекта

text

```
smart-diary/
├── backend/
│   ├── tasks/              # Основное приложение
│   │   ├── models.py       # Модели данных
│   │   ├── views/          # Представления (API)
│   │   ├── services/       # Бизнес-логика
│   │   ├── serializers.py  # Сериализаторы
│   │   └── tests/          # Тесты
│   ├── smart_diary/        # Настройки проекта
```

```
│   └── requirements.txt
├── frontend/
│   ├── src/
│   │   ├── components/     # React компоненты
│   │   ├── pages/          # Страницы
│   │   ├── store/          # Redux store
│   │   ├── api/            # API клиенты
│   │   └── hooks/          # Кастомные хуки
│   └── package.json
├── docker-compose.yml
└── README.md
```

Команда

[Имя] — Team Lead, Backend разработка

[Имя] — Frontend разработка

[Имя] — DevOps, тестирование

Планы на 2 семестр

WebSocket уведомления

Совместная работа (шаринг задач)

Интеграция с Google Calendar

PWA (офлайн-режим)

AI-рекомендации

text

Шаг 3. Подготовка к защите

Чек-лист подготовки:

№	Что проверить	Готово
1.	Презентация (10-15 слайдов)	+
2.	README.md заполнен	+
3.	Swagger доступен и аннотирован	+

4.	Все CRUD операции работают	+
5.	Аутентификация работает	+
6.	Фильтрация и поиск работают	+
7.	Статистика отображается корректно	+
8.	Тесты проходят	+
9.	Код закоммичен в репозиторий	+
10.	Репозиторий публичный или доступен преподавателю	+

Критерии оценки презентации

Критерий	Описание	Баллы
Содержание	Полнота раскрытия темы, логичность изложения	2
Качество слайдов	Визуальное оформление, читаемость, диаграммы	2
Демонстрация	Работающее приложение, наглядность	2
Владение материалом	Уверенность, ответы на вопросы	2
Документация	Полнота и качество README, API docs	2
Итого		10

Контрольные вопросы для защиты

1. Какой паттерн архитектуры использован в проекте и почему?

2. Какие технологии были выбраны и по каким критериям?
3. Как организована аутентификация и хранение паролей?
4. Как решалась проблема CORS при разработке?
5. Какие инструменты использовались для тестирования?
6. Как обеспечивается безопасность API?
7. Какие сложности возникли и как они были решены?
8. Что планируется добавить во 2 семестре?
9. Как распределены роли в команде?
10. Как организован процесс разработки (Git, задачи)?

Критерии оценки выполнения работы

Критерий	Баллы
Презентация (10-15 слайдов, логичная структура)	2
Техническая документация (README, Swagger)	2
Демонстрация работающего приложения	2
Ответы на контрольные вопросы	2
Итого	8

Результат выполнения

После выполнения лабораторной работы №9 студенты должны иметь:

1. Презентацию проекта (10-15 слайдов)

2. Техническую документацию (README, Swagger)

3. План работ на 2 семестр

4. Успешно защищённый проект первого семестра

Лабораторная работа №10

Продвинутая работа с формами и валидация

Цель работы: Реализация сложных форм с валидацией на фронтенде с использованием библиотек Formik и Yup, создание пользовательского интерфейса для управления задачами, проектами и тегами.

Задачи

№	Задача
1	Реализовать форму создания/редактирования задачи с комплексной валидацией
2	Реализовать динамический выбор тегов (мультиселект/чекбоксы) с возможностью создания новых
3	Реализовать форму управления проектами (создание, редактирование, удаление)
4	Реализовать форму управления тегами с визуальным отображением цветовой индикации
5	Реализовать динамическое добавление тегов через форму задачи (создание на лету)

Теоретическая часть

Formik - библиотека для построения форм в React, которая упрощает управление состоянием формы, валидацией и обработкой отправки.

Основные компоненты Formik:

Компонент/Хук	Описание
useFormik()	Хук для управления формой
<Formik>	Компонент-контейнер
<Field>	Компонент поля ввода
<ErrorMessage>	Компонент для отображения ошибок

Yup - библиотека для построения схем валидации JavaScript-объектов.

Пример схемы валидации:

```
const schema = Yup.object({
  title: Yup.string()
    .min(3, 'Минимум 3 символа')
    .max(100, 'Максимум 100 символов')
    .required('Обязательное поле'),
  deadline: Yup.date()
    .min(new Date(), 'Дедлайн не может быть в прошлом')
    .nullable(),
});
```

Управляемые vs Неуправляемые компоненты

Тип	Описание	React-компонент
Управляемый	Значение хранится в state React	<input value={value} onChange={...} />

Тип	Описание	React-компонент
Неуправляемый	Значение хранится в DOM	<input defaultValue={...} /> + ref

Ход выполнения работы

Шаг 1. Установка необходимых пакетов

```
npm install formik yup
npm install @mui/lab
```

Шаг 2. Создание формы задачи (TaskForm) с расширенной валидацией

src/components/tasks/TaskForm.jsx:

```
import React, { useState, useEffect } from 'react';
import { useNavigate, useParams } from 'react-router-dom';
import { useFormik } from 'formik';
import * as Yup from 'yup';
import { useDispatch, useSelector } from 'react-redux';
import {
  TextField,
  Button,
  Box,
  Paper,
  Typography,
  MenuItem,
  FormControl,
  InputLabel,
  Select,
  OutlinedInput
```

```
    Chip,
    Grid,
    FormHelperText,
    Alert,
    Dialog,
    DialogTitle,
    DialogContent,
    DialogActions,
    IconButton,
    InputAdornment,
} from '@mui/material';
import {
    Add as AddIcon,
    Close as CloseIcon,
    CalendarToday as CalendarIcon,
} from '@mui/icons-material';
import { DatePicker } from '@mui/x-date-pickers/DatePicker';
import { LocalizationProvider } from '@mui/x-date-pickers/LocalizationProvider';
import { AdapterDateFns } from '@mui/x-date-pickers/AdapterDateFns';
import { ru } from 'date-fns/locale';
import { createTask, updateTask, fetchTaskById, clearCurrentTask } from '../../store/slices/tasksSlice';
import { fetchProjects } from '../../store/slices/projectsSlice';
import { fetchTags, createTag } from '../../store/slices/tagsSlice';
import { addNotification } from '../../store/slices/uiSlice';

// Схема валидации
const TaskSchema = Yup.object({
    title: Yup.string()
        .min(3, 'Заголовок должен содержать минимум 3 символа')
        .max(200, 'Заголовок не может превышать 200 символов')
        .required('Заголовок обязателен')
        .matches(/^[^<>{}]+$/, 'Заголовок не должен содержать HTML-теги'),
```

```
  description: Yup.string()
    .max(1000, 'Описание не может превышать 1000 символов'),

  deadline: Yup.date()
    .nullable()
    .min(new Date(), 'Дедлайн не может быть в прошлом')
    .typeError('Неверный формат даты'),

  status: Yup.string()
    .oneOf(['New', 'InProgress', 'Completed'], 'Некорректный статус')
    .required('Статус обязателен'),

  priority: Yup.string()
    .oneOf(['Low', 'Medium', 'High'], 'Некорректный приоритет')
    .required('Приоритет обязателен'),

  project_id: Yup.number()
    .nullable()
    .transform((value, originalValue) => originalValue === '' ? null : value),

  tag_ids: Yup.array()
    .of(Yup.number())
    .max(10, 'Нельзя выбрать более 10 тегов'),
});

// Константы
const statusOptions = [
  { value: 'New', label: 'Новая', color: '#3498db' },
  { value: 'InProgress', label: 'В работе', color: '#f39c12' },
  { value: 'Completed', label: 'Выполнена', color: '#2ecc71' },
];
```

```
const priorityOptions = [
  { value: 'Low', label: 'Низкий', color: '#2ecc71' },
  { value: 'Medium', label: 'Средний', color: '#f39c12' },
  { value: 'High', label: 'Высокий', color: '#e74c3c' },
];

const TaskForm = () => {
  const { id } = useParams();
  const navigate = useNavigate();
  const dispatch = useDispatch();
  const isEditMode = Boolean(id);

  // Redux state
  const { currentTask, isLoading: taskLoading } = useSelector((state) => state.tasks);
  const { items: projects, isLoading: projectsLoading } = useSelector((state) => state.projects);
  const { items: tags, isLoading: tagsLoading } = useSelector((state) => state.tags);

  // Локальное состояние
  const [initialLoading, setInitialLoading] = useState(true);
  const [newTagDialogOpen, setNewTagDialogOpen] = useState(false);
  const [newTagName, setNewTagName] = useState('');
  const [newTagColor, setNewTagColor] = useState('#3498db');

  // Загрузка данных
  useEffect(() => {
    const loadData = async () => {
      await Promise.all([
        dispatch(fetchProjects()),
        dispatch(fetchTags()),
      ]);
      if (isEditMode && id) {
```

```
            await dispatch(fetchTaskById(parseInt(id)));
        }
        setInitialLoading(false);
    };

    loadData();

    return () => {
        if (isEditMode) {
            dispatch(clearCurrentTask());
        }
    };
}, [dispatch, id, isEditMode]);

// Formik
const formik = useFormik({
    initialValues: {
        title: currentTask?.title || '',
        description: currentTask?.description || '',
        deadline: currentTask?.deadline ? new Date(currentTask.deadline) : null,
        status: currentTask?.status || 'New',
        priority: currentTask?.priority || 'Medium',
        project_id: currentTask?.project?.id || '',
        tag_ids: currentTask?.tags?.map(t => t.id) || [],
    },
    validationSchema: TaskSchema,
    enableReinitialize: true,
    onSubmit: async (values, { setSubmitting, resetForm }) => {
        // Преобразование данных для отправки
        const submitData = {
            title: values.title.trim(),
            description: values.description?.trim() || '',
```

```
      deadline: values.deadline ? values.deadline.toISOString() : null,
      status: values.status,
      priority: values.priority,
      project_id: values.project_id || null,
      tag_ids: values.tag_ids,
    };

    let result;
    if (isEditMode) {
      result = await dispatch(updateTask({ id: parseInt(id), data: submitData }));
    } else {
      result = await dispatch(createTask(submitData));
    }

    if (result.meta.requestStatus === 'fulfilled') {
      dispatch(addNotification({
        type: 'success',
        message: isEditMode ? 'Задача обновлена' : 'Задача создана',
      }));
      navigate('/tasks');
    } else {
      dispatch(addNotification({
        type: 'error',
        message: typeof result.payload === 'string'
          ? result.payload
          : JSON.stringify(result.payload) || 'Ошибка сохранения задачи',
      }));
      setSubmitting(false);
    }
  },
});
```

```
// Создание нового тега на лету
const handleCreateTag = async () => {
  if (!newTagName.trim()) {
    dispatch(addNotification({ type: 'error', message: 'Введите название тега' }));
    return;
  }

  const result = await dispatch(createTag({ name: newTagName.trim(), color: newTagColor }));

  if (result.meta.requestStatus === 'fulfilled') {
    const newTag = result.payload;
    // Добавляем новый тег в выбранные
    formik.setFieldValue('tag_ids', [...formik.values.tag_ids, newTag.id]);
    setNewTagDialogOpen(false);
    setNewTagName('');
    setNewTagColor('#3498db');
    dispatch(addNotification({ type: 'success', message: 'Тег создан' }));
  } else {
    dispatch(addNotification({ type: 'error', message: 'Ошибка создания тега' }));
  }
};

if (initialLoading || (isEditMode && taskLoading)) {
  return (
    <Box display="flex" justifyContent="center" alignItems="center" minHeight="400px">
      <Typography>Загрузка...</Typography>
    </Box>
  );
}

// Готовые цвета для тегов
const tagColors = ['#3498db', '#2ecc71', '#e74c3c', '#f39c12', '#9b59b6', '#1abc9c'];
```

```
return (
  <LocalizationProvider dateAdapter={AdapterDateFns} adapterLocale={ru}>
    <Box maxWidth="md" mx="auto" mt={4} mb={4}>
      <Paper elevation={3} sx={{ p: 4 }}>
        <Typography variant="h5" gutterBottom>
          {isEditMode ? 'Редактирование задачи' : 'Новая задача'}
        </Typography>

        <form onSubmit={formik handleSubmit}>
          <Grid container spacing={3}>
            {/* Заголовок */}
            <Grid item xs={12}>
              <TextField
                fullWidth
                name="title"
                label="Заголовок *"
                placeholder="Например: Купить продукты"
                value={formik values title}
                onChange={formik handleChange}
                onBlur={formik handleBlur}
                error={formik touched title && Boolean(formik errors title)}
                helperText={formik touched title && formik errors title}
                disabled={formik isSubmitting}
                autoFocus
              />
            </Grid>

            {/* Описание */}
            <Grid item xs={12}>
              <TextField
                fullWidth
```

```
        name="description"
        label="Описание"
        multiline
        rows={4}
        placeholder="Подробное описание задачи..."
        value={formik.values.description}
        onChange={formik.handleChange}
        onBlur={formik.handleBlur}
        error={formik.touched.description && Boolean(formik.errors.description)}
        helperText={formik.touched.description && formik.errors.description}
        disabled={formik.isSubmitting}
      />
    </Grid>

    {/* Дедлайн и Статус */}
    <Grid item xs={12} sm={6}>
      <DatePicker
        label="Дедлайн"
        value={formik.values.deadline}
        onChange={(date) => formik.setFieldValue('deadline', date)}
        slotProps={{
          textField: {
            fullWidth: true,
            name: 'deadline',
            error: formik.touched.deadline && Boolean(formik.errors.deadline),
            helperText: formik.touched.deadline && formik.errors.deadline,
            InputProps: {
              startAdornment: (
                <InputAdornment position="start">
                  <CalendarIcon />
                </InputAdornment>
              ),
```

```
                    },
                  },
                }}
              />
            </Grid>

            <Grid item xs={12} sm={6}>
              <FormControl fullWidth error={formik.touched.status &&
Boolean(formik.errors.status)}>
                <InputLabel>Статус *</InputLabel>
                <Select
                  name="status"
                  value={formik.values.status}
                  onChange={formik.handleChange}
                  onBlur={formik.handleBlur}
                  label="Статус *"
                  disabled={formik.isSubmitting}
                >
                  {statusOptions.map((option) => (
                    <MenuItem key={option.value} value={option.value}>
                      <Box display="flex" alignItems="center" gap={1}>
                        <Box
                          width={10}
                          height={10}
                          borderRadius="50%"
                          bgcolor={option.color}
                        />
                        {option.label}
                      </Box>
                    </MenuItem>
                  ))}
                </Select>
```

```
                        <FormHelperText>{formik touched status &&
formik errors status}</FormHelperText>
                    </FormControl>
                </Grid>

                {/* Приоритет и Проект */}
                <Grid item xs={12} sm={6}>
                    <FormControl fullWidth error={formik touched priority &&
Boolean(formik errors priority)}>
                        <InputLabel>Приоритет *</InputLabel>
                        <Select
                            name="priority"
                            value={formik values priority}
                            onChange={formik handleChange}
                            onBlur={formik handleBlur}
                            label="Приоритет *"
                            disabled={formik isSubmitting}
                        >
                            {priorityOptions map(option => (
                                <MenuItem key={option value} value={option value}>
                                    <Box display="flex" alignItems="center" gap={1}>
                                        <Box
                                            width={10}
                                            height={10}
                                            borderRadius="50%"
                                            bgcolor={option color}
                                        />
                                        {option label}
                                    </Box>
                                </MenuItem>
                            ))}
                        </Select>
```

```
                    <FormHelperText>{formik touched priority &&
formik errors priority}</FormHelperText>
                  </FormControl>
                </Grid>

                <Grid item xs={12} sm={6}>
                  <FormControl fullWidth disabled={projectsLoading || formik isSubmitting}>
                    <InputLabel>Проект</InputLabel>
                    <Select
                      name="project_id"
                      value={formik values project_id}
                      onChange={formik handleChange}
                      label="Проект"
                    >
                      <MenuItem value="">— Без проекта —</MenuItem>
                      {projects map(project => (
                        <MenuItem key={project id} value={project id}>
                          <Box display="flex" alignItems="center" gap={1}>
                            <Box
                              width={12}
                              height={12}
                              borderRadius="2px"
                              bgcolor={project color || '#95a5a6'}
                            />
                            {project name}
                          </Box>
                        </MenuItem>
                      ))}
                    </Select>
                  </FormControl>
                </Grid>
```

```
{/* Теги (мультиселект с возможностью создания) */}
<Grid item xs={12}>
<FormControl fullWidth error={formik touched tag_ids &&
Boolean(formik errors tag_ids)}>
<InputLabel>Теги</InputLabel>
<Select
multiple
name="tag_ids"
value={formik values tag_ids}
onChange={formik handleChange}
onBlur={formik handleBlur}
input={<OutlinedInput label="Теги" />}
renderValue={(selected) => (
<Box sx={{ display: 'flex', flexWrap: 'wrap', gap: 0.5 }}>
{selected map((value) => {
const tag = tags find(t => t id === value);
return (
<Chip
key={value}
label={tag?.name || value}
size="small"
sx={{
bgcolor: tag?.color || '#95a5a6',
color: 'white',
}}
/>
);
})}
</Box>
)}
MenuProps={{
PaperProps: {
```

```
            style: {
              maxHeight: 300,
            },
          },
        }}
      >
        {tags.map((tag) => (
          <MenuItem key={tag.id} value={tag.id}>
            <Box display="flex" alignItems="center" gap={1}>
              <Box
                width={12}
                height={12}
                borderRadius="50%"
                bgcolor={tag.color || '#95a5a6'}
              />
              {tag.name}
            </Box>
          </MenuItem>
        ))}
        <MenuItem
          onClick={(e) => {
            e.preventDefault();
            e.stopPropagation();
            setNewTagDialogOpen(true);
          }}
          sx={{ color: '#3498db', borderTop: '1px solid #eee' }}
        >
          <AddIcon fontSize="small" sx={{ mr: 1 }} />
          Создать новый тег
        </MenuItem>
      </Select>
```

```
                        <FormHelperText>{formik.touched.tag_ids &&
formik.errors.tag_ids}</FormHelperText>
                      </FormControl>
                    </Grid>

                    {/* Кнопки действий */}
                    <Grid item xs={12}>
                      <Box display="flex" gap={2} justifyContent="flex-end">
                        <Button
                          variant="outlined"
                          onClick={() => navigate('/tasks')}
                          disabled={formik.isSubmitting}
                        >
                          Отмена
                        </Button>
                        <Button
                          type="submit"
                          variant="contained"
                          color="primary"
                          disabled={formik.isSubmitting || !formik.isValid}
                        >
                          {formik.isSubmitting ? 'Сохранение...' : (isEditMode ? 'Сохранить' : 'Создать')}
                        </Button>
                      </Box>
                    </Grid>
                  </Grid>
                </form>
              </Paper>
            </Box>

            {/* Диалог создания нового тега */}
            <Dialog open={newTagDialogOpen} onClose={() => setNewTagDialogOpen(false)}>
```

```
<DialogTitle>
  Создание нового тега
  <IconButton
    aria-label="close"
    onClick={() => setNewTagDialogOpen(false)}
    sx={{ position: 'absolute', right: 8, top: 8 }}
  >
    <CloseIcon />
  </IconButton>
</DialogTitle>
<DialogContent>
  <TextField
    autoFocus
    margin="dense"
    label="Название тега"
    fullWidth
    variant="outlined"
    value={newTagName}
    onChange={(e) => setNewTagName(e target value)}
    placeholder="Например: Важное, Личное, Работа"
  />

  <Typography variant="body2" sx={{ mt: 2, mb: 1 }}>
    Цвет тега:
  </Typography>
  <Box display="flex" gap={1}>
    {tagColors map(color => (
      <Box
        key={color}
        width={36}
        height={36}
        borderRadius="50%"
```

```
                    bgcolor={color}
                    border={newTagColor === color ? '3px solid #2c3e50' : 'none'}
                    sx={{ cursor: 'pointer', transition: 'transform 0.2s' }}
                    onClick={() => setNewTagColor(color)}
                    onMouseEnter={(e) => e.currentTarget.style.transform = 'scale(1.1)'}
                    onMouseLeave={(e) => e.currentTarget.style.transform = 'scale(1)'}
                  />
                ))}
              </Box>
            </DialogContent>
            <DialogActions>
              <Button onClick={() => setNewTagDialogOpen(false)}>Отмена</Button>
              <Button onClick={handleCreateTag} variant="contained" color="primary">
                Создать
              </Button>
            </DialogActions>
          </Dialog>
        </LocalizationProvider>
    );
};

export default TaskForm;
```

Шаг 3. Создание формы управления проектами src/components/projects/ProjectManager.jsx:

```
import React, { useState, useEffect } from 'react';
import {
  Box,
  Paper,
  Typography,
  TextField,
```

```
  Button,
  List,
  ListItem,
  ListItemText,
  ListItemSecondaryAction,
  IconButton,
  Dialog,
  DialogTitle,
  DialogContent,
  DialogActions,
  Grid,
  InputAdornment,
  Chip,
} from '@mui/material';
import {
  Add as AddIcon,
  Edit as EditIcon,
  Delete as DeleteIcon,
  ColorLens as ColorIcon
} from '@mui/icons-material';
import { useDispatch, useSelector } from 'react-redux';
import { fetchProjects, createProject, updateProject, deleteProject } from
'../../store/slices/projectsSlice';
import { addNotification } from '../../store/slices/uiSlice';

// Цвета для проектов
const projectColors = [
  '#3498db', '#2ecc71', '#e74c3c', '#f39c12', '#9b59b6',
  '#1abc9c', '#e67e22', '#34495e', '#16a085', '#27ae60',
  '#2980b9', '#8e44ad', '#2c3e50', '#d35400', '#c0392b',
];
```

```
const ProjectManager = () => {
  const dispatch = useDispatch();
  const { items: projects, isLoading } = useSelector((state) => state.projects);

  const [dialogOpen, setDialogOpen] = useState(false);
  const [editingProject, setEditingProject] = useState(null);
  const [formData, setFormData] = useState({ name: '', description: '', color: '#3498db' });
  const [errors, setErrors] = useState({});

  useEffect(() => {
    dispatch(fetchProjects());
  }, [dispatch]);

  const validateForm = () => {
    const newErrors = {};
    if (!formData.name.trim()) {
      newErrors.name = 'Название проекта обязательно';
    } else if (formData.name.length > 100) {
      newErrors.name = 'Название не может превышать 100 символов';
    }
    setErrors(newErrors);
    return Object.keys(newErrors).length === 0;
  };

  const handleOpenDialog = (project = null) => {
    if (project) {
      setEditingProject(project);
      setFormData({
        name: project.name,
        description: project.description || '',
        color: project.color || '#3498db',
      });
```

```
    } else {
      setEditingProject(null);
      setFormData({ name: '', description: '', color: '#3498db' });
    }
    setErrors({});
    setDialogOpen(true);
  };

  const handleCloseDialog = () => {
    setDialogOpen(false);
    setEditingProject(null);
    setFormData({ name: '', description: '', color: '#3498db' });
  };

  const handleSubmit = async () => {
    if (!validateForm()) return;

    let result;
    if (editingProject) {
      result = await dispatch(updateProject({
        id: editingProject.id,
        data: { ...formData, name: formData.name.trim() }
      }));
    } else {
      result = await dispatch(createProject({ ...formData, name: formData.name.trim() }));
    }

    if (result.meta.requestStatus === 'fulfilled') {
      dispatch(addNotification({
        type: 'success',
        message: editingProject ? 'Проект обновлён' : 'Проект создан',
      }));
```

```
      handleCloseDialog();
    } else {
      dispatch(addNotification({
        type: 'error',
        message: result.payload || 'Ошибка сохранения проекта',
      }));
    }
  };

  const handleDelete = async (id, name) => {
    if (window.confirm(`Вы уверены, что хотите удалить проект "${name}"?`)) {
      const result = await dispatch(deleteProject(id));
      if (result.meta.requestStatus === 'fulfilled') {
        dispatch(addNotification({ type: 'success', message: 'Проект удалён' }));
      } else {
        dispatch(addNotification({ type: 'error', message: 'Ошибка удаления проекта' }));
      }
    }
  };

  return (
    <Box>
      <Box display="flex" justifyContent="space-between" alignItems="center" mb={3}>
        <Typography variant="h5">Проекты</Typography>
        <Button
          variant="contained"
          startIcon={<AddIcon />}
          onClick={() => handleOpenDialog()}
        >
          Создать проект
        </Button>
      </Box>
```

```
        <Paper>
          {isLoading ? (
            <Box p={4} textAlign="center">Загрузка...</Box>
          ) : projects.length === 0 ? (
            <Box p={4} textAlign="center">
              <Typography color="text.secondary">
                Нет проектов. Создайте первый проект для организации задач.
              </Typography>
            </Box>
          ) : (
            <List>
              {projects.map((project) => (
                <ListItem key={project.id} divider>
                  <Box width={8} height={40} bgcolor={project.color || '#95a5a6'} borderRadius="4px" mr={2} />
                  <ListItemText
                    primary={project.name}
                    secondary={project.description || 'Без описания'}
                    primaryTypographyProps={{ fontWeight: 'medium' }}
                  />
                  <ListItemSecondaryAction>
                    <IconButton edge="end" onClick={() => handleOpenDialog(project)} sx={{ mr: 1 }}>
                      <EditIcon />
                    </IconButton>
                    <IconButton edge="end" onClick={() => handleDelete(project.id, project.name)} color="error">
                      <DeleteIcon />
                    </IconButton>
                  </ListItemSecondaryAction>
                </ListItem>
              ))}
```

```
        </List>
      )}
    </Paper>

    {/* Диалог создания/редактирования проекта */}
    <Dialog open={dialogOpen} onClose={handleCloseDialog} maxWidth="sm" fullWidth>
      <DialogTitle>
        {editingProject ? 'Редактирование проекта' : 'Новый проект'}
      </DialogTitle>
      <DialogContent>
        <Grid container spacing={2} sx={{ mt: 1 }}>
          <Grid item xs={12}>
            <TextField
              label="Название проекта *"
              fullWidth
              value={formData.name}
              onChange={(e) => setFormData({ ...formData, name: e.target.value })}
              error={!!errors.name}
              helperText={errors.name}
              autoFocus
            />
          </Grid>
          <Grid item xs={12}>
            <TextField
              label="Описание"
              fullWidth
              multiline
              rows={3}
              value={formData.description}
              onChange={(e) => setFormData({ ...formData, description: e.target.value })}
            />
          </Grid>
```

```
            <Grid item xs={12}>
              <Typography variant="body2" gutterBottom>
                Цвет проекта:
              </Typography>
              <Box display="flex" flexWrap="wrap" gap={1}>
                {projectColors.map(color => (
                  <Box
                    key={color}
                    width={40}
                    height={40}
                    borderRadius="50%"
                    bgcolor={color}
                    border={formData.color === color ? '3px solid #2c3e50' : 'none'}
                    sx={{ cursor: 'pointer', transition: 'transform 0.2s' }}
                    onClick={() => setFormData({ ...formData, color })}
                  />
                ))}
              </Box>
            </Grid>
          </Grid>
        </DialogContent>
        <DialogActions>
          <Button onClick={handleCloseDialog}>Отмена</Button>
          <Button onClick={handleSubmit} variant="contained" color="primary">
            {editingProject ? 'Сохранить' : 'Создать'}
          </Button>
        </DialogActions>
      </Dialog>
    </Box>
  );
};
```

```
export default ProjectManager;
```

Шаг 4. Создание формы управления тегами

src/components/tags/TagManager.jsx:

```
import React, { useState, useEffect } from 'react';
import {
  Box,
  Paper,
  Typography,
  TextField,
  Button,
  Grid,
  Chip,
  IconButton,
  Dialog,
  DialogTitle,
  DialogContent,
  DialogActions,
} from '@mui/material';
import {
  Add as AddIcon,
  Delete as DeleteIcon,
  Edit as EditIcon,
} from '@mui/icons-material';
import { useDispatch, useSelector } from 'react-redux';
import { fetchTags, createTag, deleteTag, updateTag } from '../../store/slices/tagsSlice';
import { addNotification } from '../../store/slices/uiSlice';

const tagColors = ['#3498db', '#2ecc71', '#e74c3c', '#f39c12', '#9b59b6', '#1abc9c', '#e67e22',
'#34495e'];
```

```
const TagManager = () => {
  const dispatch = useDispatch();
  const { items: tags, isLoading } = useSelector((state) => state.tags);

  const [dialogOpen, setDialogOpen] = useState(false);
  const [editingTag, setEditingTag] = useState(null);
  const [formData, setFormData] = useState({ name: '', color: '#3498db' });
  const [errors, setErrors] = useState({});

  useEffect(() => {
    dispatch(fetchTags());
  }, [dispatch]);

  const validateForm = () => {
    const newErrors = {};
    if (!formData.name.trim()) {
      newErrors.name = 'Название тега обязательно';
    } else if (formData.name.length > 50) {
      newErrors.name = 'Название не может превышать 50 символов';
    }
    setErrors(newErrors);
    return Object.keys(newErrors).length === 0;
  };

  const handleOpenDialog = (tag = null) => {
    if (tag) {
      setEditingTag(tag);
      setFormData({ name: tag.name, color: tag.color || '#3498db' });
    } else {
      setEditingTag(null);
      setFormData({ name: '', color: '#3498db' });
    }
```

```
    setErrors({});
    setDialogOpen(true);
  };

  const handleSubmit = async () => {
    if (!validateForm()) return;

    let result;
    if (editingTag) {
      result = await dispatch(updateTag({
        id: editingTag.id,
        data: { name: formData.name.trim(), color: formData.color }
      }));
    } else {
      result = await dispatch(createTag({ name: formData.name.trim(), color: formData.color }));
    }

    if (result.meta.requestStatus === 'fulfilled') {
      dispatch(addNotification({
        type: 'success',
        message: editingTag ? 'Тег обновлён' : 'Тег создан',
      }));
      handleCloseDialog();
    } else {
      dispatch(addNotification({
        type: 'error',
        message: result.payload || 'Ошибка сохранения тега',
      }));
    }
  };

  const handleDelete = async (id, name) => {
```

```
    if (window.confirm(`Вы уверены, что хотите удалить тег "${name}"?`)) {
      const result = await dispatch(deleteTag(id));
      if (result.meta.requestStatus === 'fulfilled') {
        dispatch(addNotification({ type: 'success', message: 'Тег удалён' }));
      } else {
        dispatch(addNotification({ type: 'error', message: 'Ошибка удаления тега' }));
      }
    }
  };

  return (
    <Box>
      <Box display="flex" justifyContent="space-between" alignItems="center" mb={3}>
        <Typography variant="h5">Теги</Typography>
        <Button
          variant="contained"
          startIcon={<AddIcon />}
          onClick={() => handleOpenDialog()}
        >
          Создать тег
        </Button>
      </Box>

      <Paper sx={{ p: 3 }}>
        {isLoading ? (
          <Box textAlign="center">Загрузка...</Box>
        ) : tags.length === 0 ? (
          <Box textAlign="center">
            <Typography color="text.secondary">
              Нет тегов. Создайте теги для удобной категоризации задач
            </Typography>
          </Box>
```

```
      ) : (
        <Box display="flex" flexWrap="wrap" gap={2}>
          {tags.map((tag) => (
            <Chip
              key={tag.id}
              label={tag.name}
              onDelete={() => handleDelete(tag.id, tag.name)}
              deleteIcon={<DeleteIcon />}
              onClick={() => handleOpenDialog(tag)}
              sx={{
                bgcolor: tag.color || '#3498db',
                color: 'white',
                '&:hover': {
                  bgcolor: tag.color || '#3498db',
                  opacity: 0.9,
                },
              }}
            />
          ))}
        </Box>
      )}
    </Paper>

    {/* Диалог создания/редактирования тега */}
    <Dialog open={dialogOpen} onClose={() => setDialogOpen(false)} maxWidth="sm" fullWidth>
      <DialogTitle>
        {editingTag ? 'Редактирование тега' : 'Новый тег'}
      </DialogTitle>
      <DialogContent>
        <TextField
          label="Название тега *"
          fullWidth
```

```
        value={formData.name}
        onChange={(e) => setFormData({ ...formData, name: e.target.value })}
        error={!!errors.name}
        helperText={errors.name}
        margin="normal"
        autoFocus
      />

      <Typography variant="body2" sx={{ mt: 2, mb: 1 }}>
        Цвет тега:
      </Typography>
      <Box display="flex" flexWrap="wrap" gap={1}>
        {tagColors.map((color) => (
          <Box
            key={color}
            width={40}
            height={40}
            borderRadius="50%"
            bgcolor={color}
            border={formData.color === color ? '3px solid #2c3e50' : 'none'}
            sx={{ cursor: 'pointer', transition: 'transform 0.2s' }}
            onClick={() => setFormData({ ...formData, color })}
          />
        ))}
      </Box>
    </DialogContent>
    <DialogActions>
      <Button onClick={() => setDialogOpen(false)}>Отмена</Button>
      <Button onClick={handleSubmit} variant="contained" color="primary">
        {editingTag ? 'Сохранить' : 'Создать'}
      </Button>
    </DialogActions>
```

```
        </Dialog>
      </Box>
    );
};

export default TagManager;
```

Шаг 5. Интеграция форм в маршрутизацию

src/App.jsx (дополнение):

```
import TaskForm from './components/tasks/TaskForm';
import ProjectManager from './components/projects/ProjectManager';
import TagManager from './components/tags/TagManager';

// ... внутри Routes
<Route path="/tasks/create" element={
  <PrivateRoute>
    <TaskForm />
  </PrivateRoute>
} />
<Route path="/tasks/:id/edit" element={
  <PrivateRoute>
    <TaskForm />
  </PrivateRoute>
} />
<Route path="/projects" element={
  <PrivateRoute>
    <ProjectManager />
  </PrivateRoute>
} />
<Route path="/tags" element={
  <PrivateRoute>
```

```
      <TagManager />
    </PrivateRoute>
  } />
```

Контрольные вопросы

1. Для чего используется библиотека Formik? Какие проблемы она решает?
2. Чем отличается управляемый компонент от неуправляемого?
3. Как Yup взаимодействует с Formik для валидации?
4. Как реализовать динамическое добавление тегов при создании задачи?
5. Какие типы валидации можно реализовать на стороне клиента?
6. Как синхронизировать состояние формы с глобальным store (Redux)?

Критерии оценки

Критерий	Баллы
Реализована форма задачи с валидацией	3
Реализован динамический выбор тегов	2
Реализован менеджер проектов	2
Реализован менеджер тегов	2
Реализовано создание тегов "на лету"	1
Итого	10

Результат выполнения

После выполнения лабораторной работы №10 студенты должны получить:

1. Форму создания/редактирования задачи с комплексной валидацией

2. Менеджер проектов с CRUD-операциями
3. Менеджер тегов с визуальным оформлением
4. Возможность создания тегов непосредственно при создании задачи
5. Валидацию всех форм на стороне клиента

Лабораторная работа №11

Календарь и временная шкала

Цель работы: Реализация визуализации задач на календаре и доске Kanban с возможностью drag-and-drop для изменения статусов и дат задач.

Задачи

№	Задача
1	Реализовать компонент календаря для отображения задач с дедлайнами
2	Реализовать возможность drag-and-drop для изменения дат задач
3	Реализовать представление "Kanban board" (доска задач по статусам)
4	Реализовать перетаскивание карточек между колонками Kanban
5	Добавить возможность быстрого изменения статуса задачи через drag-and-drop

Теоретическая часть

Календарь (Calendar)

Библиотеки для реализации календаря в React:

Библиотека	Описание	Плюсы	Минусы
FullCalendar	Мощная библиотека с широкими возможностями	Много функций, хорошая документация	Большой размер бандла
React Big Calendar	Простая и настраиваемая	Легкая, хорошо работает с drag-and-drop	Меньше функций
date-fns / moment.js	Для кастомной реализации	Полный контроль	Требует больше кода

Библиотеки для drag-and-drop:

Библиотека	**Описание**	**Рекомендация**
react-beautiful-dnd	От Atlassian, красивая анимация	Для Kanban досок
dnd-kit	Современная, легкая, быстрая	Универсальная
react-dnd	Низкоуровневая, гибкая	Для сложных сценариев

Kanban Board

Колонки Kanban-доски:

- To Do (Новые задачи) - задачи, которые еще не начаты
- In Progress (В работе) - активные задачи

- Review (На проверке) - задачи, ожидающие проверки (опционально)
- Done (Выполнены) - завершенные задачи

Ход выполнения работы

Шаг 1. Установка необходимых пакетов

```
# Для календаря
npm install @fullcalendar/react @fullcalendar/daygrid @fullcalendar/timegrid @fullcalendar/interaction

# Для drag-and-drop
npm install @dnd-kit/core @dnd-kit/sortable @dnd-kit/utilities

# Для работы с датами
npm install date-fns
```

Шаг 2. Создание Redux слайса для обновления статуса задачи (дополнение)

src/store/slices/tasksSlice.js (дополнение):

javascript

```
// Добавляем асинхронное действие для быстрого обновления даты/статуса
export const updateTaskQuick = createAsyncThunk(
  'tasks/updateTaskQuick',
  async ({ id, updates }, { rejectWithValue }) => {
    try {
      const response = await apiClient.patch(`/tasks/${id}/`, updates);
      return response.data;
    } catch (error) {
      return rejectWithValue(error.response?.data?.detail || 'Ошибка обновления');
    }
  }
);
```

```
// Добавляем обработчик в extraReducers
// ...
.addCase(updateTaskQuick.fulfilled, (state, action) => {
  const index = state.items.findIndex(t => t.id === action.payload.id);
  if (index !== -1) {
    state.items[index] = action.payload;
  }
  if (state.currentTask?.id === action.payload.id) {
    state.currentTask = action.payload;
  }
})
```

Шаг 3. Создание компонента TaskCalendar

src/components/calendar/TaskCalendar.jsx:

```
import React, { useState, useEffect, useCallback } from 'react';
import FullCalendar from '@fullcalendar/react';
import dayGridPlugin from '@fullcalendar/daygrid';
import timeGridPlugin from '@fullcalendar/timegrid';
import interactionPlugin from '@fullcalendar/interaction';
import { useDispatch, useSelector } from 'react-redux';
import { Box, Typography, Chip, Tooltip, IconButton, Menu, MenuItem } from '@mui/material';
import { Edit as EditIcon, Delete as DeleteIcon, DragIndicator as DragIcon } from '@mui/icons-material';
import { fetchTasks, updateTaskQuick } from '../../store/slices/tasksSlice';
import { addNotification } from '../../store/slices/uiSlice';
import { useNavigate } from 'react-router-dom';

// Цвета для приоритетов
const priorityColors = {
  Low: '#2ecc71',
  Medium: '#f39c12',
  High: '#e74c3c',
```

```
};

// Функция для рендеринга события в календаре
const renderEventContent = (eventInfo) => {
  const priority = eventInfo.event.extendedProps.priority;
  const status = eventInfo.event.extendedProps.status;

  return (
    <Tooltip title={`${eventInfo.event.title}\nПриоритет: ${priority}\nСтатус: ${status}`}>
      <Box
        sx={{
          backgroundColor: priorityColors[priority] || '#3498db',
          color: 'white',
          padding: '2px 4px',
          borderRadius: '4px',
          fontSize: '0.8rem',
          cursor: 'pointer',
          opacity: status === 'Completed' ? 0.6 : 1,
          textDecoration: status === 'Completed' ? 'line-through' : 'none',
          overflow: 'hidden',
          whiteSpace: 'nowrap',
          textOverflow: 'ellipsis',
        }}
      >
        <strong>{eventInfo.timeText}</strong> {eventInfo.event.title}
      </Box>
    </Tooltip>
  );
};

const TaskCalendar = () => {
  const dispatch = useDispatch();
```

```
const navigate = useNavigate();
const { items: tasks, isLoading } = useSelector((state) => state tasks);
const [calendarEvents, setCalendarEvents] = useState([]);
const [selectedEvent, setSelectedEvent] = useState(null);
const [contextMenu, setContextMenu] = useState(null);

// Загрузка задач
useEffect(() => {
  dispatch(fetchTasks({ page: 1, pageSize: 100 }));
}, [dispatch]);

// Преобразование задач в события календаря
useEffect(() => {
  const events = tasks
    .filter(task => task deadline)
    .map(task => ({
      id: task id,
      title: task title,
      start: task deadline,
      end: task deadline,
      allDay: false,
      extendedProps: {
        priority: task priority,
        status: task status,
        description: task description,
        project: task project?.name,
      },
    }));
  setCalendarEvents(events);
}, [tasks]);

// Обработка клика по событию
```

```
const handleEventClick = useCallback((clickInfo) => {
    navigate(`/tasks/${clickInfo.event.id}`);
}, [navigate]);

// Обработка изменения даты через drag-and-drop
const handleEventDrop = useCallback(async (dropInfo) => {
    const taskId = dropInfo.event.id;
    const newDeadline = dropInfo.event.start.toISOString();

    const result = await dispatch(updateTaskQuick({
        id: taskId,
        updates: { deadline: newDeadline }
    }));

    if (result.meta.requestStatus === 'fulfilled') {
        dispatch(addNotification({
            type: 'success',
            message: 'Дедлайн задачи изменён',
        }));
    } else {
        // Откат изменения в календаре
        dropInfo.revert();
        dispatch(addNotification({
            type: 'error',
            message: 'Ошибка изменения даты',
        }));
    }
}, [dispatch]);

// Обработка изменения размера события (длительности)
const handleEventResize = useCallback(async (resizeInfo) => {
    const taskId = resizeInfo.event.id;
```

```
    const newDeadline = resizeInfo.event.start.toISOString();

    const result = await dispatch(updateTaskQuick({
      id: taskId,
      updates: { deadline: newDeadline }
    }));

    if (result.meta.requestStatus !== 'fulfilled') {
      resizeInfo.revert();
    }
  }, [dispatch]);

  // Контекстное меню
  const handleEventContextMenu = useCallback((info) => {
    info.jsEvent.preventDefault();
    setSelectedEvent(info.event);
    setContextMenu({
      mouseX: info.jsEvent.clientX - 2,
      mouseY: info.jsEvent.clientY - 4,
    });
  }, []);

  const handleCloseContextMenu = () => {
    setContextMenu(null);
    setSelectedEvent(null);
  };

  const handleEditTask = () => {
    if (selectedEvent) {
      navigate(`/tasks/${selectedEvent.id}/edit`);
    }
    handleCloseContextMenu();
```

```
};

const handleDeleteTask = async () => {
  if (selectedEvent) {
    // Будет реализовано в ЛР №13
    handleCloseContextMenu();
  }
};

return (
  <Box sx={{ height: 'calc(100vh - 120px)', p: 2 }}>
    <FullCalendar
      plugins={[dayGridPlugin, timeGridPlugin, interactionPlugin]}
      headerToolbar={{
        left: 'prev,next today',
        center: 'title',
        right: 'dayGridMonth,timeGridWeek,timeGridDay',
      }}
      initialView="dayGridMonth"
      editable={true}
      selectable={true}
      droppable={true}
      events={calendarEvents}
      eventContent={renderEventContent}
      eventClick={handleEventClick}
      eventDrop={handleEventDrop}
      eventResize={handleEventResize}
      eventDidMount={(info) => {
        // Добавляем обработчик контекстного меню
        info.el.addEventListener('contextmenu', (e) => {
          e.preventDefault();
          handleEventContextMenu({ event: info.event, jsEvent: e });
```

```
      });
    }}
    height="100%"
    locale="ru"
    buttonText={{
      today: 'Сегодня',
      month: 'Месяц',
      week: 'Неделя',
      day: 'День',
    }}
    slotLabelFormat={{
      hour: '2-digit',
      minute: '2-digit',
      hour12: false,
    }}
    eventTimeFormat={{
      hour: '2-digit',
      minute: '2-digit',
      hour12: false,
    }}
  />

  {/* Контекстное меню */}
  <Menu
    open={contextMenu !== null}
    onClose={handleCloseContextMenu}
    anchorReference="anchorPosition"
    anchorPosition={
      contextMenu !== null
        ? { top: contextMenu.mouseY, left: contextMenu.mouseX }
        : undefined
    }
```

```
        >
          <MenuItem onClick={handleEditTask}>
            <EditIcon fontSize="small" sx={{ mr: 1 }} />
            Редактировать
          </MenuItem>
          <MenuItem onClick={handleDeleteTask} sx={{ color: '#e74c3c' }}>
            <DeleteIcon fontSize="small" sx={{ mr: 1 }} />
            Удалить
          </MenuItem>
        </Menu>
      </Box>
    );
};

export default TaskCalendar;
```

Шаг 4. Создание компонента KanbanBoard

src/components/kanban/KanbanBoard.jsx:

```
import React, { useState, useEffect, useCallback } from 'react';
import { useDispatch, useSelector } from 'react-redux';
import {
  DndContext,
  closestCenter,
  KeyboardSensor,
  PointerSensor,
  useSensor,
  useSensors,
  DragOverlay,
} from '@dnd-kit/core';
import {
  SortableContext
```

```
  verticalListSortingStrategy,
  horizontalListSortingStrategy,
} from '@dnd-kit/sortable';
import {
  Box,
  Paper,
  Typography,
  Chip,
  IconButton,
  Avatar,
  CircularProgress,
} from '@mui/material';
import { AccessTime as AccessTimeIcon, Flag as FlagIcon } from '@mui/icons-material';
import { fetchTasks, updateTaskQuick } from '../../store/slices/tasksSlice';
import { addNotification } from '../../store/slices/uiSlice';
import KanbanColumn from './KanbanColumn';
import KanbanCard from './KanbanCard';

// Конфигурация колонок Kanban
export const KANBAN_COLUMNS = [
  { id: 'New', title: '📋 To Do', color: '#3498db', limit: 10 },
  { id: 'InProgress', title: '⚙️ In Progress', color: '#f39c12', limit: 5 },
  { id: 'Completed', title: '✅ Done', color: '#2ecc71', limit: null },
];

const KanbanBoard = () => {
  const dispatch = useDispatch();
  const { items: tasks, isLoading } = useSelector((state) => state.tasks);
  const [columns, setColumns] = useState({});
  const [activeTask, setActiveTask] = useState(null);

  // Настройка сенсоров для drag-and-drop
```

```
const sensors = useSensors(
    useSensor(PointerSensor, {
        activationConstraint: {
            distance: 5,
        },
    }),
    useSensor(KeyboardSensor)
);

// Загрузка задач
useEffect(() => {
    dispatch(fetchTasks({ page: 1, pageSize: 100 }));
}, [dispatch]);

// Группировка задач по статусам
useEffect(() => {
    const grouped = {
        New: [],
        InProgress: [],
        Completed: [],
    };

    tasks.forEach(task => {
        if (grouped[task.status]) {
            grouped[task.status].push(task);
        } else {
            grouped.New.push(task);
        }
    });

    // Сортировка задач внутри колонок (по приоритету и дедлайну)
    Object.keys(grouped).forEach(status => {
```

```
      grouped[status].sort((a, b) => {
        // Сначала по приоритету
        const priorityOrder = { High: 0, Medium: 1, Low: 2 };
        if (priorityOrder[a.priority] !== priorityOrder[b.priority]) {
          return priorityOrder[a.priority] - priorityOrder[b.priority];
        }
        // Затем по дедлайну
        if (a.deadline && b.deadline) {
          return new Date(a.deadline) - new Date(b.deadline);
        }
        return 0;
      });
    });

    setColumns(grouped);
  }, [tasks]);

  // Обработка окончания drag-and-drop
  const handleDragEnd = useCallback(async (event) => {
    const { active, over } = event;
    setActiveTask(null);

    if (!over) return;

    const taskId = parseInt(active.id);
    const newStatus = over.id;
    const oldStatus = tasks.find(t => t.id === taskId)?.status;

    if (oldStatus === newStatus) return;

    // Оптимистичное обновление UI
    setColumns(prev => {
```

```
      const newColumns = { ...prev };
      const task = newColumns[oldStatus].find(t => t.id === taskId);
      if (task) {
        newColumns[oldStatus] = newColumns[oldStatus].filter(t => t.id !== taskId);
        task.status = newStatus;
        newColumns[newStatus] = [task, ...newColumns[newStatus]];
      }
      return newColumns;
    });

    // Отправка запроса на сервер
    const result = await dispatch(updateTaskQuick({
      id: taskId,
      updates: { status: newStatus }
    }));

    if (result.meta.requestStatus !== 'fulfilled') {
      // Откат при ошибке
      dispatch(fetchTasks({ page: 1, pageSize: 100 }));
      dispatch(addNotification({
        type: 'error',
        message: 'Ошибка изменения статуса',
      }));
    }
  }, [dispatch, tasks]);

  const handleDragStart = useCallback((event) => {
    const taskId = parseInt(event.active.id);
    const task = tasks.find(t => t.id === taskId);
    setActiveTask(task);
  }, [tasks]);
```

```
if (isLoading && tasks.length === 0) {
  return (
    <Box display="flex" justifyContent="center" alignItems="center" minHeight="400px">
      <CircularProgress />
    </Box>
  );
}

return (
  <DndContext
    sensors={sensors}
    collisionDetection={closestCenter}
    onDragStart={handleDragStart}
    onDragEnd={handleDragEnd}
  >
    <Box
      sx={{
        display: 'flex',
        gap: 2,
        p: 2,
        overflowX: 'auto',
        minHeight: 'calc(100vh - 120px)',
        alignItems: 'flex-start',
      }}
    >
      {KANBAN_COLUMNS.map(column => (
        <KanbanColumn
          key={column.id}
          id={column.id}
          title={column.title}
          color={column.color}
          limit={column.limit}
```

```
            tasks={columns[column.id] || []}
          />
        ))}
      </Box>

      <DragOverlay>
        {activeTask ? (
          <KanbanCard task={activeTask} isDragging />
        ) : null}
      </DragOverlay>
    </DndContext>
  );
};

export default KanbanBoard;
```

Шаг 5. Создание компонента KanbanColumn

src/components/kanban/KanbanColumn.jsx:

```
import React from 'react';
import { useDroppable } from '@dnd-kit/core';
import {
  SortableContext,
  verticalListSortingStrategy,
} from '@dnd-kit/sortable';
import {
  Paper,
  Typography,
  Box,
  Chip,
} from '@mui/material';
import KanbanCard from './KanbanCard';
```

```
const KanbanColumn = ({ id, title, color, limit, tasks }) => {
  const { setNodeRef } = useDroppable({ id });

  const isOverLimit = limit && tasks.length >= limit;

  return (
    <Paper
      ref={setNodeRef}
      elevation={2}
      sx={{
        minWidth: 320,
        width: 320,
        bgcolor: '#f8f9fa',
        borderRadius: 2,
        overflow: 'hidden',
        display: 'flex',
        flexDirection: 'column',
        maxHeight: 'calc(100vh - 120px)',
      }}
    >
      {/* Заголовок колонки */}
      <Box
        sx={{
          p: 2,
          bgcolor: color,
          color: 'white',
          display: 'flex',
          justifyContent: 'space-between',
          alignItems: 'center',
        }}
      >
```

```
    <Typography variant="h6">
      {title}
    </Typography>
    <Chip
      label={tasks.length}
      size="small"
      sx={{
        bgcolor: 'rgba(255,255,255,0.3)',
        color: 'white',
        fontWeight: 'bold',
      }}
    />
  </Box>

  {/* Предупреждение о лимите */}
  {isOverLimit && (
    <Box sx={{ p: 1, bgcolor: '#fff3cd', color: '#856404' }}>
      <Typography variant="caption">
        ⚠ Лимит колонки: {limit} задач
      </Typography>
    </Box>
  )}

  {/* Список задач */}
  <SortableContext
    items={tasks.map(t => t.id.toString())}
    strategy={verticalListSortingStrategy}
  >
    <Box
      sx={{
        p: 2,
        flex: 1,
```

```
            overflowY: 'auto',
            minHeight: 200,
            display: 'flex',
            flexDirection: 'column',
            gap: 1.5,
          }}
        >
          {tasks.length === 0 ? (
            <Box
              sx={{
                textAlign: 'center',
                py: 4,
                color: 'text.secondary',
                border: '1px dashed #ddd',
                borderRadius: 2,
              }}
            >
              <Typography variant="body2">
                Перетащите задачу сюда
              </Typography>
            </Box>
          ) : (
            tasks.map(task => (
              <KanbanCard key={task.id} task={task} />
            ))
          )}
        </Box>
      </SortableContext>
    </Paper>
  );
};
```

```
export default KanbanColumn;
```

Шаг 6. Создание компонента KanbanCard

src/components/kanban/KanbanCard.jsx:

```
import React from 'react';
import { useSortable } from '@dnd-kit/sortable';
import { CSS } from '@dnd-kit/utilities';
import {
  Paper,
  Typography,
  Box,
  Chip,
  IconButton,
  Tooltip,
} from '@mui/material';
import {
  DragIndicator as DragIcon,
  AccessTime as AccessTimeIcon,
  Flag as FlagIcon,
} from '@mui/icons-material';
import { useNavigate } from 'react-router-dom';

const priorityColors = {
  Low: { color: '#2ecc71', label: 'Низкий' },
  Medium: { color: '#f39c12', label: 'Средний' },
  High: { color: '#e74c3c', label: 'Высокий' },
};

const KanbanCard = ({ task, isDragging = false }) => {
  const navigate = useNavigate();
  const {
    attributes,
```

```
    listeners,
    setNodeRef,
    transform,
    transition,
    isDragging: isSortableDragging
  } = useSortable({ id: task.id.toString() });

  const style = {
    transform: CSS.Transform.toString(transform),
    transition,
    opacity: isDragging || isSortableDragging ? 0.5 : 1,
    cursor: 'grab',
  };

  const formatDate = (dateString) => {
    if (!dateString) return null;
    const date = new Date(dateString);
    const today = new Date();
    const tomorrow = new Date(today);
    tomorrow.setDate(tomorrow.getDate() + 1);

    if (date.toDateString() === today.toDateString()) {
      return 'Сегодня';
    }
    if (date.toDateString() === tomorrow.toDateString()) {
      return 'Завтра';
    }
    return date.toLocaleDateString('ru-RU', {
      day: 'numeric',
      month: 'short',
    });
  };
```

```
  const isOverdue = task.deadline && new Date(task.deadline) < new Date() && task.status !== 'Completed';

  const handleCardClick = (e) => {
    // Предотвращаем навигацию при клике на кнопки
    if (e.target.closest('.MuiIconButton-root')) return;
    navigate(`/tasks/${task.id}`);
  };

  return (
    <Paper
      ref={setNodeRef}
      style={style}
      {...attributes}
      {...listeners}
      onClick={handleCardClick}
      sx={{
        p: 1.5,
        borderRadius: 2,
        bgcolor: 'white',
        cursor: 'pointer',
        transition: 'box-shadow 0.2s',
        '&:hover': {
          boxShadow: 3,
        },
        ...(task.status === 'Completed' && {
          opacity: 0.7,
          textDecoration: 'line-through',
        }),
      }}
    >
```

```
<Box display="flex" alignItems="flex-start" gap={1}>
  <DragIcon sx={{ color: '#bdc3c7', fontSize: 20, cursor: 'grab' }} />
  <Typography
    variant="body2"
    sx={{
      flex: 1,
      fontWeight: 'medium',
      ...(task status === 'Completed' && {
        textDecoration: 'line-through',
      }),
    }}
  >
    {task title}
  </Typography>
</Box>

{task description && (
  <Typography
    variant="caption"
    color="text.secondary"
    sx={{
      display: 'block',
      mt: 1,
      ml: 4,
      overflow: 'hidden',
      textOverflow: 'ellipsis',
      whiteSpace: 'nowrap',
    }}
  >
    {task description length > 60
      ? `${task description substring(0, 60)}...`
      : task description}
```

```
    </Typography>
  )}

  <Box display="flex" flexWrap="wrap" gap={0.5} sx={{ mt: 1, ml: 4 }}>
    <Chip
      size="small"
      icon={<FlagIcon sx={{ fontSize: 12 }} />}
      label={priorityColors[task.priority]?.label || task.priority}
      sx={{
        bgcolor: priorityColors[task.priority]?.color,
        color: 'white',
        height: 20,
        '& .MuiChip-label': { fontSize: '0.65rem', px: 1 },
        '& .MuiChip-icon': { fontSize: 12, ml: 0.5 },
      }}
    />

    {task.deadline && (
      <Chip
        size="small"
        icon={<AccessTimeIcon sx={{ fontSize: 12 }} />}
        label={formatDate(task.deadline)}
        sx={{
          bgcolor: isOverdue ? '#e74c3c' : '#95a5a6',
          color: 'white',
          height: 20,
          '& .MuiChip-label': { fontSize: '0.65rem', px: 1 },
          '& .MuiChip-icon': { fontSize: 12, ml: 0.5 },
        }}
      />
    )}
```

```
            {task.project && (
              <Chip
                size="small"
                label={task.project.name}
                sx={{
                  bgcolor: task.project.color || '#9b59b6',
                  color: 'white',
                  height: 20,
                  '& .MuiChip-label': { fontSize: '0.65rem', px: 1 },
                }}
              />
            )}
          </Box>
        </Paper>
    );
};

export default KanbanCard;
```

Шаг 7. Обновление навигации

src/components/common/Navbar.jsx (дополнение):

```
// Добавляем пункты меню для календаря и Kanban
const menuItems = [
    { text: 'Панель управления', icon: <DashboardIcon />, path: '/' },
    { text: 'Задачи', icon: <TasksIcon />, path: '/tasks' },
    { text: 'Доска', icon: <ViewColumnIcon />, path: '/kanban' },
    { text: 'Календарь', icon: <CalendarIcon />, path: '/calendar' },
    { text: 'Статистика', icon: <StatsIcon />, path: '/statistics' },
];
```

Шаг 8. Добавление маршрутов

src/App.jsx (дополнение):

```
import TaskCalendar from './components/calendar/TaskCalendar';
import KanbanBoard from './components/kanban/KanbanBoard';

// ... внутри Routes
<Route path="/calendar" element={
  <PrivateRoute>
    <TaskCalendar />
  </PrivateRoute>
} />
<Route path="/kanban" element={
  <PrivateRoute>
    <KanbanBoard />
  </PrivateRoute>
} />
```

Контрольные вопросы

1. Какие библиотеки можно использовать для реализации календаря в React?
2. Как работает drag-and-drop в @dnd-kit?
3. Как организована группировка задач по статусам в Kanban-доске?
4. Как обрабатывается изменение даты задачи через перетаскивание в календаре?
5. Как оптимизировать производительность при большом количестве задач на доске?

Критерии оценки

Критерий	Баллы
Реализован компонент календаря (FullCalendar)	3

Критерий	Баллы
Реализована возможность drag-and-drop для изменения дат	2
Реализована Kanban-доска с колонками по статусам	3
Реализовано перетаскивание карточек между колонками	2
Итого	10

Результат выполнения

1. После выполнения лабораторной работы №11 студенты должны получить:
2. Календарь с отображением задач по датам
3. Drag-and-drop для изменения дедлайнов задач
4. Kanban-доску для визуального управления задачами по статусам
5. Возможность быстрого изменения статуса через перетаскивание

Лабораторная работа №12

Реальное время. WebSockets и уведомления

Цель работы: Реализация real-time функционала через WebSockets: уведомления о дедлайнах, обновление статусов задач в реальном времени, отображение онлайн-статуса пользователей.

Задачи

№	Задача
1	Настроить WebSocket соединение между клиентом и сервером
2	Реализовать отправку уведомлений о приближающихся дедлайнах

№	Задача
3	Реализовать real-time обновление списка задач при изменениях
4	Добавить браузерные push-уведомления (Notification API)
5	Реализовать отображение онлайн-статуса пользователей

Теоретическая часть

WebSocket - протокол полнодуплексной связи поверх TCP, обеспечивающий постоянное соединение между клиентом и сервером.

Отличие от HTTP:

1. HTTP - запрос-ответ (клиент инициирует)
2. WebSocket - двусторонняя связь (сервер может отправлять данные без запроса)

Django Channels - расширение Django для поддержки WebSocket, асинхронных задач и других протоколов.

Notification API - браузерный API для отображения системных уведомлений пользователю.

SignalR (для ASP.NET)

SignalR - библиотека для ASP.NET Core, упрощающая добавление real-time функциональности.

Ход выполнения работы

Шаг 1. Установка необходимых пакетов (Django)

```
pip install channels channels-redis
pip install daphne
```

Шаг 2. Настройка Django Channels

settings.py:

```
INSTALLED_APPS = [
    # ...
    'channels',
]

# Настройка Channels
ASGI_APPLICATION = 'smart_diary.asgi.application'

# Redis для каналов (для production)
CHANNEL_LAYERS = {
    'default': {
        'BACKEND': 'channels_redis.core.RedisChannelLayer',
        'CONFIG': {
            'hosts': [('127.0.0.1', 6379)],
        },
    },
}

# Для разработки без Redis (только один процесс)
# CHANNEL_LAYERS = {
#     'default': {
#         'BACKEND': 'channels.layers.InMemoryChannelLayer',
#     },
# }
```

asgi.py:

```
import os
from django.core.asgi import get_asgi_application
from channels.routing import ProtocolTypeRouter, URLRouter
from channels.auth import AuthMiddlewareStack
from channels.security.websocket import AllowedHostsOriginValidator
from tasks.routing import websocket_urlpatterns
```

```
os environ setdefault('DJANGO_SETTINGS_MODULE', 'smart_diary.settings')

application = ProtocolTypeRouter({
    'http': get_asgi_application(),
    'websocket': AllowedHostsOriginValidator(
        AuthMiddlewareStack(
            URLRouter(websocket_urlpatterns)
        )
    ),
})
```

Шаг 3. Создание WebSocket Consumer

tasks/consumers.py:

```
import json
from channels generic websocket import AsyncWebsocketConsumer
from channels db import database_sync_to_async
from django utils import timezone
from datetime import timedelta
from . models import Task, User

class TaskConsumer(AsyncWebsocketConsumer):
    """Consumer для real-time уведомлений о задачах"""

    async def connect(self):
        self user = self scope['user']

        if self user is_anonymous:
            await self close()
            return

        self room_group_name = f'user_{self user id}'
```

```
        # Присоединение к группе пользователя
        await self.channel_layer.group_add(
            self.room_group_name,
            self.channel_name
        )

        await self.accept()

        # Отправка онлайн-статуса
        await self.update_online_status(True)

        # Отправка непрочитанных уведомлений
        await self.send_pending_notifications()

    async def disconnect(self, close_code):
        if hasattr(self, 'room_group_name'):
            await self.channel_layer.group_discard(
                self.room_group_name,
                self.channel_name
            )
            await self.update_online_status(False)

    async def receive(self, text_data):
        """Обработка сообщений от клиента"""
        data = json.loads(text_data)
        message_type = data.get('type')

        if message_type == 'ping':
            await self.send(text_data=json.dumps({
                'type': 'pong',
                'timestamp': timezone.now().isoformat(),
```

```
            }))
        elif message_type == 'notification_read':
            await self.mark_notification_read(data.get('notification_id'))

    async def task_notification(self, event):
        """Отправка уведомления о задаче"""
        await self.send(text_data=json.dumps({
            'type': 'task_notification',
            'notification_id': event['notification_id'],
            'task_id': event['task_id'],
            'task_title': event['task_title'],
            'message': event['message'],
            'deadline': event.get('deadline'),
            'created_at': event['created_at'],
        }))

    async def task_updated(self, event):
        """Отправка обновления задачи"""
        await self.send(text_data=json.dumps({
            'type': 'task_updated',
            'task_id': event['task_id'],
            'action': event['action'],
            'task_data': event.get('task_data'),
        }))

    async def user_status(self, event):
        """Отправка статуса пользователя"""
        await self.send(text_data=json.dumps({
            'type': 'user_status',
            'user_id': event['user_id'],
            'username': event['username'],
            'is_online': event['is_online'],
```

```
            'last_seen': event.get('last_seen'),
        }))

    @database_sync_to_async
    def update_online_status(self, is_online):
        """Обновление онлайн-статуса пользователя"""
        if hasattr(self.user, 'profile'):
            self.user.profile.is_online = is_online
            self.user.profile.last_seen = timezone.now()
            self.user.profile.save()

    @database_sync_to_async
    def send_pending_notifications(self):
        """Отправка непрочитанных уведомлений"""
        from .models import Notification
        notifications = Notification.objects.filter(
            user=self.user,
            is_read=False,
            created_at__gte=timezone.now() - timedelta(days=1)
        )[:20]

        for notif in notifications:
            self.channel_layer.group_send(
                self.room_group_name,
                {
                    'type': 'task_notification',
                    'notification_id': notif.id,
                    'task_id': notif.task_id,
                    'task_title': notif.task_title,
                    'message': notif.message,
                    'created_at': notif.created_at.isoformat(),
                }
```

```
        )

    @database_sync_to_async
    def mark_notification_read(self, notification_id):
        """Отметка уведомления как прочитанного"""
        from .models import Notification
        Notification.objects.filter(
            id=notification_id,
            user=self.user
        ).update(is_read=True)
```

Шаг 4. Создание маршрутизации WebSocket

tasks/routing.py:

```
from django.urls import re_path
from . import consumers

websocket_urlpatterns = [
    re_path(r'ws/tasks/$', consumers.TaskConsumer.as_asgi()),
]
```

Шаг 5. Создание моделей для уведомлений

tasks/models.py (дополнение):

```
class Notification(models.Model):
    """Модель уведомления"""
    user = models.ForeignKey(
        User,
        on_delete=models.CASCADE,
        related_name='notifications'
    )
    task = models.ForeignKey(
```

```
    Task,
    on_delete=models.CASCADE,
    null=True,
    blank=True,
    related_name='notifications'
)
task_title = models.CharField(max_length=200)
message = models.TextField()
is_read = models.BooleanField(default=False)
created_at = models.DateTimeField(auto_now_add=True)

class Meta:
    ordering = ['-created_at']
    indexes = [
        models.Index(fields=['user', 'is_read']),
        models.Index(fields=['created_at']),
    ]

def __str__(self):
    return f'Notification for {self.user.username}: {self.message[:50]}'
```

Шаг 6. Создание сигналов для автоматических уведомлений

tasks/signals.py:

```
from django.db.models.signals import post_save
from django.dispatch import receiver
from django.utils import timezone
from datetime import timedelta
from channels.layers import get_channel_layer
from asgiref.sync import async_to_sync
from .models import Task, Notification
```

```python
@receiver(post_save, sender=Task)
def task_updated_notification(sender, instance, created, **kwargs):
    """Отправка уведомления при создании/обновлении задачи"""
    channel_layer = get_channel_layer()

    if created:
        message = f'Создана новая задача: "{instance.title}"'
        action = 'created'
    else:
        message = f'Задача "{instance.title}" была обновлена'
        action = 'updated'

    # Создание уведомления в БД
    notification = Notification.objects.create(
        user=instance.user,
        task=instance,
        task_title=instance.title,
        message=message
    )

    # Отправка через WebSocket
    async_to_sync(channel_layer.group_send)(
        f'user_{instance.user.id}',
        {
            'type': 'task_notification',
            'notification_id': notification.id,
            'task_id': instance.id,
            'task_title': instance.title,
            'message': message,
            'created_at': timezone.now().isoformat(),
        }
    )
```

```
def check_deadline_notifications():
    """Проверка дедлайнов и отправка напоминаний"""
    channel_layer = get_channel_layer()
    now = timezone.now()

    # Задачи с дедлайном в ближайший час
    upcoming_tasks = Task.objects.filter(
        deadline__gte=now,
        deadline__lte=now + timedelta(hours=1),
        status__in=['New', 'InProgress'],
        notified=False
    )

    for task in upcoming_tasks:
        notification = Notification.objects.create(
            user=task.user,
            task=task,
            task_title=task.title,
            message=f'⚠ Дедлайн задачи "{task.title}" приближается! Осталось менее часа.'
        )

        task.notified = True
        task.save(update_fields=['notified'])

        async_to_sync(channel_layer.group_send)(
            f'user_{task.user.id}',
            {
                'type': 'task_notification',
                'notification_id': notification.id,
                'task_id': task.id,
```

```
            'task_title': task.title,
            'message': notification_message,
            'deadline': task.deadline.isoformat(),
            'created_at': now.isoformat(),
        }
    )
```

Шаг 7. Создание клиентского WebSocket сервиса

src/services/websocketService.js:

```
class WebSocketService {
  constructor() {
    this.socket = null;
    this.listeners = new Map();
    this.reconnectAttempts = 0;
    this.maxReconnectAttempts = 5;
    this.reconnectDelay = 3000;
    this.isConnecting = false;
  }

  connect() {
    if (this.isConnecting) return;
    if (this.socket?.readyState === WebSocket.OPEN) return;

    this.isConnecting = true;
    const token = localStorage.getItem('access_token');
    const wsUrl = `${import.meta.env.VITE_WS_URL ||
'ws://localhost:8000'}/ws/tasks/?token=${token}`;

    try {
      this.socket = new WebSocket(wsUrl);
```

```
        this.socket.onopen = () => {
            console.log('WebSocket connected');
            this.reconnectAttempts = 0;
            this.isConnecting = false;
            this.startHeartbeat();
            this.trigger('connected', null);
        };

        this.socket.onmessage = (event) => {
            const data = JSON.parse(event.data);
            this.handleMessage(data);
        };

        this.socket.onclose = (event) => {
            console.log('WebSocket closed:', event.code, event.reason);
            this.isConnecting = false;
            this.stopHeartbeat();
            this.trigger('disconnected', event);
            this.reconnect();
        };

        this.socket.onerror = (error) => {
            console.error('WebSocket error:', error);
            this.trigger('error', error);
        };
    } catch (error) {
        console.error('Failed to connect WebSocket:', error);
        this.isConnecting = false;
        this.reconnect();
    }
}
```

```
  reconnect() {
    if (this.reconnectAttempts >= this.maxReconnectAttempts) {
      console.log('Max reconnection attempts reached');
      return;
    }

    this.reconnectAttempts++;
    console.log(`Reconnecting... Attempt
${this.reconnectAttempts}/${this.maxReconnectAttempts}`);

    setTimeout(() => {
      this.connect();
    }, this.reconnectDelay);
  }

  disconnect() {
    if (this.socket) {
      this.socket.close();
      this.socket = null;
    }
    this.stopHeartbeat();
  }

  send(data) {
    if (this.socket?.readyState === WebSocket.OPEN) {
      this.socket.send(JSON.stringify(data));
    } else {
      console.warn('WebSocket is not open');
    }
  }

  handleMessage(data) {
```

```
        switch (data type) {
            case 'task_notification':
                this.trigger('notification', data);
                // Показать браузерное уведомление
                this.showBrowserNotification(data message, data task_title);
                break;

            case 'task_updated':
                this.trigger('taskUpdated', {
                    taskId: data task_id,
                    action: data action,
                    taskData: data task_data,
                });
                break;

            case 'user_status':
                this.trigger('userStatus', {
                    userId: data user_id
                    username: data username,
                    isOnline: data is_online,
                    lastSeen: data last_seen,
                });
                break;

            case 'pong':
                this.trigger('pong', data);
                break;

            default:
                console.log('Unknown message type:', data.type);
        }
    }
```

```
startHeartbeat() {
    this.heartbeatInterval = setInterval(() => {
        this.send({ type: 'ping' });
    }, 30000); // каждые 30 секунд
}

stopHeartbeat() {
    if (this.heartbeatInterval) {
        clearInterval(this.heartbeatInterval);
        this.heartbeatInterval = null;
    }
}

showBrowserNotification(title, body) {
    if (!('Notification' in window)) {
        console.log('This browser does not support notifications');
        return;
    }

    if (Notification.permission === 'granted') {
        new Notification(title, {
            body: body,
            icon: '/icon-192.png',
            silent: false,
        });
    } else if (Notification.permission !== 'denied') {
        Notification.requestPermission();
    }
}

on(event, callback) {
```

```
    if (!this.listeners.has(event)) {
      this.listeners.set(event, []);
    }
    this.listeners.get(event).push(callback);
  }

  off(event, callback) {
    if (!this.listeners.has(event)) return;
    const callbacks = this.listeners.get(event);
    const index = callbacks.indexOf(callback);
    if (index !== -1) callbacks.splice(index, 1);
  }

  trigger(event, data) {
    if (!this.listeners.has(event)) return;
    this.listeners.get(event).forEach(callback => callback(data));
  }
}

export const wsService = new WebSocketService();
```

Шаг 8. Создание компонента уведомлений

src/components/notifications/NotificationCenter.jsx:

```
import React, { useState, useEffect } from 'react';
import {
  IconButton,
  Badge,
  Popover,
  List,
  ListItem,
  ListItemText,
```

```
    ListItemAvatar,
    Avatar,
    Typography,
    Box,
    Divider,
    Button,
    Chip,
} from '@mui/material';
import {
    Notifications as NotificationsIcon,
    Task as TaskIcon,
    CheckCircle as CheckCircleIcon,
    Warning as WarningIcon,
    Info as InfoIcon,
} from '@mui/icons-material';
import { useNavigate } from 'react-router-dom';
import { wsService } from '../../services/websocketService';

const getNotificationIcon = (type, isRead) => {
    const color = isRead ? '#95a5a6' : '#3498db';

    switch (type) {
        case 'deadline':
            return <WarningIcon sx={{ color: '#e74c3c' }} />;
        case 'task_created':
            return <TaskIcon sx={{ color }} />;
        case 'task_updated':
            return <InfoIcon sx={{ color }} />;
        default:
            return <TaskIcon sx={{ color }} />;
    }
};
```

```
const NotificationCenter = () => {
  const navigate = useNavigate();
  const [notifications, setNotifications] = useState([]);
  const [unreadCount, setUnreadCount] = useState(0);
  const [anchorEl, setAnchorEl] = useState(null);

  useEffect(() => {
    // Загрузка уведомлений из API
    fetchNotifications();

    // Подписка на WebSocket уведомления
    wsService.on('notification', handleNewNotification);

    // Запрос разрешения на уведомления
    if ('Notification' in window && Notification.permission === 'default') {
      Notification.requestPermission();
    }

    return () => {
      wsService.off('notification', handleNewNotification);
    };
  }, []);

  const fetchNotifications = async () => {
    try {
      const response = await fetch('/api/notifications/', {
        headers: {
          'Authorization': `Bearer ${localStorage.getItem('access_token')}`,
        },
      });
      const data = await response.json();
```

```
      setNotifications(data.results || []);
      setUnreadCount(data.results?.filter(n => !n.is_read).length || 0);
    } catch (error) {
      console.error('Error fetching notifications:', error);
    }
  };

  const handleNewNotification = (notification) => {
    setNotifications(prev => [notification, ...prev]);
    setUnreadCount(prev => prev + 1);
  };

  const handleOpen = (event) => {
    setAnchorEl(event.currentTarget);
  };

  const handleClose = () => {
    setAnchorEl(null);
  };

  const handleNotificationClick = async (notification) => {
    // Отметка как прочитанного
    if (!notification.is_read) {
      await markAsRead(notification.id);
      setNotifications(prev =>
        prev.map(n =>
          n.id === notification.id ? { ...n, is_read: true } : n
        )
      );
      setUnreadCount(prev => prev - 1);
    }
```

```
    // Переход к задаче
    if (notification.task_id) {
        navigate(`/tasks/${notification.task_id}`);
    }

    handleClose();
};

const markAsRead = async (id) => {
    try {
        await fetch(`/api/notifications/${id}/read/`, {
            method: 'POST',
            headers: {
                'Authorization': `Bearer ${localStorage.getItem('access_token')}`,
            },
        });
    } catch (error) {
        console.error('Error marking notification as read:', error);
    }
};

const markAllAsRead = async () => {
    try {
        await fetch('/api/notifications/mark-all-read/', {
            method: 'POST',
            headers: {
                'Authorization': `Bearer ${localStorage.getItem('access_token')}`,
            },
        });
        setNotifications(prev =>
            prev.map(n => ({ ...n, is_read: true }))
        );
```

```
      setUnreadCount(0);
    } catch (error) {
      console.error('Error marking all as read:', error);
    }
  };

  const open = Boolean(anchorEl);
  const id = open ? 'notification-popover' : undefined;

  return (
    <>
      <IconButton color="inherit" onClick={handleOpen}>
        <Badge badgeContent={unreadCount} color="error">
          <NotificationsIcon />
        </Badge>
      </IconButton>

      <Popover
        id={id}
        open={open}
        anchorEl={anchorEl}
        onClose={handleClose}
        anchorOrigin={{
          vertical: 'bottom',
          horizontal: 'right',
        }}
        transformOrigin={{
          vertical: 'top',
          horizontal: 'right',
        }}
        PaperProps={{
          sx: { width: 380, maxHeight: 500 },
```

```
    }}
>
    <Box sx={{ p: 2, borderBottom: '1px solid #eee' }}>
      <Typography variant="h6" component="span">
        Уведомления
      </Typography>
      {unreadCount > 0 && (
        <Button
          size="small"
          onClick={markAllAsRead}
          sx={{ float: 'right' }}
        >
          Прочитать все
        </Button>
      )}
    </Box>

    <List sx={{ p: 0 }}>
      {notifications.length === 0 ? (
        <ListItem>
          <ListItemText
            primary="Нет уведомлений"
            secondary="У вас пока нет новых уведомлений"
            primaryTypographyProps={{ align: 'center' }}
            secondaryTypographyProps={{ align: 'center' }}
          />
        </ListItem>
      ) : (
        notifications.map((notification, index) => (
          <React.Fragment key={notification.id}>
            {index > 0 && <Divider />}
            <ListItem
```

```
                button
                onClick={() => handleNotificationClick(notification)}
                sx={{
                  bgcolor: notification.is_read ? 'transparent' : '#f0f7ff',
                  '&:hover': { bgcolor: '#e3f2fd' },
                }}
              >
                <ListItemAvatar>
                  <Avatar sx={{ bgcolor: 'transparent' }}>
                    {getNotificationIcon(notification.type, notification.is_read)}
                  </Avatar>
                </ListItemAvatar>
                <ListItemText
                  primary={
                    <Typography variant="body2" fontWeight={notification.is_read ? 'normal' :
'bold'}>
                      {notification.message}
                    </Typography>
                  }
                  secondary={
                    <Typography variant="caption" color="text.secondary">
                      {new Date(notification.created_at).toLocaleString('ru-RU')}
                    </Typography>
                  }
                />
                {!notification.is_read && (
                  <Box
                    sx={{
                      width: 8,
                      height: 8,
                      borderRadius: '50%',
                      bgcolor: '#3498db',
```

```
                                ml: 1,
                              }}
                            />
                          ))}
                        </ListItem>
                      </React Fragment>
                    ))
                  )}
                </List>

                {notifications length > 5 && (
                  <Box sx={{ p: 1, borderTop: '1px solid #eee', textAlign: 'center' }}>
                    <Button size="small" onClick={handleClose}>
                      Показать все
                    </Button>
                  </Box>
                )}
              </Popover>
            </>
          );
};

export default NotificationCenter;
```

Шаг 9. Интеграция WebSocket в приложение

src/App.jsx (дополнение):

```
import { useEffect } from 'react';
import { wsService } from './services/websocketService';
import { useAuth } from './context/AuthContext';
import NotificationCenter from './components/notifications/NotificationCenter';
```

```
const AppContent = () => {
  const { isAuthenticated } = useAuth();

  useEffect(() => {
    if (isAuthenticated) {
      wsService.connect();
    } else {
      wsService.disconnect();
    }

    return () => {
      wsService.disconnect();
    };
  }, [isAuthenticated]);

  return (
    <>
      {isAuthenticated && (
        <>
          <Navbar user={user} onLogout={logout} />
          <NotificationCenter />
        </>
      )}
      {/* ... остальное содержимое */}
    </>
  );
};
```

Шаг 10. Реализация задачи для проверки дедлайнов (Celery)

tasks/tasks.py:

```
from celery import shared_task
from django.utils import timezone
```

```python
from datetime import timedelta
from channels layers import get_channel_layer
from asgiref sync import async_to_sync
from  models import Task, Notification

@shared_task
def check_deadlines():
    """Периодическая задача для проверки дедлайнов"""
    channel_layer = get_channel_layer()
    now = timezone now()

    # Задачи с дедлайном в ближайший час (не уведомленные)
    upcoming_tasks = Task objects filter(
        deadline__gte=now,
        deadline__lte=now + timedelta(hours=1),
        status__in=['New', 'InProgress'],
        deadline_notified=False
    )

    for task in upcoming_tasks:
        notification = Notification objects create(
            user=task user,
            task=task,
            task_title=task title,
            message=f'⚠ Дедлайн задачи "{task title}" приближается!',
            type='deadline'
        )

        task deadline_notified = True
        task save(update_fields=['deadline_notified'])

        async_to_sync(channel_layer group_send)(
```

```
            f'user_{task.user.id}',
            {
                'type': 'task_notification',
                'notification_id': notification.id,
                'task_id': task.id,
                'task_title': task.title,
                'message': notification.message,
                'created_at': now.isoformat(),
            }
        )


@shared_task
def update_user_offline_status():
    """Обновление статуса offline для неактивных пользователей"""
    from .models import UserProfile
    timeout = timezone.now() - timedelta(minutes=5)
    UserProfile.objects.filter(
        is_online=True,
        last_seen__lt=timeout
    ).update(is_online=False)
```

Контрольные вопросы

1. В чем отличие WebSocket от HTTP протокола?
2. Как работает аутентификация в Django Channels?
3. Как организована отправка уведомлений через WebSocket?
4. Какие типы уведомлений можно реализовать?
5. Как работает механизм периодической проверки дедлайнов?
6. Что такое Celery и для чего он используется?

Критерии оценки

Критерий	Баллы
Настроено WebSocket соединение	2
Реализована отправка уведомлений о дедлайнах	2
Реализовано real-time обновление списка задач	2
Добавлены браузерные push-уведомления	2
Реализован компонент NotificationCenter	2
Итого	10

Результат выполнения

1. После выполнения лабораторной работы №12 студенты должны получить:
2. WebSocket соеди
3. нение между React и Django
4. Real-time уведомления о приближающихся дедлайнах
5. Автоматическое обновление списка задач при изменениях
6. Браузерные push-уведомления
7. Центр уведомлений с историей

Лабораторная работа №13

Совместная работа. Шаринг задач и проектов

Цель работы: Реализация функционала совместной работы над задачами: приглашение пользователей в проекты, назначение ролей, комментарии к задачам, история изменений.

Задачи

№	Задача
1	Реализовать систему приглашений пользователей в проекты (шаринг)
2	Добавить роли в проекте (владелец, редактор, читатель)
3	Реализовать комментарии к задачам
4	Добавить систему логов действий (история изменений задачи)
5	Реализовать уведомления о назначении задачи и приглашении в проект

Теоретическая часть

Модель доступа к проектам

Роль	**Права**
Владелец	Полный доступ: управление проектом, приглашение пользователей, изменение ролей, удаление
Редактор	Создание, редактирование, удаление задач в проекте
Читатель	Только просмотр задач проекта

Шаринг (Sharing)

Шаринг - механизм предоставления доступа к ресурсу другим пользователям.

Способы приглашения:

1. По email (отправка ссылки-приглашения)
2. По username (поиск пользователя в системе)
3. По ссылке-приглашению с токеном

4. Аудит действий (Activity Log)

Activity Log - запись всех действий пользователей над задачей.

Типы действий:

1. Создание задачи
2. Изменение статуса
3. Изменение приоритета
4. Изменение дедлайна
5. Добавление/удаление тегов
6. Добавление комментария
7. Назначение пользователя

Ход выполнения работы

Шаг 1. Создание моделей для совместной работы

tasks/models.py (дополнение):

```
python
from django contrib auth import get_user_model

User = get_user_model()



class ProjectMember(models Model):
    """Участник проекта с определенной ролью"""

    class Role(models TextChoices):
        OWNER = 'owner', 'Владелец'
        EDITOR = 'editor', 'Редактор'
        VIEWER = 'viewer', 'Читатель'

    project = models ForeignKey(
        Project,
        on_delete=models CASCADE,
```

```
        related_name='members'
    )
    user = models.ForeignKey(
        User,
        on_delete=models.CASCADE,
        related_name='project_memberships'
    )
    role = models.CharField(
        max_length=20,
        choices=Role.choices,
        default=Role.VIEWER
    )
    invited_by = models.ForeignKey(
        User,
        on_delete=models.SET_NULL,
        null=True,
        related_name='sent_invites'
    )
    invited_at = models.DateTimeField(auto_now_add=True)
    accepted_at = models.DateTimeField(null=True, blank=True)

    class Meta:
        unique_together = ['project', 'user']
        indexes = [
            models.Index(fields=['project', 'user']),
            models.Index(fields=['user']),
        ]

    def __str__(self):
        return f"{self.user.username} - {self.project.name} ({self.role})"
```

```
class TaskAssignment(models Model):
    """Назначение задачи пользователю"""
    task = models ForeignKey(
        Task,
        on_delete=models CASCADE,
        related_name='assignments'
    )
    user = models ForeignKey(
        User,
        on_delete=models CASCADE,
        related_name='assigned_tasks'
    )
    assigned_by = models ForeignKey(
        User,
        on_delete=models SET_NULL,
        null=True,
        related_name='assigned_by'
    )
    assigned_at = models DateTimeField(auto_now_add=True)

    class Meta:
        unique_together = ['task', 'user']

    def __str__(self):
        return f"{self user username} assigned to {self task title}"


class Comment(models Model):
    """Комментарий к задаче"""
    task = models ForeignKey(
        Task,
        on_delete=models CASCADE,
```

```
        related_name='comments'
    )
    author = models.ForeignKey(
        User,
        on_delete=models.CASCADE,
        related_name='comments'
    )
    content = models.TextField(max_length=2000)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    parent = models.ForeignKey(
        'self',
        on_delete=models.CASCADE,
        null=True,
        blank=True,
        related_name='replies'
    )

    class Meta:
        ordering = ['created_at']
        indexes = [
            models.Index(fields=['task', 'created_at']),
        ]

    def __str__(self):
        return f"Comment by {self.author.username} on {self.task.title}"


class ActivityLog(models.Model):
    """Лог действий над задачей"""

    class ActionType(models.TextChoices):
```

```
    CREATED = 'created', 'Создана'
    UPDATED = 'updated', 'Обновлена'
    STATUS_CHANGED = 'status_changed', 'Изменён статус'
    PRIORITY_CHANGED = 'priority_changed', 'Изменён приоритет'
    DEADLINE_CHANGED = 'deadline_changed', 'Изменён дедлайн'
    ASSIGNED = 'assigned', 'Назначена'
    UNASSIGNED = 'unassigned', 'Снято назначение'
    COMMENT_ADDED = 'comment_added', 'Добавлен комментарий'
    TAG_ADDED = 'tag_added', 'Добавлен тег'
    TAG_REMOVED = 'tag_removed', 'Удалён тег'
    PROJECT_CHANGED = 'project_changed', 'Изменён проект'

task = models.ForeignKey(
    Task,
    on_delete=models.CASCADE,
    related_name='activities'
)
user = models.ForeignKey(
    User,
    on_delete=models.SET_NULL,
    null=True,
    related_name='activities'
)
action_type = models.CharField(max_length=50, choices=ActionType.choices)
old_value = models.TextField(blank=True, null=True)
new_value = models.TextField(blank=True, null=True)
created_at = models.DateTimeField(auto_now_add=True)

class Meta:
    ordering = ['-created_at']
    indexes = [
        models.Index(fields=['task', '-created_at']),
```

```
            models Index(fields=['user']),
        ]

    def __str__(self):
        return f"{self.user.username} - {self.action_type} task {self.task.id}"


class ProjectInvite(models.Model):
    """Приглашение в проект"""

    class Status(models.TextChoices):
        PENDING = 'pending', 'Ожидает'
        ACCEPTED = 'accepted', 'Принято'
        REJECTED = 'rejected', 'Отклонено'
        EXPIRED = 'expired', 'Истекло'

    project = models.ForeignKey(
        Project,
        on_delete=models.CASCADE,
        related_name='invites'
    )
    email = models.EmailField()
    invited_by = models.ForeignKey(
        User,
        on_delete=models.CASCADE,
        related_name='sent_project_invites'
    )
    role = models.CharField(
        max_length=20,
        choices=ProjectMember.Role.choices,
        default=ProjectMember.Role.VIEWER
    )
```

```
    token = models.CharField(max_length=64, unique=True)
    status = models.CharField(max_length=20, choices=Status.choices, default=Status.PENDING)
    created_at = models.DateTimeField(auto_now_add=True)
    expires_at = models.DateTimeField()

    def __str__(self):
        return f"Invite to {self.project.name} for {self.email}"
```

Шаг 2. Создание миграций

```
python manage.py makemigrations
python manage.py migrate
```

Шаг 3. Создание сервисов для совместной работы

tasks/services/collaboration_service.py:

```
import uuid
from datetime import timedelta
from django.utils import timezone
from django.core.mail import send_mail
from django.conf import settings
from django.db import transaction
from ..models import (
    Project, ProjectMember, TaskAssignment, Comment,
    ActivityLog, ProjectInvite, User, Task
)


class CollaborationService:
    """Сервис для совместной работы"""

    # ============================================
    # Управление участниками проекта
    # ============================================
```

```python
@staticmethod
def add_project_member(project_id, user_id, role, invited_by_id):
    """Добавление участника в проект"""
    project = Project.objects.get(id=project_id)
    user = User.objects.get(id=user_id)

    member, created = ProjectMember.objects.get_or_create(
        project=project,
        user=user,
        defaults={
            'role': role,
            'invited_by_id': invited_by_id,
            'accepted_at': timezone.now()
        }
    )

    if not created:
        member.role = role
        member.save()

    return member

@staticmethod
def remove_project_member(project_id, user_id):
    """Удаление участника из проекта"""
    ProjectMember.objects.filter(project_id=project_id, user_id=user_id).delete()

@staticmethod
def get_project_members(project_id):
    """Получение всех участников проекта"""
    return ProjectMember.objects.filter(project_id=project_id).select_related('user')
```

```python
@staticmethod
def check_user_permission(project_id, user_id, required_role):
    """Проверка прав пользователя в проекте"""
    try:
        member = ProjectMember.objects.get(project_id=project_id, user_id=user_id)
        role_priority = {
            'owner': 3,
            'editor': 2,
            'viewer': 1
        }
        return role_priority[member.role] >= role_priority[required_role]
    except ProjectMember.DoesNotExist:
        return False

# ============================================
# Приглашения в проект
# ============================================

@staticmethod
def create_invite(project_id, email, role, invited_by_id):
    """Создание приглашения в проект"""
    token = uuid.uuid4().hex
    expires_at = timezone.now() + timedelta(days=7)

    invite = ProjectInvite.objects.create(
        project_id=project_id,
        email=email,
        invited_by_id=invited_by_id,
        role=role,
        token=token,
        expires_at=expires_at
```

```
        )

        # Отправка email с приглашением
        invite_link = f"{settings.FRONTEND_URL}/invite/{token}"
        send_mail(
            subject=f'Приглашение в проект {invite.project.name}',
            message=f'Вас пригласили в проект "{invite.project.name}".\n'
                    f'Ссылка для принятия приглашения: {invite_link}\n'
                    f'Приглашение действительно до {expires_at.strftime("%d.%m.%Y %H:%M")}',
            from_email=settings.DEFAULT_FROM_EMAIL,
            recipient_list=[email],
            fail_silently=False,
        )

        return invite

    @staticmethod
    def accept_invite(token, user):
        """Принятие приглашения"""
        try:
            invite = ProjectInvite.objects.get(
                token=token,
                status=ProjectInvite.Status.PENDING,
                expires_at__gt=timezone.now()
            )
        except ProjectInvite.DoesNotExist:
            return None

        # Проверка, не зарегистрирован ли пользователь с другим email
        if user.email != invite.email:
            # Можно либо создать связь, либо отклонить
            # Здесь принимаем решение: обновляем email в приглашении
```

```
            invite.email = user.email
            invite.save()

        # Добавляем пользователя в проект
        CollaborationService.add_project_member(
            invite.project.id,
            user.id,
            invite.role,
            invite.invited_by_id
        )

        invite.status = ProjectInvite.Status.ACCEPTED
        invite.save()

        return invite

    # ============================================
    # Назначение задач
    # ============================================

    @staticmethod
    def assign_task(task_id, user_id, assigned_by_id):
        """Назначение задачи пользователю"""
        assignment, created = TaskAssignment.objects.get_or_create(
            task_id=task_id,
            user_id=user_id,
            defaults={'assigned_by_id': assigned_by_id}
        )

        # Логирование
        ActivityLog.objects.create(
            task_id=task_id,
```

```
            user_id=assigned_by_id,
            action_type=ActivityLog.ActionType.ASSIGNED,
            new_value=User.objects.get(id=user_id).username
        )

        return assignment

    @staticmethod
    def unassign_task(task_id, user_id, unassigned_by_id):
        """Снятие назначения задачи"""
        TaskAssignment.objects.filter(task_id=task_id, user_id=user_id).delete()

        ActivityLog.objects.create(
            task_id=task_id,
            user_id=unassigned_by_id,
            action_type=ActivityLog.ActionType.UNASSIGNED,
            old_value=User.objects.get(id=user_id).username
        )

    @staticmethod
    def get_task_assignees(task_id):
        """Получение назначенных пользователей на задачу"""
        return TaskAssignment.objects.filter(task_id=task_id).select_related('user')

    # ==========================================
    # Комментарии
    # ==========================================

    @staticmethod
    def add_comment(task_id, author_id, content, parent_id=None):
        """Добавление комментария"""
        comment = Comment.objects.create(
```

```
            task_id=task_id,
            author_id=author_id,
            content=content,
            parent_id=parent_id
        )

        ActivityLog.objects.create(
            task_id=task_id,
            user_id=author_id,
            action_type=ActivityLog.ActionType.COMMENT_ADDED,
            new_value=f"Комментарий: {content[:100]}..."
        )

        return comment

    @staticmethod
    def get_task_comments(task_id):
        """Получение всех комментариев задачи"""
        return Comment.objects.filter(task_id=task_id).select_related('author')

    @staticmethod
    def delete_comment(comment_id, user_id):
        """Удаление комментария (только автор или владелец)"""
        comment = Comment.objects.get(id=comment_id)
        if comment.author_id == user_id:
            comment.delete()
            return True
        return False

    @staticmethod
    def update_comment(comment_id, user_id, new_content):
        """Обновление комментария"""
```

```
        comment = Comment.objects.get(id=comment_id)
        if comment.author_id == user_id:
            comment.content = new_content
            comment.save()
            return True
        return False

    # ============================================
    # История изменений
    # ============================================

    @staticmethod
    @transaction.atomic
    def log_task_activity(task_id, user_id, action_type, old_value=None, new_value=None):
        """Запись действия в лог"""
        ActivityLog.objects.create(
            task_id=task_id,
            user_id=user_id,
            action_type=action_type,
            old_value=old_value,
            new_value=new_value
        )

    @staticmethod
    def get_task_activity(task_id, limit=50):
        """Получение истории действий задачи"""
        return ActivityLog.objects.filter(task_id=task_id).select_related('user')[:limit]

    @staticmethod
    def get_user_activity(user_id, limit=50):
        """Получение истории действий пользователя"""
        return ActivityLog.objects.filter(user_id=user_id)[:limit]
```

Шаг 4. Создание API эндпоинтов

tasks/views/collaboration_views.py:

```
from rest_framework views import APIView
from rest_framework response import Response
from rest_framework permissions import IsAuthenticated
from drf_spectacular utils import extend_schema
from ..models import Project, Task, User
from ..services collaboration_service import CollaborationService
from ..serializers import (
    ProjectMemberSerializer, CommentSerializer,
    ActivityLogSerializer, TaskAssignmentSerializer
)


class ProjectMembersView(APIView):
    """Управление участниками проекта"""
    permission_classes = [IsAuthenticated]

    @extend_schema(summary="Получить участников проекта")
    def get(self, request, project_id):
        # Проверка прав
        if not CollaborationService check_user_permission(project_id, request user.id, 'viewer'):
            return Response({'error': 'No permission'}, status=403)

        members = CollaborationService get_project_members(project_id)
        return Response(ProjectMemberSerializer(members, many=True).data)

    @extend_schema(summary="Добавить участника в проект")
    def post(self, request, project_id):
        # Только владелец может добавлять участников
```

```
        if not CollaborationService.check_user_permission(project_id, request.user.id, 'owner'):
            return Response({'error': 'Only owner can add members'}, status=403)

        user_id = request.data.get('user_id')
        role = request.data.get('role', 'viewer')

        try:
            member = CollaborationService.add_project_member(
                project_id, user_id, role, request.user.id
            )
            return Response(ProjectMemberSerializer(member).data, status=201)
        except Exception as e:
            return Response({'error': str(e)}, status=400)

    @extend_schema(summary="Удалить участника из проекта")
    def delete(self, request, project_id, user_id):
        if not CollaborationService.check_user_permission(project_id, request.user.id, 'owner'):
            return Response({'error': 'Only owner can remove members'}, status=403)

        CollaborationService.remove_project_member(project_id, user_id)
        return Response(status=204)


class ProjectInviteView(APIView):
    """Приглашения в проект"""
    permission_classes = [IsAuthenticated]

    @extend_schema(summary="Создать приглашение")
    def post(self, request, project_id):
        if not CollaborationService.check_user_permission(project_id, request.user.id, 'owner'):
            return Response({'error': 'Only owner can invite'}, status=403)
```

```
        email = request.data.get('email')
        role = request.data.get('role', 'viewer')

        invite = CollaborationService.create_invite(project_id, email, role, request.user.id)
        return Response({'invite_link': f"/invite/{invite.token}"}, status=201)


class AcceptInviteView(APIView):
    """Принятие приглашения"""
    permission_classes = [IsAuthenticated]

    @extend_schema(summary="Принять приглашение")
    def post(self, request, token):
        result = CollaborationService.accept_invite(token, request.user)
        if result:
            return Response({'message': 'Invite accepted'})
        return Response({'error': 'Invalid or expired invite'}, status=400)


class TaskAssignmentsView(APIView):
    """Назначение задач"""
    permission_classes = [IsAuthenticated]

    @extend_schema(summary="Назначить задачу пользователю")
    def post(self, request, task_id):
        task = Task.objects.get(id=task_id)
        project_id = task.project_id

        if project_id and not CollaborationService.check_user_permission(project_id, request.user.id,
'editor'):
            return Response({'error': 'No permission'}, status=403)
```

```
        user_id = request.data.get('user_id')
        assignment = CollaborationService.assign_task(task_id, user_id, request.user.id)
        return Response(TaskAssignmentSerializer(assignment).data, status=201)

    @extend_schema(summary="Снять назначение задачи")
    def delete(self, request, task_id, user_id):
        task = Task.objects.get(id=task_id)
        project_id = task.project_id

        if project_id and not CollaborationService.check_user_permission(project_id, request.user.id,
'editor'):
            return Response({'error': 'No permission'}, status=403)

        CollaborationService.unassign_task(task_id, user_id, request.user.id)
        return Response(status=204)


class TaskCommentsView(APIView):
    """Комментарии к задачам"""
    permission_classes = [IsAuthenticated]

    @extend_schema(summary="Получить комментарии задачи")
    def get(self, request, task_id):
        comments = CollaborationService.get_task_comments(task_id)
        return Response(CommentSerializer(comments, many=True).data)

    @extend_schema(summary="Добавить комментарий")
    def post(self, request, task_id):
        content = request.data.get('content')
        parent_id = request.data.get('parent_id')

        if not content:
```

```
            return Response({'error': 'Content is required'}, status=400)

        comment = CollaborationService.add_comment(
            task_id, request.user.id, content, parent_id
        )
        return Response(CommentSerializer(comment).data, status=201)


class TaskActivityView(APIView):
    """История изменений задачи"""
    permission_classes = [IsAuthenticated]

    @extend_schema(summary="Получить историю задачи")
    def get(self, request, task_id):
        activities = CollaborationService.get_task_activity(task_id)
        return Response(ActivityLogSerializer(activities, many=True).data)
```

Шаг 5. Добавление маршрутов

tasks/urls.py (дополнение):

```
from .views.collaboration_views import (
    ProjectMembersView, ProjectInviteView, AcceptInviteView,
    TaskAssignmentsView, TaskCommentsView, TaskActivityView
)

urlpatterns += [
    # Участники проекта
    path('api/projects/<int:project_id>/members/', ProjectMembersView.as_view(), name='project-members'),
    path('api/projects/<int:project_id>/members/<int:user_id>/', ProjectMembersView.as_view(), name='project-member-delete'),

    # Приглашения
```

```
    path('api/projects/<int:project_id>/invite/', ProjectInviteView.as_view(), name='project-invite'),
    path('api/invite/<str:token>/accept/', AcceptInviteView.as_view(), name='accept-invite'),

    # Назначения
    path('api/tasks/<int:task_id>/assign/', TaskAssignmentsView.as_view(), name='task-assign'),
    path('api/tasks/<int:task_id>/assign/<int:user_id>/', TaskAssignmentsView.as_view(), name='task-unassign'),

    # Комментарии
    path('api/tasks/<int:task_id>/comments/', TaskCommentsView.as_view(), name='task-comments'),

    # История
    path('api/tasks/<int:task_id>/activity/', TaskActivityView.as_view(), name='task-activity'),
]
```

Шаг 6. Фронтенд: компонент комментариев

src/components/comments/CommentSection.jsx:

```
import React, { useState, useEffect } from 'react';
import {
  Box,
  Paper,
  Typography,
  TextField,
  Button,
  Avatar,
  Divider,
  IconButton,
  CircularProgress,
} from '@mui/material';
import {
  Send as SendIcon,
```

```
  Delete as DeleteIcon,
  Edit as EditIcon,
} from '@mui/icons-material';
import { formatDistanceToNow } from 'date-fns';
import { ru } from 'date-fns/locale';
import apiClient from '../../api/client';

const Comment = ({ comment, onDelete, onReply, currentUser }) => {
  const [showReplyForm, setShowReplyForm] = useState(false);
  const [replyContent, setReplyContent] = useState('');

  return (
    <Box sx={{ mb: 2, ml: comment.parent_id ? 4 : 0 }}>
      <Box display="flex" gap={2}>
        <Avatar sx={{ bgcolor: '#3498db' }}>
          {comment.author?.username?.[0]?.toUpperCase()}
        </Avatar>
        <Box flex={1}>
          <Box display="flex" alignItems="center" gap={1} mb={0.5}>
            <Typography variant="subtitle2">
              {comment.author?.username}
            </Typography>
            <Typography variant="caption" color="text.secondary">
              {formatDistanceToNow(new Date(comment.created_at), {
                addSuffix: true,
                locale: ru,
              })}
            </Typography>
          </Box>
          <Typography variant="body2" sx={{ whiteSpace: 'pre-wrap' }}>
            {comment.content}
          </Typography>
```

```
<Box display="flex" gap={1} mt={1}>
  <Button size="small" onClick={() => setShowReplyForm(!showReplyForm)}>
    Ответить
  </Button>
  {comment.author?.id === currentUser?.id && (
    <>
      <IconButton size="small" onClick={() => onDelete(comment.id)}>
        <DeleteIcon fontSize="small" />
      </IconButton>
    </>
  )}
</Box>

{showReplyForm && (
  <Box display="flex" gap={1} mt={1}>
    <TextField
      size="small"
      placeholder="Написать ответ..."
      value={replyContent}
      onChange={(e) => setReplyContent(e.target.value)}
      fullWidth
    />
    <Button
      variant="contained"
      size="small"
      onClick={() => {
        onReply(comment.id, replyContent);
        setReplyContent('');
        setShowReplyForm(false);
      }}
    >
      Ответить
```

```
                    </Button>
                  </Box>
                ))}
              </Box>
            </Box>
          </Box>
        );
};

const CommentSection = ({ taskId }) => {
  const [comments, setComments] = useState([]);
  const [newComment, setNewComment] = useState('');
  const [loading, setLoading] = useState(true);
  const [submitting, setSubmitting] = useState(false);

  useEffect(() => {
    fetchComments();
  }, [taskId]);

  const fetchComments = async () => {
    try {
      const response = await apiClient.get(`/tasks/${taskId}/comments/`);
      setComments(response.data);
    } catch (error) {
      console.error('Error fetching comments:', error);
    } finally {
      setLoading(false);
    }
  };

  const handleAddComment = async () => {
    if (!newComment.trim()) return;
```

```
    setSubmitting(true);
    try {
      await apiClient.post(`/tasks/${taskId}/comments/`, {
        content: newComment
      });
      setNewComment('');
      await fetchComments();
    } catch (error) {
      console.error('Error adding comment:', error);
    } finally {
      setSubmitting(false);
    }
  };

  const handleDeleteComment = async (commentId) => {
    if (!window.confirm('Удалить комментарий?')) return;

    try {
      await apiClient.delete(`/comments/${commentId}/`);
      await fetchComments();
    } catch (error) {
      console.error('Error deleting comment:', error);
    }
  };

  const handleReply = async (parentId, content) => {
    try {
      await apiClient.post(`/tasks/${taskId}/comments/`, {
        content,
        parent_id: parentId,
      });
```

```
      await fetchComments();
    } catch (error) {
      console.error('Error adding reply:', error);
    }
};

// Организация комментариев в дерево
const commentMap = {};
const rootComments = [];

comments.forEach(comment => {
  commentMap[comment.id] = { ...comment, replies: [] };
});

comments.forEach(comment => {
  if (comment.parent_id) {
    if (commentMap[comment.parent_id]) {
      commentMap[comment.parent_id].replies.push(commentMap[comment.id]);
    }
  } else {
    rootComments.push(commentMap[comment.id]);
  }
});

if (loading) {
  return (
    <Box display="flex" justifyContent="center" py={4}>
      <CircularProgress />
    </Box>
  );
}
```

```
return (
  <Paper sx={{ p: 2, mt: 2 }}>
    <Typography variant="h6" gutterBottom>
      Комментарии ({comments.length})
    </Typography>

    <Box display="flex" gap={2} mb={3}>
      <TextField
        fullWidth
        multiline
        rows={3}
        placeholder="Написать комментарий..."
        value={newComment}
        onChange={(e) => setNewComment(e.target.value)}
        disabled={submitting}
      />
      <Button
        variant="contained"
        onClick={handleAddComment}
        disabled={submitting || !newComment.trim()}
        sx={{ alignSelf: 'flex-end' }}
      >
        <SendIcon />
      </Button>
    </Box>

    <Divider />

    <Box sx={{ mt: 2 }}>
      {rootComments.map(comment => (
        <Comment
          key={comment.id}
```

```
                comment={comment}
                onDelete={handleDeleteComment}
                onReply={handleReply}
              />
            ))}
            {rootComments.length === 0 && (
              <Typography color="text.secondary" textAlign="center" py={4}>
                Нет комментариев. Будьте первым!
              </Typography>
            )}
          </Box>
        </Paper>
    );
};

export default CommentSection;
```

Шаг 7. Фронтенд: компонент истории изменений

src/components/history/ActivityLog.jsx:

```
import React, { useState, useEffect } from 'react';
import {
  Box,
  Paper,
  Typography,
  List,
  ListItem,
  ListItemText,
  ListItemAvatar,
  Avatar,
  Chip,
  CircularProgress,
```

```
} from '@mui/material';
import {
  Create as CreateIcon,
  Edit as EditIcon,
  SwapHoriz as StatusIcon,
  Flag as PriorityIcon,
  Schedule as DeadlineIcon,
  Person as AssignIcon,
  Comment as CommentIcon,
} from '@mui/icons-material';
import { formatDistanceToNow } from 'date-fns';
import { ru } from 'date-fns/locale';
import apiClient from '../../api/client';

const actionIcons = {
  created: <CreateIcon />,
  updated: <EditIcon />,
  status_changed: <StatusIcon />,
  priority_changed: <PriorityIcon />,
  deadline_changed: <DeadlineIcon />,
  assigned: <AssignIcon />,
  comment_added: <CommentIcon />,
};

const actionLabels = {
  created: 'создал(а) задачу',
  updated: 'обновил(а) задачу',
  status_changed: 'изменил(а) статус',
  priority_changed: 'изменил(а) приоритет',
  deadline_changed: 'изменил(а) дедлайн',
  assigned: 'назначил(а) задачу',
  comment_added: 'добавил(а) комментарий',
```

```
};

const statusMap = {
    New: 'Новая',
    InProgress: 'В работе',
    Completed: 'Выполнена',
};

const priorityMap = {
    Low: 'Низкий',
    Medium: 'Средний',
    High: 'Высокий',
};

const ActivityLog = ({ taskId }) => {
    const [activities, setActivities] = useState([]);
    const [loading, setLoading] = useState(true);

    useEffect(() => {
        fetchActivities();
    }, [taskId]);

    const fetchActivities = async () => {
        try {
            const response = await apiClient.get(`/tasks/${taskId}/activity/`);
            setActivities(response.data);
        } catch (error) {
            console.error('Error fetching activities:', error);
        } finally {
            setLoading(false);
        }
    };
```

```
  const formatChange = (activity) => {
    if (activity.action_type === 'status_changed') {
      return `${statusMap[activity.old_value] || activity.old_value} →
${statusMap[activity.new_value] || activity.new_value}`;
    }
    if (activity.action_type === 'priority_changed') {
      return `${priorityMap[activity.old_value] || activity.old_value} →
${priorityMap[activity.new_value] || activity.new_value}`;
    }
    if (activity.action_type === 'deadline_changed') {
      const formatDate = (date) => date ? new Date(date).toLocaleDateString('ru-RU') : 'не указано';
      return `${formatDate(activity.old_value)} → ${formatDate(activity.new_value)}`;
    }
    if (activity.action_type === 'assigned') {
      return `назначил(а) задачу пользователю ${activity.new_value}`;
    }
    if (activity.action_type === 'comment_added') {
      return activity.new_value;
    }
    return '';
  };

  if (loading) {
    return (
      <Box display="flex" justifyContent="center" py={4}>
        <CircularProgress />
      </Box>
    );
  }

  return (
    <Paper sx={{ p: 2, mt: 2 }}>
```

```
<Typography variant="h6" gutterBottom>
  История изменений
</Typography>

<List>
  {activities.map((activity) => (
    <ListItem key={activity.id} divider>
      <ListItemAvatar>
        <Avatar sx={{ bgcolor: '#3498db' }}>
          {actionIcons[activity.action_type] || <EditIcon />}
        </Avatar>
      </ListItemAvatar>
      <ListItemText
        primary={
          <Box>
            <Typography component="span" fontWeight="bold">
              {activity.user?.username}
            </Typography>
            <Typography component="span" ml={1}>
              {actionLabels[activity.action_type] || activity.action_type}
            </Typography>
          </Box>
        }
        secondary={
          <Box>
            {activity.old_value && (
              <Typography variant="caption" color="text.secondary" component="div">
                {formatChange(activity)}
              </Typography>
            )}
            <Typography variant="caption" color="text.secondary">
              {formatDistanceToNow(new Date(activity.created_at), {
```

```
                        addSuffix: true,
                        locale: ru,
                      })}
                    </Typography>
                  </Box>
                }
              />
            </ListItem>
          ))}
          {activities.length === 0 && (
            <Typography color="text.secondary" textAlign="center" py={4}>
              История изменений пуста
            </Typography>
          )}
        </List>
      </Paper>
    );
};

export default ActivityLog;
```

Контрольные вопросы

1. Какие роли в проекте существуют и какие права они дают?
2. Как работает система приглашений в проект?
3. Как реализовано назначение задач пользователям?
4. Какие типы действий логируются в истории изменений?
5. Как организована система комментариев (включая вложенные ответы)?
6. Как проверяются права пользователя при выполнении действий?

Критерии оценки

Критерий	Баллы
Реализована система приглашений в проекты	2
Реализованы роли в проекте (владелец, редактор, читатель)	2
Реализованы комментарии к задачам	2
Реализована система логов действий	2
Реализованы уведомления о назначении	1
Реализовано API и фронтенд	1
Итого	10

Результат выполнения

После выполнения лабораторной работы №13 студенты должны получить:

1. Систему приглашений пользователей в проекты
2. Ролевую модель доступа к проектам
3. Комментарии к задачам с ответами
4. Историю изменений задач
5. Уведомления о назначении задач и приглашениях

Лабораторная работа №14

Экспорт, импорт и отчеты в PDF

Цель работы

Реализация расширенных функций экспорта данных в различных форматах (CSV, JSON, PDF), импорта задач из внешних файлов и генерации отчетов.

Задачи

№	Задача
1	Реализовать экспорт задач в CSV и JSON форматы
2	Реализовать генерацию PDF-отчетов с использованием библиотеки
3	Реализовать импорт задач из CSV/JSON файлов
4	Добавить возможность выбора полей для экспорта
5	Реализовать генерацию и отправку еженедельного отчета по email
6	Реализовать предпросмотр отчета перед экспортом

Теоретическая часть

Форматы экспорта

Формат	**Расширение**	**MIME-тип**	**Применение**
CSV	.csv	text/csv	Импорт в Excel, Google Sheets
JSON	.json	application/json	API, обмен данными
PDF	.pdf	application/pdf	Печатные отчеты, документация

Библиотеки для работы с PDF в Python

Библиотека	Описание
reportlab	Мощная библиотека для генерации PDF
WeasyPrint	Конвертация HTML в PDF
pdfkit	Обертка над wkhtmltopdf

Библиотеки для работы с PDF в JavaScript (фронтенд)

Библиотека	Описание
jsPDF	Генерация PDF на клиенте
react-pdf	React-компоненты для PDF
pdf-lib	Работа с существующими PDF

Ход выполнения работы

Шаг 1. Установка необходимых пакетов

Бэкенд (Django):

```
pip install reportlab
pip install openpyxl  # для Excel
pip install django-celery-email  # для асинхронной отправки email
```

Фронтенд (React):

```
bash
npm install jspdf jspdf-autotable
npm install file-saver
npm install papaparse  # для парсинга CSV
```

Шаг 2. Создание сервиса экспорта

tasks/services/export_service.py:

```
import csv
import json
import io
from datetime import datetime
from django.http import HttpResponse
from django.utils import timezone
from reportlab.lib import colors
from reportlab.lib.pagesizes import A4, landscape
from reportlab.lib.styles import getSampleStyleSheet, ParagraphStyle
from reportlab.lib.units import inch, cm
from reportlab.platypus import (
    SimpleDocTemplate, Paragraph, Spacer, Table, TableStyle,
    PageBreak, Image, KeepTogether
)
from reportlab.lib.enums import TA_CENTER, TA_RIGHT
from reportlab.pdfbase import pdfmetrics
from reportlab.pdfbase.ttfonts import TTFont
from .models import Task, Project, Tag


class ExportService:
    """Сервис для экспорта данных в различные форматы"""

    @staticmethod
    def export_to_csv(user_id, filters=None, fields=None):
        """Экспорт задач в CSV"""
        response = HttpResponse(content_type='text/csv; charset=utf-8')
        response['Content-Disposition'] = f'attachment; filename="tasks_export_{datetime.now().strftime("%Y%m%d_%H%M%S")}.csv"'

        tasks = ExportService._get_filtered_tasks(user_id, filters)
```

```
    # Поля для экспорта по умолчанию
    if not fields:
        fields = ['id', 'title', 'description', 'status', 'priority',
                  'deadline', 'created_at', 'project_name', 'tags']

    # Заголовки на русском
    headers_map = {
        'id': 'ID',
        'title': 'Заголовок',
        'description': 'Описание',
        'status': 'Статус',
        'priority': 'Приоритет',
        'deadline': 'Дедлайн',
        'created_at': 'Дата создания',
        'completed_at': 'Дата выполнения',
        'project_name': 'Проект',
        'tags': 'Теги',
    }

    writer = csv.writer(response)
    writer.writerow([headers_map.get(f, f) for f in fields])

    status_map = {'New': 'Новая', 'InProgress': 'В работе', 'Completed': 'Выполнена'}
    priority_map = {'Low': 'Низкий', 'Medium': 'Средний', 'High': 'Высокий'}

    for task in tasks:
        row = []
        for field in fields:
            if field == 'status':
                value = status_map.get(task.status, task.status)
            elif field == 'priority':
                value = priority_map.get(task.priority, task.priority)
```

```
                elif field == 'deadline':
                    value = task.deadline.strftime('%Y-%m-%d %H:%M') if task.deadline else ''
                elif field == 'created_at':
                    value = task.created_at.strftime('%Y-%m-%d %H:%M')
                elif field == 'completed_at':
                    value = task.completed_at.strftime('%Y-%m-%d %H:%M') if task.completed_at
else ''
                elif field == 'project_name':
                    value = task.project.name if task.project else ''
                elif field == 'tags':
                    value = ', '.join([tt.tag.name for tt in task.task_tags.all()])
                else:
                    value = getattr(task, field, '')
                row.append(str(value) if value else '')
            writer.writerow(row)

        return response

    @staticmethod
    def export_to_json(user_id, filters=None, include_stats=True):
        """Экспорт задач в JSON"""
        tasks = ExportService._get_filtered_tasks(user_id, filters)

        status_map = {'New': 'Новая', 'InProgress': 'В работе', 'Completed': 'Выполнена'}
        priority_map = {'Low': 'Низкий', 'Medium': 'Средний', 'High': 'Высокий'}

        data = {
            'export_date': datetime.now().isoformat(),
            'total_tasks': tasks.count(),
            'tasks': []
        }
```

```
        for task in tasks
            task_data = {
                'id': task.id,
                'title': task.title,
                'description': task.description,
                'status': task.status,
                'status_display': status_map.get(task.status, task.status),
                'priority': task.priority,
                'priority_display': priority_map.get(task.priority, task.priority),
                'deadline': task.deadline.isoformat() if task.deadline else None,
                'created_at': task.created_at.isoformat(),
            'completed_at': task.completed_at.isoformat() if task.completed_at else None,
                'project': {
                    'id': task.project.id,
                    'name': task.project.name,
                    'color': task.project.color
                } if task.project else None,
                'tags': [{'id': tt.tag.id, 'name': tt.tag.name} for tt in task.task_tags.all()],
            }
            data['tasks'].append(task_data)

        if include_stats:
            data['statistics'] = ExportService._get_statistics(user_id, tasks)

        response = HttpResponse(
            json.dumps(data, indent=2, ensure_ascii=False),
            content_type='application/json'
        )
        response['Content-Disposition'] = f'attachment; filename="tasks_export_{datetime.now().strftime("%Y%m%d_%H%M%S")}.json"'
        return response
```

```
    @staticmethod
    def export_to_pdf(user_id, filters=None, include_charts=True):
        """Экспорт задач в PDF с отчетом"""
        tasks = ExportService._get_filtered_tasks(user_id, filters)
        statistics = ExportService._get_statistics(user_id, tasks)

        response = HttpResponse(content_type='application/pdf')
        response['Content-Disposition'] = f'attachment; filename="tasks_report_{datetime.now().strftime("%Y%m%d_%H%M%S")}.pdf"'

        # Создание PDF документа
        buffer = io.BytesIO()
        doc = SimpleDocTemplate(
            buffer,
            pagesize=A4,
            rightMargin=72,
            leftMargin=72,
            topMargin=72,
            bottomMargin=72,
        )

        # Стили
        styles = getSampleStyleSheet()
        title_style = ParagraphStyle(
            'CustomTitle',
            parent=styles['Heading1'],
            fontSize=24,
            textColor=colors.HexColor('#2c3e50'),
            spaceAfter=30,
            alignment=TA_CENTER
        )
```

```
heading_style = ParagraphStyle(
    'CustomHeading',
    parent=styles['Heading2'],
    fontSize=16,
    textColor=colors HexColor('#34495e'),
    spaceAfter=15,
    spaceBefore=20
)

normal_style = ParagraphStyle(
    'CustomNormal',
    parent=styles['Normal'],
    fontSize=10,
    leading=14
)

# Сборка содержимого
story = []

# Заголовок
story append(Paragraph("Отчет по задачам", title_style))
story append(Paragraph(
    f"Дата генерации: {timezone now().strftime('%d.%m.%Y %H:%M:%S')}",
    normal_style
))
story append(Spacer(1, 20))

# Статистика
story append(Paragraph("1. Общая статистика", heading_style))

stats_data = [
    ['Показатель', 'Значение'],
```

```
            ['Всего задач', str(statistics['total'])],
            ['Выполнено', f"{statistics['completed']} ({statistics['completion_rate']}%)"],
            ['В работе', str(statistics['in_progress'])],
            ['Новые', str(statistics['new'])],
            ['Просрочено', str(statistics['overdue'])],
        ]

        stats_table = Table(stats_data, colWidths=[200, 100])
        stats_table.setStyle(TableStyle([
            ('BACKGROUND', (0, 0), (-1, 0), colors.HexColor('#34495e')),
            ('TEXTCOLOR', (0, 0), (-1, 0), colors.white),
            ('ALIGN', (0, 0), (-1, -1), 'LEFT'),
            ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
            ('FONTSIZE', (0, 0), (-1, 0), 12),
            ('BOTTOMPADDING', (0, 0), (-1, 0), 12),
            ('BACKGROUND', (0, 1), (-1, -1), colors.HexColor('#ecf0f1')),
            ('GRID', (0, 0), (-1, -1), 1, colors.HexColor('#bdc3c7')),
            ('PADDING', (0, 0), (-1, -1), 8),
        ]))
        story.append(stats_table)
        story.append(Spacer(1, 20))

        # Распределение по приоритетам
        story.append(Paragraph("2. Распределение по приоритетам", heading_style))

        priority_data = [
            ['Приоритет', 'Количество', 'Процент'],
            ['Высокий', str(statistics['priority_high']),
f"{statistics['priority_high_percent']}%"],
            ['Средний', str(statistics['priority_medium']),
f"{statistics['priority_medium_percent']}%"],
            ['Низкий', str(statistics['priority_low']), f"{statistics['priority_low_percent']}%"],
```

```
]

priority_table = Table(priority_data, colWidths=[100, 80, 80])
priority_table.setStyle(TableStyle([
    ('BACKGROUND', (0, 0), (-1, 0), colors.HexColor('#34495e')),
    ('TEXTCOLOR', (0, 0), (-1, 0), colors.white),
    ('ALIGN', (1, 0), (2, -1), 'CENTER'),
    ('GRID', (0, 0), (-1, -1), 1, colors.HexColor('#bdc3c7')),
    ('PADDING', (0, 0), (-1, -1), 8),
]))
story.append(priority_table)
story.append(Spacer(1, 20))

# Список задач
story.append(Paragraph("3. Список задач", heading_style))

if tasks.exists():
    task_data = [['№', 'Задача', 'Статус', 'Приоритет', 'Дедлайн']]
    for idx, task in enumerate(tasks[:50], 1):
        task_data.append([
            str(idx),
            task.title[:50] + '...' if len(task.title) > 50 else task.title,
            status_map.get(task.status, task.status),
            priority_map.get(task.priority, task.priority),
            task.deadline.strftime('%d.%m.%Y') if task.deadline else '-'
        ])

    task_table = Table(task_data, colWidths=[40, 250, 70, 70, 80])
    task_table.setStyle(TableStyle([
        ('BACKGROUND', (0, 0), (-1, 0), colors.HexColor('#34495e')),
        ('TEXTCOLOR', (0, 0), (-1, 0), colors.white),
        ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
```

```
            ('FONTSIZE', (0, 0), (-1, 0), 9),
            ('GRID', (0, 0), (-1, -1), 0.5, colors.HexColor('#bdc3c7')),
            ('FONTSIZE', (0, 1), (-1, -1), 8),
            ('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),
        ]))
        story.append(task_table)

        if tasks.count() > 50:
            story.append(Paragraph(
                f"<i>... и еще {tasks.count() - 50} задач</i>",
                normal_style
            ))
    else:
        story.append(Paragraph("Нет задач для отображения", normal_style))

    # Сборка PDF
    doc.build(story)

    pdf = buffer.getvalue()
    buffer.close()
    response.write(pdf)
    return response

@staticmethod
def _get_filtered_tasks(user_id, filters=None):
    """Получение отфильтрованных задач"""
    queryset = Task.objects.filter(user_id=user_id).select_related('project')

    if filters:
        if filters.get('status'):
            queryset = queryset.filter(status=filters['status'])
        if filters.get('priority'):
```

```
            queryset = queryset.filter(priority=filters['priority'])
        if filters.get('project_id'):
            queryset = queryset.filter(project_id=filters['project_id'])
        if filters.get('date_from'):
            queryset = queryset.filter(created_at__gte=filters['date_from'])
        if filters.get('date_to'):
            queryset = queryset.filter(created_at__lte=filters['date_to'])
        if filters.get('search'):
            queryset = queryset.filter(title__icontains=filters['search'])

    return queryset

@staticmethod
def _get_statistics(user_id, queryset=None):
    """Получение статистики для отчета"""
    if queryset is None:
        queryset = Task.objects.filter(user_id=user_id)

    total = queryset.count()
    completed = queryset.filter(status='Completed').count()
    in_progress = queryset.filter(status='InProgress').count()
    new = queryset.filter(status='New').count()
    overdue = queryset.filter(
        deadline__lt=timezone.now(),
        status__in=['New', 'InProgress']
    ).count()

    priority_high = queryset.filter(priority='High').count()
    priority_medium = queryset.filter(priority='Medium').count()
    priority_low = queryset.filter(priority='Low').count()

    return {
```

```
            'total': total,
            'completed': completed,
            'in_progress': in_progress,
            'new': new,
            'overdue': overdue,
            'completion_rate': round(completed / total * 100, 1) if total > 0 else 0,
            'priority_high': priority_high,
            'priority_medium': priority_medium,
            'priority_low': priority_low,
            'priority_high_percent': round(priority_high / total * 100, 1) if total > 0 else 0,
            'priority_medium_percent': round(priority_medium / total * 100, 1) if total > 0
else 0,
            'priority_low_percent': round(priority_low / total * 100, 1) if total > 0 else 0,
        }


class ImportService:
    """Сервис для импорта данных"""

    @staticmethod
    def import_from_csv(user_id, csv_file):
        """Импорт задач из CSV"""
        import csv
        from io import TextIOWrapper

        decoded_file = TextIOWrapper(csv_file, encoding='utf-8')
        reader = csv.DictReader(decoded_file)

        results = {
            'created': 0,
            'updated': 0,
            'errors': [],
```

```
        'skipped': 0
    }

    status_map = {'Новая': 'New', 'В работе': 'InProgress', 'Выполнена': 'Completed'}
    priority_map = {'Низкий': 'Low', 'Средний': 'Medium', 'Высокий': 'High'}

    for row_num, row in enumerate(reader, 2):
        try:
            # Поиск существующей задачи по ID (если указан)
            task_id = row.get('ID')
            if task_id:
                try:
                    task = Task.objects.get(id=task_id, user_id=user_id)
                    # Обновление существующей задачи
                    if row.get('Заголовок'):
                        task.title = row['Заголовок']
                    if row.get('Описание'):
                        task.description = row['Описание']
                    if row.get('Статус'):
                        task.status = status_map.get(row['Статус'], row['Статус'])
                    if row.get('Приоритет'):
                        task.priority = priority_map.get(row['Приоритет'], row['Приоритет'])
                    task.save()
                    results['updated'] += 1
                    continue
                except Task.DoesNotExist:
                    pass

            # Создание новой задачи
            task_data = {
                'title': row.get('Заголовок', ''),
                'description': row.get('Описание', ''),
```

```
                'status': status_map.get(row.get('Статус', ''), 'New'),
                'priority': priority_map.get(row.get('Приоритет', ''), 'Medium'),
                'user_id': user_id,
            }

            if not task_data['title']:
                results['skipped'] += 1
                results['errors'].append(f'Строка {row_num}: отсутствует заголовок')
                continue

            task = Task.objects.create(**task_data)
            results['created'] += 1

        except Exception as e:
            results['errors'].append(f'Строка {row_num}: {str(e)}')

    return results

@staticmethod
def import_from_json(user_id, json_file):
    """Импорт задач из JSON"""
    import json

    data = json.load(json_file)
    results = {
        'created': 0,
        'updated': 0,
        'errors': [],
        'skipped': 0
    }

    tasks_data = data if isinstance(data, list) else data.get('tasks', [])
```

```
        for task_data in tasks_data:
            try:
                task_id = task_data.get('id')
                if task_id:
                    try:
                        task = Task.objects.get(id=task_id, user_id=user_id)
                        task.title = task_data.get('title', task.title)
                        task.description = task_data.get('description', task.description)
                        task.status = task_data.get('status', task.status)
                        task.priority = task_data.get('priority', task.priority)
                        task.save()
                        results['updated'] += 1
                        continue
                    except Task.DoesNotExist:
                        pass

                task = Task.objects.create(
                    title=task_data.get('title', ''),
                    description=task_data.get('description', ''),
                    status=task_data.get('status', 'New'),
                    priority=task_data.get('priority', 'Medium'),
                    user_id=user_id,
                )
                results['created'] += 1

            except Exception as e:
                results['errors'].append(str(e))

        return results
```

Шаг 3. Создание API эндпоинтов для экспорта/импорта

tasks/views/export_views.py:

```
from rest_framework views import APIView
from rest_framework response import Response
from rest_framework permissions import IsAuthenticated
from rest_framework parsers import MultiPartParser, FormParser
from drf_spectacular utils import extend_schema, OpenApiParameter
from ..services export_service import ExportService, ImportService

class ExportTasksView(APIView):
    """API для экспорта задач"""
    permission_classes = [IsAuthenticated]

    @extend_schema(
        summary="Экспорт задач",
        description="Экспорт задач в указанном формате (csv, json, pdf)",
        parameters=[
            OpenApiParameter(name='format', description='Формат экспорта (csv, json, pdf)', required=True),
            OpenApiParameter(name='status', description='Фильтр по статусу'),
            OpenApiParameter(name='priority', description='Фильтр по приоритету'),
            OpenApiParameter(name='project_id', description='Фильтр по проекту'),
            OpenApiParameter(name='date_from', description='Дата начала'),
            OpenApiParameter(name='date_to', description='Дата окончания'),
        ]
    )
    def get(self, request):
        format_type = request query_params get('format', 'csv')

        filters = {
            'status': request query_params get('status'),
            'priority': request query_params get('priority'),
```

```
            'project_id': request.query_params.get('project_id'),
            'date_from': request.query_params.get('date_from'),
            'date_to': request.query_params.get('date_to'),
            'search': request.query_params.get('search'),
        }
        filters = {k: v for k, v in filters.items() if v}

        if format_type == 'csv':
            return ExportService.export_to_csv(request.user.id, filters)
        elif format_type == 'json':
            return ExportService.export_to_json(request.user.id, filters)
        elif format_type == 'pdf':
            return ExportService.export_to_pdf(request.user.id, filters)

        return Response({'error': 'Unsupported format'}, status=400)


class ImportTasksView(APIView):
    """API для импорта задач"""
    permission_classes = [IsAuthenticated]
    parser_classes = [MultiPartParser, FormParser]

    @extend_schema(
        summary="Импорт задач",
        description="Импорт задач из CSV или JSON файла",
        request={'multipart/form-data': {'file': 'file'}}
    )
    def post(self, request):
        file = request.FILES.get('file')
        if not file:
            return Response({'error': 'No file provided'}, status=400)
```

```
            file_name = file.name.lower()

            try:
                if file_name.endswith('.csv'):
                    result = ImportService.import_from_csv(request.user.id, file)
                elif file_name.endswith('.json'):
                    result = ImportService.import_from_json(request.user.id, file)
                else:
                    return Response({'error': 'Unsupported file format. Use CSV or JSON'}, status=400)

                return Response({
                    'message': f'Импорт завершен. Создано: {result["created"]}, Обновлено: {result["updated"]}, Пропущено: {result["skipped"]}',
                    'details': result
                })
            except Exception as e:
                return Response({'error': str(e)}, status=500)


class WeeklyReportView(APIView):
    """API для генерации и отправки еженедельного отчета"""
    permission_classes = [IsAuthenticated]

    @extend_schema(summary="Отправить еженедельный отчет на email")
    def post(self, request):
        from .tasks import send_weekly_report_email

        # Асинхронная отправка email
        send_weekly_report_email.delay(request.user.id, request.user.email)

        return Response({'message': 'Отчет отправлен на ваш email'})
```

Шаг 4. Создание Celery задачи для отправки отчета

tasks/tasks.py (дополнение):

```
from celery import shared_task
from django.core.mail import EmailMessage
from django.template.loader import render_to_string
from django.utils import timezone
from datetime import timedelta
from io import BytesIO
from reportlab.lib.pagesizes import A4
from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer, Table, TableStyle
from reportlab.lib.styles import getSampleStyleSheet, ParagraphStyle
from reportlab.lib import colors
from .services.export_service import ExportService

@shared_task
def send_weekly_report_email(user_id, email):
    """Отправка еженедельного отчета на email"""
    # Генерация PDF отчета за последнюю неделю
    end_date = timezone.now()
    start_date = end_date - timedelta(days=7)

    filters = {
        'date_from': start_date.isoformat(),
        'date_to': end_date.isoformat(),
    }

    # Генерация PDF
    pdf_buffer = BytesIO()
    doc = SimpleDocTemplate(pdf_buffer, pagesize=A4)
```

```
            # Стили
            styles = getSampleStyleSheet()
            title_style = ParagraphStyle('Title', parent=styles['Heading1'], fontSize=20,
textColor=colors.HexColor('#2c3e50'))

            story = []
            story.append(Paragraph(f"Еженедельный отчет", title_style))
            story.append(Paragraph(f"Период: {start_date.strftime('%d.%m.%Y')} -
{end_date.strftime('%d.%m.%Y')}", styles['Normal']))
            story.append(Spacer(1, 20))

            # Статистика за неделю
            tasks = ExportService._get_filtered_tasks(user_id, filters)
            stats = ExportService._get_statistics(user_id, tasks)

            stats_data = [
                ['Создано задач', str(tasks.count())],
                ['Выполнено задач', str(stats['completed'])],
                ['Текущая продуктивность', f"{stats['completion_rate']}%"],
            ]

            stats_table = Table(stats_data, colWidths=[150, 100])
            stats_table.setStyle(TableStyle([
                ('GRID', (0, 0), (-1, -1), 1, colors.grey),
                ('BACKGROUND', (0, 0), (-1, 0), colors.HexColor('#34495e')),
                ('TEXTCOLOR', (0, 0), (-1, 0), colors.white),
                ('PADDING', (0, 0), (-1, -1), 8),
            ]))
            story.append(stats_table)

            doc.build(story)
```

```
            # Отправка email
            pdf_buffer.seek(0)
            email_message = EmailMessage(
                subject=f'Smart Diary: Ваш еженедельный отчет за {start_date.strftime("%d.%m")} - {end_date.strftime("%d.%m.%Y")}',
                body='Во вложении ваш еженедельный отчет по задачам.',
                from_email='noreply@smartdiary.com',
                to=[email],
            )
            email_message.attach(f'weekly_report_{start_date.strftime("%Y%m%d")}.pdf', pdf_buffer.read(), 'application/pdf')
            email_message.send()

            return f'Report sent to {email}'
```

Шаг 5. Создание фронтенд компонента экспорта

src/components/export/ExportPanel.jsx:

```
javascript
import React, { useState } from 'react';
import {
  Box,
  Paper,
  Typography,
  Button,
  FormControl,
  InputLabel,
  Select,
  MenuItem,
  FormGroup,
  FormControlLabel,
  Checkbox,
  TextField,
```

```
    Grid,
    Alert,
    Chip,
    CircularProgress,
} from '@mui/material';
import {
    Download as DownloadIcon,
    Upload as UploadIcon,
    Email as EmailIcon,
    PictureAsPdf as PdfIcon,
    TableChart as CsvIcon,
    Code as JsonIcon,
} from '@mui/icons-material';
import { useAuth } from '../../context/AuthContext';
import apiClient from '../../api/client';

const EXPORT_FIELDS = [
    { id: 'title', label: 'Заголовок' },
    { id: 'description', label: 'Описание' },
    { id: 'status', label: 'Статус' },
    { id: 'priority', label: 'Приоритет' },
    { id: 'deadline', label: 'Дедлайн' },
    { id: 'created_at', label: 'Дата создания' },
    { id: 'project_name', label: 'Проект' },
    { id: 'tags', label: 'Теги' },
];

const ExportPanel = () => {
    const { user } = useAuth();
    const [exportFormat, setExportFormat] = useState('csv');
    const [selectedFields, setSelectedFields] = useState(EXPORT_FIELDS.map(f => f.id));
    const [exportFilters, setExportFilters] = useState({
```

```
    status: '',
    priority: '',
    project_id: '',
    date_from: '',
    date_to: '',
});
const [loading, setLoading] = useState(false);
const [importLoading, setImportLoading] = useState(false);
const [reportLoading, setReportLoading] = useState(false);
const [importResult, setImportResult] = useState(null);

// Экспорт
const handleExport = async () => {
    setLoading(true);
    try {
        const params = new URLSearchParams({
            format: exportFormat,
            ...exportFilters,
        });

        // Добавляем выбранные поля (только для CSV)
        if (exportFormat === 'csv') {
            params.append('fields', selectedFields.join(','));
        }

        const response = await apiClient.get(`/export/tasks/?${params.toString()}`, {
            responseType: 'blob',
        });

        // Создание ссылки для скачивания
        const url = window.URL.createObjectURL(new Blob([response.data]));
        const link = document.createElement('a');
```

```
        link.href = url;
        const contentDisposition = response.headers['content-disposition'];
        let filename = `export.${exportFormat}`;
        if (contentDisposition) {
            const match = contentDisposition.match(/filename="(.+)"/);
            if (match) filename = match[1];
        }
        link.setAttribute('download', filename);
        document.body.appendChild(link);
        link.click();
        link.remove();
        window.URL.revokeObjectURL(url);
    } catch (error) {
        console.error('Export error:', error);
    } finally {
        setLoading(false);
    }
};

// Импорт
const handleImport = async (event) => {
    const file = event.target.files[0];
    if (!file) return;

    setImportLoading(true);
    setImportResult(null);

    const formData = new FormData();
    formData.append('file', file);

    try {
        const response = await apiClient.post('/import/tasks/', formData, {
```

```
            headers: { 'Content-Type': 'multipart/form-data' },
          });
          setImportResult({ type: 'success', message: response.data.message, details: response.data.details });
        } catch (error) {
          setImportResult({ type: 'error', message: error.response?.data?.error || 'Ошибка импорта' });
        } finally {
          setImportLoading(false);
          event.target.value = '';
        }
      };

      // Отправка еженедельного отчета
      const handleSendReport = async () => {
        setReportLoading(true);
        try {
          await apiClient.post('/report/weekly/');
          setImportResult({ type: 'success', message: 'Отчет отправлен на ваш email' });
        } catch (error) {
          setImportResult({ type: 'error', message: 'Ошибка отправки отчета' });
        } finally {
          setReportLoading(false);
          setTimeout(() => setImportResult(null), 3000);
        }
      };

      return (
        <Box>
          <Grid container spacing={3}>
            {/* Экспорт */}
            <Grid item xs={12} md={6}>
```

```
                    <Paper sx={{ p: 3 }}>
                    <Typography variant="h6" gutterBottom display="flex" alignItems="center"
gap={1}>
                        <DownloadIcon /> Экспорт данных
                    </Typography>

                    <FormControl fullWidth sx={{ mb: 2 }}>
                        <InputLabel>Формат экспорта</InputLabel>
                        <Select
                            value={exportFormat}
                            onChange={(e) => setExportFormat(e.target.value)}
                            label="Формат экспорта"
                        >
                            <MenuItem value="csv">
                                <CsvIcon sx={{ mr: 1 }} /> CSV (Excel, Google Sheets)
                            </MenuItem>
                            <MenuItem value="json">
                                <JsonIcon sx={{ mr: 1 }} /> JSON
                            </MenuItem>
                            <MenuItem value="pdf">
                                <PdfIcon sx={{ mr: 1 }} /> PDF (Отчет)
                            </MenuItem>
                        </Select>
                    </FormControl>

                    {exportFormat === 'csv' && (
                        <Box sx={{ mb: 2 }}>
                            <Typography variant="body2" gutterBottom>Поля для
экспорта:</Typography>
                            <FormGroup row>
                                {EXPORT_FIELDS.map((field) => (
                                    <FormControlLabel
```

```
                key={field.id}
                control={
                  <Checkbox
                    checked={selectedFields.includes(field.id)}
                    onChange={(e) => {
                      if (e.target.checked) {
                        setSelectedFields([...selectedFields, field.id]);
                      } else {
                        setSelectedFields(selectedFields.filter(f => f !== field.id));
                      }
                    }}
                    size="small"
                  />
                }
                label={field.label}
              />
            ))}
          </FormGroup>
        </Box>
      ))}

      <Typography variant="body2" color="text.secondary" sx={{ mb: 2 }}>
        Фильтры экспорта:
      </Typography>
      <Box display="flex" flexWrap="wrap" gap={2} sx={{ mb: 3 }}>
        <TextField
          select
          label="Статус"
          value={exportFilters.status}
          onChange={(e) => setExportFilters({ ...exportFilters, status: e.target.value })}
          size="small"
```

```
            sx={{ minWidth: 120 }}
          >
            <MenuItem value="">Все</MenuItem>
            <MenuItem value="New">Новые</MenuItem>
            <MenuItem value="InProgress">В работе</MenuItem>
            <MenuItem value="Completed">Выполнены</MenuItem>
          </TextField>

          <TextField
            select
            label="Приоритет"
            value={exportFilters.priority}
            onChange={(e) => setExportFilters({ ...exportFilters, priority:
e.target.value })}
            size="small"
            sx={{ minWidth: 120 }}
          >
            <MenuItem value="">Все</MenuItem>
            <MenuItem value="Low">Низкий</MenuItem>
            <MenuItem value="Medium">Средний</MenuItem>
            <MenuItem value="High">Высокий</MenuItem>
          </TextField>
        </Box>

        <Button
          variant="contained"
          onClick={handleExport}
          disabled={loading}
          fullWidth
          startIcon={loading ? <CircularProgress size={20} /> : <DownloadIcon
/>}
        >
```

```
                {loading ? 'Экспорт...' : 'Скачать'}
              </Button>
            </Paper>
          </Grid>

          {/* Импорт */}
          <Grid item xs={12} md={6}>
            <Paper sx={{ p: 3 }}>
             <Typography variant="h6" gutterBottom display="flex" alignItems="center"
gap={1}>
                <UploadIcon /> Импорт данных
              </Typography>

              <Box sx={{ mb: 2 }}>
                <Typography variant="body2" color="text.secondary" gutterBottom>
                  Поддерживаемые форматы: CSV, JSON
                </Typography>
                <Typography variant="caption" color="text.secondary">
                  CSV должен содержать колонки: Заголовок, Описание, Статус,
Приоритет, Дедлайн
                </Typography>
              </Box>

              <Button
                variant="outlined"
                component="label"
                disabled={importLoading}
                fullWidth
               startIcon={importLoading ? <CircularProgress size={20} /> : <UploadIcon
/>}
                sx={{ mb: 2 }}
              >
```

```
                {importLoading ? 'Импорт...' : 'Выбрать файл'}
                <input type="file" hidden accept=".csv,.json" onChange={handleImport}
/>
              </Button>

              <Button
                variant="outlined"
                onClick={handleSendReport}
                disabled={reportLoading}
                fullWidth
              startIcon={reportLoading ? <CircularProgress size={20} /> : <EmailIcon
/>}
              >
                {reportLoading ? 'Отправка...' : 'Отправить еженедельный отчет на
email'}
              </Button>

              {importResult && (
                <Alert severity={importResult.type} sx={{ mt: 2 }}>
                  {importResult.message}
                  {importResult.details && (
                    <Box mt={1}>
                      <Typography variant="caption" component="div">
                        Создано: {importResult.details.created}, Обновлено:
{importResult.details.updated}
                      </Typography>
                      {importResult.details.errors.length > 0 && (
                        <Typography variant="caption" color="error" component="div">
                          Ошибки: {importResult.details.errors.length}
                        </Typography>
                      )}
                    </Box>
```

```
                )}
              </Alert>
            )}
          </Paper>
        </Grid>
      </Grid>
    </Box>
  );
};

export default ExportPanel;
```

Контрольные вопросы

1. Какие форматы экспорта данных поддерживаются и для каких целей они используются?
2. Как генерируется PDF-отчет с помощью ReportLab?
3. Как реализован импорт задач из CSV и JSON?
4. Как организована асинхронная отправка email-отчетов через Celery?
5. Как обрабатываются ошибки при импорте данных?
6. Какие поля можно выбрать для экспорта в CSV?

Критерии оценки

Критерий	Баллы
Реализован экспорт в CSV и JSON	2
Реализована генерация PDF-отчетов	2
Реализован импорт из CSV и JSON	2

Критерий	Баллы
Реализован выбор полей для экспорта	1
Реализована отправка отчета по email	1
Создан компонент ExportPanel	2
Итого	10

Результат выполнения

После выполнения лабораторной работы №14 студенты должны получить:

1. Функции экспорта данных в CSV, JSON и PDF форматах
2. Функции импорта задач из CSV и JSON файлов
3. Автоматическую отправку еженедельного отчета на email
4. Компонент ExportPanel с выбором форматов и полей
5. Валидацию и обработку ошибок при импорте

Лабораторная работа №15

Интеграция с внешними API. Погода и календари

Цель работы: Интеграция сторонних сервисов через их API: получение данных о погоде (OpenWeatherMap), синхронизация с Google Calendar API, интеграция с другими внешними сервисами для расширения функциональности приложения.

Задачи

№	Задача
1	Интегрировать API погоды (OpenWeatherMap) для отображения погоды на главном экране
2	Интегрировать Google Calendar API для импорта событий
3	Реализовать синхронизацию задач с внешним календарем (экспорт/импорт)
4	Добавить отображение цитат дня / мотивационных сообщений (API Quotable)
5	Реализовать кэширование запросов к внешним API

Теоретическая часть

Внешние API для интеграции

Сервис	**API**	**Тип данных**	**Лимиты**
OpenWeatherMap	weather	Погода	60 запросов/мин (бесплатно)
Google Calendar	calendar/v3	События	1 000 000 запросов/день
Quotable	/random	Цитаты	Безлимитно
Unsplash	/photos	Изображения	50 запросов/час (бесплатно)

OAuth 2.0 Flow

1. Пользователь кликает "Подключить Google Calendar"
2. Перенаправление на страницу авторизации Google

3. Пользователь дает разрешение

4. Google возвращает authorization code

5. Сервер обменивает code на access_token и refresh_token

6. Сервер сохраняет токены в БД

7. При необходимости сервер обновляет токен через refresh_token

Кэширование внешних запросов

Для уменьшения количества запросов к внешним API используется кэширование:

1. Погода - кэш 30 минут
2. Цитаты - кэш 1 час
3. Календарь - кэш 5 минут (но синхронизация по запросу)

Ход выполнения работы

Шаг 1. Установка необходимых пакетов

```
# Бэкенд
pip install requests
pip install google-auth google-auth-oauthlib google-auth-httplib2
pip install google-api-python-client
pip install django-cors-headers

# Фронтенд
npm install axios
```

Шаг 2. Создание сервиса для работы с внешними API

tasks/services/external_api_service.py:

```
import requests
import json
from datetime import datetime  timedelta
from django core cache import cache
```

```
from django conf import settings
from google oauth2 credentials import Credentials
from googleapiclient discovery import build
from google auth transport requests import Request


class WeatherService:
    """Сервис для работы с OpenWeatherMap API"""

    BASE_URL = "https://api.openweathermap.org/data/2.5"

    @classmethod
    def get_weather(cls, city="Moscow", units="metric"):
        """Получение текущей погоды"""
        cache_key = f"weather:{city}:{units}"
        cached = cache get(cache_key)
        if cached:
            return cached

        params = {
            'q': city,
            'appid': settings OPENWEATHER_API_KEY,
            'units': units,
            'lang': 'ru'
        }

        try:
            response = requests get(f"{cls BASE_URL}/weather", params=params
timeout=10)
            response raise_for_status()
            data = response json()
```

```
            result = {
                'city': data.get('name', city),
                'temperature': round(data['main']['temp']),
                'feels_like': round(data['main']['feels_like']),
                'humidity': data['main']['humidity'],
                'pressure': data['main']['pressure'],
                'wind_speed': data['wind']['speed'],
                'description': data['weather'][0]['description'].capitalize(),
                'icon': data['weather'][0]['icon'],
                'sunrise': datetime.fromtimestamp(data['sys']['sunrise']).strftime('%H:%M'),
                'sunset': datetime.fromtimestamp(data['sys']['sunset']).strftime('%H:%M'),
            }

            # Кэшируем на 30 минут
            cache.set(cache_key, result, timeout=1800)
            return result

        except requests.RequestException as e:
            return {'error': str(e), 'city': city}

    @classmethod
    def get_forecast(cls, city="Moscow", units="metric", days=5):
        """Получение прогноза погоды на несколько дней"""
        cache_key = f"forecast:{city}:{units}:{days}"
        cached = cache.get(cache_key)
        if cached:
            return cached

        params = {
            'q': city,
            'appid': settings.OPENWEATHER_API_KEY,
            'units': units,
```

```
            'lang': 'ru',
            'cnt': days * 8  # 8 измерений в день
        }

        try:
            response = requests.get(f"{cls.BASE_URL}/forecast", params=params,
timeout=10)
            response.raise_for_status()
            data = response.json()

            # Группировка по дням
            forecast_by_day = {}
            for item in data['list']:
                date = item['dt_txt'].split(' ')[0]
                if date not in forecast_by_day:
                    forecast_by_day[date] = []
                forecast_by_day[date].append(item)

            result = []
            for date, items in list(forecast_by_day.items())[:days]:
                temps = [item['main']['temp'] for item in items]
                result.append({
                    'date': date,
                    'temp_min': round(min(temps)),
                    'temp_max': round(max(temps)),
                    'description': items[0]['weather'][0]['description'].capitalize(),
                    'icon': items[0]['weather'][0]['icon'],
                })

            cache.set(cache_key, result, timeout=3600)
            return result
```

```
        except requests.RequestException as e:
            return {'error': str(e)}


class QuoteService:
    """Сервис для получения мотивационных цитат"""

    @classmethod
    def get_random_quote(cls):
        """Получение случайной цитаты"""
        cache_key = "random_quote"
        cached = cache.get(cache_key)
        if cached:
            return cached

        try:
            # API Quotable
            response = requests.get("https://api.quotable.io/random", timeout=5)
            response.raise_for_status()
            data = response.json()

            result = {
                'content': data['content'],
                'author': data['author'],
            }

            # Кэшируем на 1 час
            cache.set(cache_key, result, timeout=3600)
            return result

        except requests.RequestException:
            # Fallback цитата
```

```
            return {
                'content': 'Начинай делать то, что нужно. Затем делай то, что
возможно. И внезапно ты начнёшь делать невозможное.',
                'author': 'Франциск Ассизский'
            }


class GoogleCalendarService:
    """Сервис для работы с Google Calendar API"""

    SCOPES = ['https://www.googleapis.com/auth/calendar']

    @classmethod
    def get_credentials(cls, user):
        """Получение учетных данных пользователя для Google Calendar"""
        from ..models import GoogleToken

        try:
            token_data = GoogleToken.objects.get(user=user)
            credentials = Credentials(
                token=token_data.access_token,
                refresh_token=token_data.refresh_token,
                token_uri='https://oauth2.googleapis.com/token',
                client_id=settings.GOOGLE_CLIENT_ID,
                client_secret=settings.GOOGLE_CLIENT_SECRET,
                scopes=cls.SCOPES
            )

            # Обновление токена если истек
            if credentials.expired:
                credentials.refresh(Request())
                token_data.access_token = credentials.token
```

```
            token_data.save()

        return credentials
    except GoogleToken.DoesNotExist:
        return None

@classmethod
def get_calendar_service(cls, user):
    """Получение сервиса Google Calendar"""
    credentials = cls.get_credentials(user)
    if not credentials:
        return None
    return build('calendar', 'v3', credentials=credentials)

@classmethod
def list_events(cls, user, max_results=10, time_min=None):
    """Получение списка событий из календаря"""
    service = cls.get_calendar_service(user)
    if not service:
        return []

    if not time_min:
        time_min = datetime.utcnow().isoformat() + 'Z'

    try:
        events_result = service.events().list(
            calendarId='primary',
            timeMin=time_min,
            maxResults=max_results,
            singleEvents=True,
            orderBy='startTime'
        ).execute()
```

```
            events = events_result.get('items', [])
            return [
                {
                    'id': event['id'],
                    'title': event.get('summary', 'Без названия'),
                    'description': event.get('description', ''),
                    'start': event['start'].get('dateTime', event['start'].get('date')),
                    'end': event['end'].get('dateTime', event['end'].get('date')),
                    'location': event.get('location', ''),
                }
                for event in events
            ]
        except Exception as e:
            print(f"Error fetching calendar events: {e}")
            return []

    @classmethod
    def create_event(cls, user, task):
        """Создание события в календаре на основе задачи"""
        service = cls.get_calendar_service(user)
        if not service:
            return None

        event = {
            'summary': task.title,
            'description': task.description or '',
            'start': {
                'dateTime': task.deadline.isoformat() if task.deadline else
datetime.now().isoformat(),
                'timeZone': 'Europe/Moscow',
            },
```

```
            'end': {
                'dateTime': (task.deadline + timedelta(hours=1)).isoformat() if task.deadline
else (datetime.now() + timedelta(hours=1)).isoformat(),
                'timeZone': 'Europe/Moscow',
            },
        }

        try:
            event = service.events().insert(calendarId='primary', body=event).execute()
            return event.get('id')
        except Exception as e:
            print(f"Error creating calendar event: {e}")
            return None

    @classmethod
    def sync_tasks_to_calendar(cls, user, task_ids):
        """Синхронизация задач с календарем"""
        from .models import Task

        tasks = Task.objects.filter(id__in=task_ids, user=user)
        results = {'created': 0, 'failed': 0}

        for task in tasks:
            if task.calendar_event_id:
                # Обновление существующего события
                # ... логика обновления
                pass
            else:
                event_id = cls.create_event(user, task)
                if event_id:
                    task.calendar_event_id = event_id
                    task.save(update_fields=['calendar_event_id'])
```

```
                results['created'] += 1
            else:
                results['failed'] += 1

        return results


class MotivationService:
    """Сервис для получения мотивационного контента"""

    @classmethod
    def get_daily_motivation(cls):
        """Получение ежедневной мотивации"""
        cache_key = "daily_motivation"
        cached = cache.get(cache_key)
        if cached:
            return cached

        # Получаем цитату
        quote = QuoteService.get_random_quote()

        # Получаем случайное изображение (Unsplash)
        image_url = cls._get_random_nature_image()

        result = {
            'quote': quote,
            'image_url': image_url,
            'date': datetime.now().date().isoformat(),
        }

        cache.set(cache_key, result, timeout=86400)  # на 24 часа
        return result
```

```
    @classmethod
    def _get_random_nature_image(cls):
        """Получение случайного изображения природы от Unsplash"""
        try:
            response = requests.get(
                "https://api.unsplash.com/photos/random",
                params={'query': 'nature', 'orientation': 'landscape'},
                headers={'Authorization': f'Client-ID {settings.UNSPLASH_ACCESS_KEY}'},
                timeout=5
            )
            if response.status_code == 200:
                return response.json().get('urls', {}).get('regular')
        except Exception:
            pass
        return None
```

Шаг 3. Модель для хранения Google токенов

tasks/models.py (дополнение):

```
class GoogleToken(models.Model):
    """Хранение токенов Google OAuth"""
    user = models.OneToOneField(
        User,
        on_delete=models.CASCADE,
        related_name='google_token'
    )
    access_token = models.TextField()
    refresh_token = models.TextField()
    expires_at = models.DateTimeField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

```
    class Meta:
        db_table = 'google_tokens'

    def __str__(self):
        return f"Google token for {self.user.username}"
```

Шаг 4. Добавление поля calendar_event_id в Task

tasks/models.py (дополнение):

```
class Task(models.Model):
    # ... существующие поля
    calendar_event_id = models.CharField(max_length=255, null=True, blank=True)
```

Шаг 5. Создание миграций

```
python manage.py makemigrations
python manage.py migrate
```

Шаг 6. Создание API эндпоинтов

tasks/views/external_views.py:

```
python
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework.permissions import IsAuthenticated
from drf_spectacular.utils import extend_schema
from ..services.external_api_service import (
    WeatherService, QuoteService, GoogleCalendarService, MotivationService
)
from ..models import Task


class WeatherView(APIView):
```

```
    """API для получения погоды"""
    permission_classes = [IsAuthenticated]

    @extend_schema(summary="Получить текущую погоду")
    def get(self, request):
        city = request.query_params.get('city', 'Moscow')
        units = request.query_params.get('units', 'metric')
        forecast = request.query_params.get('forecast', 'false').lower() == 'true'

        if forecast:
            data = WeatherService.get_forecast(city, units)
        else:
            data = WeatherService.get_weather(city, units)

        return Response(data)


class QuoteView(APIView):
    """API для получения мотивационной цитаты"""
    permission_classes = [IsAuthenticated]

    @extend_schema(summary="Получить случайную цитату")
    def get(self, request):
        quote = QuoteService.get_random_quote()
        return Response(quote)


class MotivationView(APIView):
    """API для получения ежедневной мотивации"""
    permission_classes = [IsAuthenticated]

    @extend_schema(summary="Получить ежедневную мотивацию")
```

```
    def get(self, request):
        motivation = MotivationService.get_daily_motivation()
        return Response(motivation)


class GoogleCalendarAuthView(APIView):
    """API для авторизации Google Calendar"""
    permission_classes = [IsAuthenticated]

    @extend_schema(summary="Получить URL для авторизации Google")
    def get(self, request):
        from google_auth_oauthlib.flow import Flow
        from django.conf import settings

        flow = Flow.from_client_config(
            {
                "web": {
                    "client_id": settings.GOOGLE_CLIENT_ID,
                    "client_secret": settings.GOOGLE_CLIENT_SECRET,
                    "auth_uri": "https://accounts.google.com/o/oauth2/auth",
                    "token_uri": "https://oauth2.googleapis.com/token",
                }
            },
            scopes=GoogleCalendarService.SCOPES,
            redirect_uri=settings.GOOGLE_REDIRECT_URI
        )

        auth_url, state = flow.authorization_url(
            access_type='offline',
            include_granted_scopes='true',
            prompt='consent'
        )
```

```
        request.session['google_oauth_state'] = state
        return Response({'auth_url': auth_url})


class GoogleCalendarCallbackView(APIView):
    """Callback для Google OAuth"""
    permission_classes = [IsAuthenticated]

    def get(self, request):
        from google_auth_oauthlib.flow import Flow
        from ..models import GoogleToken

        code = request.query_params.get('code')
        state = request.query_params.get('state')

        if not code:
            return Response({'error': 'No code provided'}, status=400)

        flow = Flow.from_client_config(
            {
                "web": {
                    "client_id": settings.GOOGLE_CLIENT_ID,
                    "client_secret": settings.GOOGLE_CLIENT_SECRET,
                    "auth_uri": "https://accounts.google.com/o/oauth2/auth",
                    "token_uri": "https://oauth2.googleapis.com/token",
                }
            },
            scopes=GoogleCalendarService.SCOPES,
            redirect_uri=settings.GOOGLE_REDIRECT_URI
        )
```

```
        flow.fetch_token(code=code)
        credentials = flow.credentials

        # Сохранение токенов
        GoogleToken.objects.update_or_create(
            user=request.user,
            defaults={
                'access_token': credentials.token,
                'refresh_token': credentials.refresh_token,
                'expires_at': datetime.fromtimestamp(credentials.expiry.timestamp()),
            }
        )

        return Response({'message': 'Google Calendar connected successfully'})


class GoogleCalendarEventsView(APIView):
    """API для получения событий из Google Calendar"""
    permission_classes = [IsAuthenticated]

    @extend_schema(summary="Получить события из Google Calendar")
    def get(self, request):
        events = GoogleCalendarService.list_events(request.user, max_results=20)
        return Response(events)


class SyncTasksToCalendarView(APIView):
    """API для синхронизации задач с календарем"""
    permission_classes = [IsAuthenticated]

    @extend_schema(summary="Синхронизировать задачи с Google Calendar")
    def post(self, request):
```

```
        task_ids = request.data.get('task_ids', [])
        if not task_ids:
            tasks = Task.objects.filter(user=request.user, deadline__isnull=False)
            task_ids = list(tasks.values_list('id', flat=True))

        result = GoogleCalendarService.sync_tasks_to_calendar(request.user, task_ids)
        return Response(result)
```

Шаг 7. Добавление маршрутов

tasks/urls.py (дополнение):

```
from .views.external_views import (
    WeatherView, QuoteView, MotivationView,
    GoogleCalendarAuthView, GoogleCalendarCallbackView,
    GoogleCalendarEventsView, SyncTasksToCalendarView
)

urlpatterns += [
    # Погода
    path('api/weather/', WeatherView.as_view(), name='weather'),
    path('api/quote/', QuoteView.as_view(), name='quote'),
    path('api/motivation/', MotivationView.as_view(), name='motivation'),

    # Google Calendar
    path('api/calendar/auth/', GoogleCalendarAuthView.as_view(), name='calendar-
auth'),
    path('api/calendar/callback/', GoogleCalendarCallbackView.as_view(),
name='calendar-callback'),
    path('api/calendar/events/', GoogleCalendarEventsView.as_view(),
name='calendar-events'),
    path('api/calendar/sync/', SyncTasksToCalendarView.as_view(), name='calendar-
sync'),
```

```
]
```

Шаг 8. Фронтенд: компонент погоды

src/components/weather/WeatherWidget.jsx:

```
import React, { useState, useEffect } from 'react';
import {
  Paper,
  Box,
  Typography,
  IconButton,
  CircularProgress,
  TextField,
  MenuItem,
} from '@mui/material';
import {
  Refresh as RefreshIcon,
  WbSunny as SunIcon,
  Cloud as CloudIcon,
  Grain as RainIcon,
  AcUnit as SnowIcon,
  Thunderstorm as StormIcon,
  Opacity as HumidityIcon,
  Air as WindIcon,
  Thermostat as TempIcon,
} from '@mui/icons-material';
import apiClient from '../../api/client';

const weatherIcons = {
  '01d': <SunIcon sx={{ fontSize: 48 }} />,
  '01n': <SunIcon sx={{ fontSize: 48 }} />,
  '02d': <CloudIcon sx={{ fontSize: 48 }} />,
```

```
    '02n': <CloudIcon sx={{ fontSize: 48 }} />,
    '03d': <CloudIcon sx={{ fontSize: 48 }} />,
    '03n': <CloudIcon sx={{ fontSize: 48 }} />,
    '04d': <CloudIcon sx={{ fontSize: 48 }} />,
    '04n': <CloudIcon sx={{ fontSize: 48 }} />,
    '09d': <RainIcon sx={{ fontSize: 48 }} />,
    '09n': <RainIcon sx={{ fontSize: 48 }} />,
    '10d': <RainIcon sx={{ fontSize: 48 }} />,
    '10n': <RainIcon sx={{ fontSize: 48 }} />,
    '11d': <StormIcon sx={{ fontSize: 48 }} />,
    '11n': <StormIcon sx={{ fontSize: 48 }} />,
    '13d': <SnowIcon sx={{ fontSize: 48 }} />,
    '13n': <SnowIcon sx={{ fontSize: 48 }} />,
    '50d': <CloudIcon sx={{ fontSize: 48 }} />,
    '50n': <CloudIcon sx={{ fontSize: 48 }} />,
};

const cities = [
    { value: 'Moscow', label: 'Москва' },
    { value: 'Saint Petersburg', label: 'Санкт-Петербург' },
    { value: 'Kazan', label: 'Казань' },
    { value: 'Novosibirsk', label: 'Новосибирск' },
    { value: 'Yekaterinburg', label: 'Екатеринбург' },
    { value: 'London', label: 'Лондон' },
    { value: 'Paris', label: 'Париж' },
    { value: 'Berlin', label: 'Берлин' },
    { value: 'New York', label: 'Нью-Йорк' },
    { value: 'Tokyo', label: 'Токио' },
];

const WeatherWidget = () => {
    const [weather, setWeather] = useState(null);
```

```
const [loading, setLoading] = useState(true);
const [city, setCity] = useState('Moscow');
const [forecast, setForecast] = useState(null);

useEffect(() => {
  fetchWeather();
}, [city]);

const fetchWeather = async () => {
  setLoading(true);
  try {
    const [weatherRes, forecastRes] = await Promise.all([
      apiClient.get('/weather/', { params: { city } }),
      apiClient.get('/weather/', { params: { city, forecast: true } }),
    ]);
    setWeather(weatherRes.data);
    setForecast(forecastRes.data);
  } catch (error) {
    console.error('Error fetching weather:', error);
  } finally {
    setLoading(false);
  }
};

if (loading) {
  return (
    <Paper sx={{ p: 2, display: 'flex', justifyContent: 'center' }}>
      <CircularProgress size={32} />
    </Paper>
  );
}
```

```
  if (weather?.error) {
    return (
      <Paper sx={{ p: 2 }}>
        <Typography color="error">Ошибка загрузки погоды</Typography>
      </Paper>
    );
  }

  return (
    <Paper sx={{ p: 2 }}>
      <Box display="flex" justifyContent="space-between" alignItems="center" mb={2}>
        <TextField
          select
          size="small"
          value={city}
          onChange={(e) => setCity(e.target.value)}
          sx={{ minWidth: 150 }}
        >
          {cities.map(c => (
            <MenuItem key={c.value} value={c.value}>{c.label}</MenuItem>
          ))}
        </TextField>
        <IconButton onClick={fetchWeather} size="small">
          <RefreshIcon />
        </IconButton>
      </Box>

      {weather && (
        <>
          <Box display="flex" alignItems="center" gap={2} mb={2}>
            {weatherIcons[weather.icon] || <SunIcon sx={{ fontSize: 48 }} />}
```

```
                <Box>
                  <Typography variant="h3">
                    {weather.temperature}°C
                  </Typography>
                  <Typography variant="body2" color="text.secondary">
                    Ощущается как {weather.feels_like}°C
                  </Typography>
                </Box>
              </Box>

              <Typography variant="body1" gutterBottom>
                {weather.description}
              </Typography>

              <Box display="flex" gap={2} mt={2}>
                <Box display="flex" alignItems="center" gap={0.5}>
                  <HumidityIcon fontSize="small" color="action" />
                  <Typography variant="body2">{weather.humidity}%</Typography>
                </Box>
                <Box display="flex" alignItems="center" gap={0.5}>
                  <WindIcon fontSize="small" color="action" />
                  <Typography variant="body2">{weather.wind_speed}
м/с</Typography>
                </Box>
                <Box display="flex" alignItems="center" gap={0.5}>
                  <TempIcon fontSize="small" color="action" />
                  <Typography variant="body2">{weather.pressure}
гПа</Typography>
                </Box>
              </Box>
```

```
                    <Box display="flex" justifyContent="space-between" mt={2} pt={1}
borderTop="1px solid #eee">
                      <Typography variant="caption">Восход:
{weather.sunrise}</Typography>
                      <Typography variant="caption">Закат:
{weather.sunset}</Typography>
                    </Box>
                  </>
                )}

                {forecast && !forecast.error && (
                  <Box mt={2} pt={2} borderTop="1px solid #eee">
                    <Typography variant="subtitle2" gutterBottom>Прогноз на 5
дней</Typography>
                    <Box display="flex" justifyContent="space-between">
                      {forecast.slice(0, 5).map((day, idx) => (
                        <Box key={idx} textAlign="center">
                          <Typography variant="caption">
                            {day.date.slice(5)}
                          </Typography>
                          {weatherIcons[day.icon]}
                          <Typography variant="caption" display="block">
                            {day.temp_min}°/{day.temp_max}°
                          </Typography>
                        </Box>
                      ))}
                    </Box>
                  </Box>
                )}
              </Paper>
            );
          };
```

```
export default WeatherWidget
```

Шаг 9. Фронтенд: компонент мотивационной цитаты

src/components/motivation/MotivationCard.jsx:

```
import React, { useState, useEffect } from 'react';
import { Paper, Typography, Box, IconButton, CircularProgress } from
'@mui/material';
import { Refresh as RefreshIcon, FormatQuote as QuoteIcon } from '@mui/icons-
material';
import apiClient from '../../api/client';

const MotivationCard = () => {
  const [motivation, setMotivation] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    fetchMotivation();
  }, []);

  const fetchMotivation = async () => {
    setLoading(true);
    try {
      const response = await apiClient.get('/motivation/');
      setMotivation(response.data);
    } catch (error) {
      console.error('Error fetching motivation:', error);
    } finally {
      setLoading(false);
    }
  };
```

```
  if (loading) {
    return (
      <Paper sx={{ p: 2, display: 'flex', justifyContent: 'center' }}>
        <CircularProgress size={24} />
      </Paper>
    );
  }

  if (!motivation) return null;

  return (
    <Paper
      sx={{
        p: 3,
        position: 'relative',
        overflow: 'hidden',
        background: 'linear-gradient(135deg, #667eea 0%, #764ba2 100%)',
        color: 'white',
      }}
    >
      <QuoteIcon sx={{ position: 'absolute', bottom: 10, right: 10, opacity: 0.2,
fontSize: 80 }} />

      <Box display="flex" justifyContent="flex-end">
        <IconButton onClick={fetchMotivation} sx={{ color: 'white' }} size="small">
          <RefreshIcon />
        </IconButton>
      </Box>

      <Typography variant="body1" fontStyle="italic" sx={{ mb: 2 }}>
        «{motivation.quote?.content}»
```

```
        </Typography>

        <Typography variant="caption" sx={{ opacity: 0.8 }}>
          — {motivation.quote?.author}
        </Typography>

        {motivation.image_url && (
          <Box
            component="img"
            src={motivation.image_url}
            sx={{
              width: '100%',
              height: 120,
              objectFit: 'cover',
              borderRadius: 1,
              mt: 2,
            }}
          />
        )}
      </Paper>
    );
};

export default MotivationCard;
```

Шаг 10. Фронтенд: компонент синхронизации с календарем

src/components/calendar/GoogleCalendarSync.jsx:

```
import React, { useState } from 'react';
import {
  Button,
  Dialog
```

```
    DialogTitle,
    DialogContent,
    DialogActions,
    List,
    ListItem,
    ListItemText,
    ListItemAvatar,
    Avatar,
    Chip,
    Box,
    Typography,
    Alert,
    CircularProgress,
} from '@mui/material';
import {
    Google as GoogleIcon,
    Event as EventIcon,
    Sync as SyncIcon,
    CheckCircle as CheckIcon,
} from '@mui/icons-material';
import apiClient from '../../api/client';

const GoogleCalendarSync = ({ tasks, onSyncComplete }) => {
    const [open, setOpen] = useState(false);
    const [loading, setLoading] = useState(false);
    const [events, setEvents] = useState([]);
    const [syncResult, setSyncResult] = useState(null);

    const handleConnect = async () => {
        try {
            const response = await apiClient.get('/calendar/auth/');
            window.location.href = response.data.auth_url;
```

```
    } catch (error) {
      console.error('Error connecting to Google Calendar:', error);
    }
  };

  const handleFetchEvents = async () => {
    setLoading(true);
    try {
      const response = await apiClient.get('/calendar/events/');
      setEvents(response.data);
    } catch (error) {
      console.error('Error fetching events:', error);
    } finally {
      setLoading(false);
    }
  };

  const handleSyncTasks = async () => {
    setLoading(true);
    try {
      const taskIds = tasks.filter(t => t.deadline).map(t => t.id);
      const response = await apiClient.post('/calendar/sync/', { task_ids: taskIds });
      setSyncResult(response.data);
      onSyncComplete?.();
    } catch (error) {
      console.error('Error syncing tasks:', error);
    } finally {
      setLoading(false);
    }
  };

  const handleOpen = () => {
```

```
    setOpen(true);
    handleFetchEvents();
  };

  return (
    <>
      <Button
        variant="outlined"
        startIcon={<GoogleIcon />}
        onClick={handleOpen}
      >
        Google Календарь
      </Button>

      <Dialog open={open} onClose={() => setOpen(false)} maxWidth="md"
fullWidth>
        <DialogTitle>Google Календарь</DialogTitle>
        <DialogContent>
          {syncResult && (
            <Alert severity="success" sx={{ mb: 2 }}>
              Синхронизировано задач: {syncResult.created}
            </Alert>
          )}

          <Box display="flex" gap={2} mb={3}>
            <Button
              variant="contained"
              startIcon={<GoogleIcon />}
              onClick={handleConnect}
            >
              Подключить аккаунт
            </Button>
```

```
    <Button
      variant="outlined"
      startIcon={<SyncIcon />}
      onClick={handleSyncTasks}
      disabled={loading}
    >
      Синхронизировать задачи
    </Button>
  </Box>

  <Typography variant="subtitle2" gutterBottom>
    События из календаря:
  </Typography>

  {loading ? (
    <Box display="flex" justifyContent="center" py={4}>
      <CircularProgress />
    </Box>
  ) : (
    <List>
      {events.map((event) => (
        <ListItem key={event.id} divider>
          <ListItemAvatar>
            <Avatar>
              <EventIcon />
            </Avatar>
          </ListItemAvatar>
          <ListItemText
            primary={event.title}
            secondary={
              <>
                <Typography variant="caption" display="block">
```

```
                                        {new Date(event.start).toLocaleString('ru-RU')}
                                      </Typography>
                                      {event.description && (
                                        <Typography variant="caption"
color="text.secondary">
                                          {event.description.length > 100
                                            ? `${event.description.substring(0, 100)}...`
                                            : event.description}
                                        </Typography>
                                      )}
                                    </>
                                  }
                                />
                              </ListItem>
                            ))}
                            {events.length === 0 && !loading && (
                              <Typography color="text.secondary" textAlign="center" py={4}>
                                Нет событий в календаре
                              </Typography>
                            )}
                          </List>
                        )}
                      </DialogContent>
                      <DialogActions>
                        <Button onClick={() => setOpen(false)}>Закрыть</Button>
                      </DialogActions>
                    </Dialog>
                  </>
                );
              };

              export default GoogleCalendarSync;
```

Контрольные вопросы

1. Как работает аутентификация с Google Calendar API?
2. Зачем нужно кэшировать ответы от внешних API?
3. Как обрабатывать ошибки при недоступности внешнего сервиса?
4. Какие данные о погоде можно получить из OpenWeatherMap?
5. Как реализована синхронизация задач с календарем?
6. Какие меры безопасности нужно учитывать при работе с OAuth?

Критерии оценки

Критерий	Баллы
Интегрирован API погоды (OpenWeatherMap)	2
Интегрирован Google Calendar API	2
Реализована синхронизация задач с календарем	2
Добавлен виджет мотивационных цитат	2
Реализовано кэширование запросов	1
Реализована обработка ошибок API	1
Итого	10

Результат выполнения

После выполнения лабораторной работы №15 студенты должны получить:

1. Виджет погоды с текущей погодой и прогнозом
2. Интеграцию с Google Calendar (авторизация, чтение событий)
3. Синхронизацию задач с календарем

4. Виджет мотивационных цитат с изображениями
5. Кэширование всех внешних запросов

Лабораторная работа №16

Контейнеризация с Docker

Цель работы: Контейнеризация приложения (Django/ASP.NET + React + PostgreSQL + Redis + Nginx) с помощью Docker и Docker Compose для упрощения развертывания и обеспечения воспроизводимости окружения.

Задачи

№	Задача
1	Создать Dockerfile для backend-приложения (Django/ASP.NET)
2	Создать Dockerfile для frontend-приложения (React) с многостадийной сборкой
3	Настроить docker-compose для всех сервисов (backend, frontend, PostgreSQL, Redis, Nginx)
4	Настроить volumes для персистентности данных базы данных
5	Настроить переменные окружения через .env файл

Теоретическая часть

Docker - платформа для разработки, доставки и запуска приложений в изолированных контейнерах.

Ключевые понятия:

Понятие	Описание
Образ (Image)	Шаблон для создания контейнеров (содержит код, зависимости, настройки)
Контейнер (Container)	Запущенный экземпляр образа
Dockerfile	Файл с инструкциями для сборки образа
Docker Compose	Инструмент для запуска многоконтейнерных приложений
Volume	Механизм для хранения данных вне контейнера
Network	Сеть для взаимодействия контейнеров

Многостадийная сборка (Multi-stage build)

Многостадийная сборка позволяет создавать минимальные образы, разделяя этапы сборки и запуска.

dockerfile

```
# Стадия 1: Сборка
FROM node:18 AS builder
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build

# Стадия 2: Запуск
FROM nginx:alpine
COPY --from=builder /app/dist /usr/share/nginx/html
```

Структура проекта для контейнеризации

```
project/
├── backend/
│   ├── Dockerfile
│   ├── requirements.txt
│   └── ...
├── frontend/
│   ├── Dockerfile
│   ├── package.json
│   └── ...
├── nginx/
│   └── nginx.conf
├── docker-compose.yml
├── .env
└── .dockerignore
```

Ход выполнения работы

Траектория A: Django + React

Шаг 1. Создание Dockerfile для backend (Django)

backend/Dockerfile:

dockerfile

```
# ============================================
# Стадия 1: Базовый образ с зависимостями
# ============================================
FROM python:3.11-slim as builder

# Установка системных зависимостей
RUN apt-get update && apt-get install -y --no-install-recommends \
    gcc \
```

```
    libpq-dev \
    && rm -rf /var/lib/apt/lists/*

# Установка рабочей директории
WORKDIR /app

# Копирование зависимостей
COPY requirements.txt .

# Установка Python-зависимостей
RUN pip install --no-cache-dir -r requirements.txt

# ==============================================
# Стадия 2: Финальный образ
# ==============================================
FROM python:3.11-slim

# Установка рабочей директории
WORKDIR /app

# Копирование установленных зависимостей из builder
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

# Копирование кода приложения
COPY . .

# Создание директории для статических файлов
RUN mkdir -p /app/staticfiles /app/mediafiles
```

```
# Создание непривилегированного пользователя
RUN addgroup --system --gid 1001 appgroup && \
    adduser --system --uid 1001 --gid 1001 appuser && \
    chown -R appuser:appgroup /app

# Переключение на непривилегированного пользователя
USER appuser

# Открытие порта
EXPOSE 8000

# Команда для запуска приложения
CMD ["gunicorn", "smart_diary.wsgi:application", "--bind", "0.0.0.0:8000", "--workers", "4", "--threads", "2"]
```

Шаг 2. Создание Dockerfile для frontend (React)

frontend/Dockerfile:

dockerfile

```
# ==========================================
# Стадия 1: Сборка React-приложения
# ==========================================
FROM node:18-alpine AS builder

# Установка рабочей директории
WORKDIR /app

# Копирование файлов зависимостей
COPY package*.json ./

# Установка зависимостей
```

```
RUN npm ci --only=production

# Копирование исходного кода
COPY . .

# Сборка приложения
RUN npm run build

# ==========================================
# Стадия 2: Запуск через Nginx
# ==========================================
FROM nginx:alpine

# Копирование собранных файлов из стадии builder
COPY --from=builder /app/dist /usr/share/nginx/html

# Копирование кастомной конфигурации Nginx
COPY nginx.conf /etc/nginx/conf.d/default.conf

# Открытие порта
EXPOSE 80

# Запуск Nginx
CMD ["nginx", "-g", "daemon off;"]
```

Шаг 3. Создание конфигурации Nginx

frontend/nginx.conf:

```
server {
```

```
        listen 80;
        server_name localhost;

        root /usr/share/nginx/html;
        index index.html;

        # Gzip сжатие
        gzip on;
        gzip_types text/plain text/css application/json application/javascript text/xml application/xml application/xml+rss text/javascript;

        # Статические файлы
        location /static/ {
            alias /usr/share/nginx/html/static/;
            expires 30d;
            add_header Cache-Control "public, immutable";
        }

        # API проксирование на backend
        location /api/ {
            proxy_pass http://backend:8000;
            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;
        }

        # Админка Django
        location /admin/ {
            proxy_pass http://backend:8000;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
        }

        # WebSocket проксирование
        location /ws/ {
            proxy_pass http://backend:8000;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection "upgrade";
            proxy_set_header Host $host;
        }

        # SPA маршрутизация (все остальные запросы отдаём index.html)
        location / {
            try_files $uri $uri/ /index.html;
        }
    }
```

Шаг 4. Создание .env файла для переменных окружения

.env:

env

DEBUG=False

SECRET_KEY=your-secret-key-here-change-in-production

DJANGO_ALLOWED_HOSTS=localhost,backend,nginx

База данных

DB_NAME=smartdiary

DB_USER=postgres

DB_PASSWORD=postgres

DB_HOST=db

DB_PORT=5432

Redis

REDIS_HOST=redis

REDIS_PORT=6379

JWT

JWT_SECRET_KEY=your-jwt-secret-here

Шаг 5. Настройка settings.py для работы с Docker

smart_diary/settings.py (дополнение):

```
import os
from pathlib import Path

# Базовые настройки
BASE_DIR = Path(__file__) resolve() parent parent

# Секретный ключ из переменных окружения
SECRET_KEY = os environ get('SECRET_KEY', 'django-insecure-default-key-for-dev')
```

```
# DEBUG из переменных окружения
DEBUG = os.environ.get('DEBUG', 'True') == 'True'

# ALLOWED_HOSTS
ALLOWED_HOSTS = os.environ.get('DJANGO_ALLOWED_HOSTS', 'localhost,127.0.0.1').split(',')

# Настройки базы данных
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': os.environ.get('DB_NAME', 'smartdiary'),
        'USER': os.environ.get('DB_USER', 'postgres'),
        'PASSWORD': os.environ.get('DB_PASSWORD', 'postgres'),
        'HOST': os.environ.get('DB_HOST', 'localhost'),
        'PORT': os.environ.get('DB_PORT', '5432'),
    }
}

# Redis для кэширования
CACHES = {
    'default': {
        'BACKEND': 'django_redis.cache.RedisCache',
        'LOCATION': f"redis://{os.environ.get('REDIS_HOST', 'localhost')}:{os.environ.get('REDIS_PORT', '6379')}/1",
        'OPTIONS': {
            'CLIENT_CLASS': 'django_redis.client.DefaultClient',
        }
    }
}

# 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, 'mediafiles')

# CSRF доверенные источники
CSRF_TRUSTED_ORIGINS = ['http://localhost', 'http://nginx']
```

Шаг 6. Создание docker-compose.yml

docker-compose.yml:

```yaml
version: '3.8'

services:
  # ==========================================
  # База данных PostgreSQL
  # ==========================================
  db:
    image: postgres 15 alpine
    container_name: smartdiary_db
    restart: unless stopped
    volumes:
      - postgres_data:/var/lib/postgresql/data
    environment:
      - POSTGRES_DB=${DB_NAME}
      - POSTGRES_USER=${DB_USER}
      - POSTGRES_PASSWORD=${DB_PASSWORD}
    ports:
      - "5432:5432"
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U ${DB_USER} -d ${DB_NAME}"]
      interval: 10s
      timeout: 5s
      retries: 5
    networks:
      - smartdiary_network

  # ==========================================
  # Redis для кэширования и WebSocket
  # ==========================================
  redis:
    image: redis:7 alpine
    container_name: smartdiary_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:
      - smartdiary_network

  # ==========================================
  # Backend (Django)
```

```
  # ==========================================
  backend:
    build:
      context: ./backend
      dockerfile: Dockerfile
    container_name: smartdiary_backend
    restart: unless-stopped
    volumes:
      - ./backend:/app
      - static_volume /app/staticfiles
      - media_volume /app/mediafiles
    environment:
      - DEBUG=${DEBUG}
      - SECRET_KEY=${SECRET_KEY}
      - DJANGO_ALLOWED_HOSTS=${DJANGO_ALLOWED_HOSTS}
      - DB_NAME=${DB_NAME}
      - DB_USER=${DB_USER}
      - DB_PASSWORD=${DB_PASSWORD}
      - DB_HOST=db
      - DB_PORT=5432
      - REDIS_HOST=redis
      - REDIS_PORT=6379
    depends_on:
      db:
        condition: service_healthy
      redis:
        condition: service_healthy
    command: >
      sh -c "
        python manage.py migrate &&
        python manage.py collectstatic --noinput &&
        python manage.py loaddata fixtures/initial_data.json &&
        gunicorn smart_diary.wsgi:application --bind 0.0.0.0:8000 --workers 4 --threads
2
      "
    networks:
      - smartdiary_network

  # ==========================================
  # Celery Worker (фоновые задачи)
  # ==========================================
  celery:
    build:
      context: ./backend
      dockerfile: Dockerfile
    container_name: smartdiary_celery
    restart: unless-stopped
    volumes:
```

```
    - ./backend:/app
  environment:
    - DEBUG=${DEBUG}
    - SECRET_KEY=${SECRET_KEY}
    - DB_NAME=${DB_NAME}
    - DB_USER=${DB_USER}
    - DB_PASSWORD=${DB_PASSWORD}
    - DB_HOST=db
    - DB_PORT=5432
    - REDIS_HOST=redis
    - REDIS_PORT=6379
  depends_on:
    db:
      condition: service_healthy
    redis:
      condition: service_healthy
  command: celery -A smart_diary worker --loglevel=info
  networks:
    - smartdiary_network

# ==========================================
# Celery Beat (периодические задачи)
# ==========================================
celery-beat:
  build:
    context: ./backend
    dockerfile: Dockerfile
  container_name: smartdiary_celery_beat
  restart: unless-stopped
  volumes:
    - ./backend:/app
  environment:
    - DEBUG=${DEBUG}
    - SECRET_KEY=${SECRET_KEY}
    - DB_NAME=${DB_NAME}
    - DB_USER=${DB_USER}
    - DB_PASSWORD=${DB_PASSWORD}
    - DB_HOST=db
    - DB_PORT=5432
    - REDIS_HOST=redis
    - REDIS_PORT=6379
  depends_on:
    db:
      condition: service_healthy
    redis:
      condition: service_healthy
  command: celery -A smart_diary beat --loglevel=info
  networks:
```

```
    - smartdiary_network

  # ============================================
  # Frontend (React + Nginx)
  # ============================================
  frontend:
    build:
      context: ./frontend
      dockerfile: Dockerfile
    container_name: smartdiary_frontend
    restart: unless stopped
    depends_on:
      - backend
    ports:
      - "80:80"
    volumes:
      - static_volume /usr/share/nginx/html/static
    networks:
      - smartdiary_network

# ============================================
# Volumes для персистентности данных
# ============================================
volumes:
  postgres_data:
    driver: local
  redis_data:
    driver: local
  static_volume:
    driver: local
  media_volume:
    driver: local

# ============================================
# Сеть для внутреннего взаимодействия
# ============================================
networks:
  smartdiary_network:
    driver: bridge
```

Шаг 7. Создание .dockerignore файла

.dockerignore:

```
# Python
__pycache__/
```

```
*.py[cod]
*$py.class
*.so
.Python
env/
venv/
*.egg-info/
*.egg

# Django
*.log
db.sqlite3
staticfiles/
mediafiles/

# Node
node_modules/
npm-debug.log

# IDE
.vscode/
.idea/
*.swp

# Git
.git/
.gitignore

# Docker
Dockerfile
```

.dockerignore

```
# Environment
.env
.env.local

# Coverage
.coverage
htmlcov/

# Misc
.DS_Store
*.pyc
*.pyo
*.pyd
```

Траектория Б: ASP.NET Core + React

Шаг 1. Создание Dockerfile для backend (ASP.NET Core)

backend/Dockerfile:

dockerfile

```
# ==========================================
# Стадия 1: Build
# ==========================================
FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
WORKDIR /src

# Копирование csproj и восстановление зависимостей
COPY ["SmartDiary.Web/SmartDiary.Web.csproj", "SmartDiary.Web/"]
RUN dotnet restore "SmartDiary.Web/SmartDiary.Web.csproj"
```

```
# Копирование остального кода и сборка
COPY . .
WORKDIR "/src/SmartDiary.Web"
RUN dotnet build "SmartDiary.Web.csproj" -c Release -o /app/build

# ==========================================
# Стадия 2: Publish
# ==========================================
FROM build AS publish
RUN dotnet publish "SmartDiary.Web.csproj" -c Release -o /app/publish

# ==========================================
# Стадия 3: Runtime
# ==========================================
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS final
WORKDIR /app

# Установка curl для healthcheck
RUN apt-get update && apt-get install -y curl && rm -rf /var/lib/apt/lists/*

# Копирование опубликованных файлов
COPY --from=publish /app/publish .

# Создание непривилегированного пользователя
RUN adduser --disabled-password --gecos "" --uid 1001 appuser && \
    chown -R appuser:appuser /app
USER appuser

EXPOSE 8080
```

EXPOSE 8081

ENV ASPNETCORE_ENVIRONMENT=Production
ENV ASPNETCORE_URLS=http://+:8080

ENTRYPOINT ["dotnet", "SmartDiary.Web.dll"]

Шаг 2. Dockerfile для frontend (React)
Аналогичен траектории А.

Шаг 3. docker-compose.yml для ASP.NET
docker-compose.yml:

```yaml
version: '3.8'

services:
  db:
    image: postgres 15 alpine
    container_name: smartdiary_db
    restart: unless stopped
    volumes:
      - postgres_data:/var/lib/postgresql/data
    environment:
      - POSTGRES_DB=${DB_NAME}
      - POSTGRES_USER=${DB_USER}
      - POSTGRES_PASSWORD=${DB_PASSWORD}
    ports:
      - "5432:5432"
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U ${DB_USER} -d ${DB_NAME}"]
      interval: 10s
      timeout: 5s
      retries: 5
    networks:
      - smartdiary_network

  redis:
    image: redis:7 alpine
    container_name: smartdiary_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:
      - smartdiary_network

  backend:
    build:
      context: ./backend
      dockerfile: Dockerfile
    container_name: smartdiary_backend
    restart: unless stopped
    environment:
      - ASPNETCORE_ENVIRONMENT=Production
      -
ConnectionStrings__DefaultConnection=Host=db;Database=${DB_NAME};Username=${DB_
USER};Password=${DB_PASSWORD}
      - JWT__Secret=${JWT_SECRET}
      - JWT__Issuer=${JWT_ISSUER}
      - JWT__Audience=${JWT_AUDIENCE}
    depends_on:
      db:
        condition: service_healthy
      redis:
        condition: service_healthy
    ports:
      - "8080:8080"
    networks:
      - smartdiary_network

  frontend:
    build:
      context: ./frontend
      dockerfile: Dockerfile
    container_name: smartdiary_frontend
    restart: unless stopped
    depends_on:
      - backend
    ports:
      - "80:80"
    networks:
```

```
      - smartdiary_network

volumes:
  postgres_data:
    driver: local
  redis_data:
    driver: local

networks:
  smartdiary_network:
    driver: bridge
```

Команды для работы с Docker

Сборка и запуск

```
# Сборка образов
docker-compose build

# Сборка с отключением кэша
docker-compose build --no-cache

# Запуск всех сервисов в фоновом режиме
docker-compose up -d

# Запуск с логами
docker-compose up

# Просмотр логов
docker-compose logs -f

# Остановка сервисов
docker-compose down

# Остановка с удалением volumes (удаляет данные БД)
```

```
docker-compose down -v
```

Управление сервисами

```
bash
# Просмотр запущенных контейнеров
docker-compose ps

# Перезапуск конкретного сервиса
docker-compose restart backend

# Выполнение команды в контейнере
docker-compose exec backend python manage.py createsuperuser

# Просмотр использования ресурсов
docker stats

# Очистка неиспользуемых образов и контейнеров
docker system prune -a
```

Миграции и обслуживание

```
# Выполнение миграций
docker-compose exec backend python manage.py migrate

# Сбор статических файлов
docker-compose exec backend python manage.py collectstatic --noinput

# Создание дампа базы данных
docker-compose exec db pg_dump -U postgres smartdiary > backup.sql

# Восстановление базы данных
```

```
cat backup.sql | docker-compose exec -T db psql -U postgres smartdiary
```

Контрольные вопросы

1. Что такое Docker и для чего он используется?
2. Чем отличается образ от контейнера?
3. Что такое многостадийная сборка и зачем она нужна?
4. Как в Docker Compose организована связь между контейнерами?
5. Для чего нужны volumes и как они работают?
6. Как передать переменные окружения в контейнер?
7. Какие команды Docker используются для управления контейнерами?

Критерии оценки

Критерий	**Баллы**
Создан Dockerfile для backend	2
Создан Dockerfile для frontend с многостадийной сборкой	2
Настроен docker-compose для всех сервисов	2
Настроены volumes для персистентности данных	1
Настроены переменные окружения через .env	1
Приложение успешно запускается в контейнерах	2
Итого	10

Результат выполнения

После выполнения лабораторной работы №16 студенты должны получить:

1. Dockerfile для контейнеризации backend-приложения

2. Dockerfile для контейнеризации frontend-приложения
3. docker-compose.yml для оркестрации всех сервисов
4. Работающее приложение в контейнерах Docker
5. Документацию по запуску и обслуживанию

Лабораторная работа №17

CI/CD пайплайн (GitHub Actions / GitLab CI)

Цель работы: Автоматизация процессов сборки, тестирования и деплоя приложения через CI/CD пайплайн с использованием GitHub Actions или GitLab CI.

Задачи

№	Задача
1	Настроить CI/CD пайплайн для автоматического тестирования при пуше в репозиторий
2	Добавить этапы линтинга и статического анализа кода
3	Настроить автоматическую сборку Docker-образов
4	Настроить автоматический деплой на тестовый сервер
5	Настроить уведомления о статусе сборки

Теоретическая часть

CI/CD (Continuous Integration / Continuous Delivery)

Понятие	**Описание**
Continuous Integration (CI)	Автоматическая сборка и тестирование кода при каждом изменении

Понятие	Описание
Continuous Delivery (CD)	Автоматическое развертывание протестированного кода
Continuous Deployment	Автоматическое развертывание в production

GitHub Actions

GitHub Actions - платформа для автоматизации CI/CD внутри GitHub.

Ключевые понятия:

Понятие	Описание
Workflow	Автоматизированный процесс (один или несколько jobs)
Job	Набор шагов, выполняющихся на одном runner
Step	Отдельная задача (команда или action)
Runner	Сервер, выполняющий workflow
Action	Переиспользуемый блок кода

Структура workflow файла

```yaml
name: CI/CD Pipeline

on:
  push:
    branches: [main, develop]
  pull_request:
    branches: [main]

env:
```

```
  REGISTRY: ghcr.io
  IMAGE_NAME: ${{ github.repository }}

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Run tests
        run: make test

  build:
    runs-on: ubuntu-latest
    needs: test
    steps:
      - name: Build Docker image
        run: docker build -t myapp .
```

Ход выполнения работы

Траектория A: Django + React

Шаг 1. Создание структуры пайплайна

.github/workflows/ci-cd.yml:

```
name: Smart Diary CI/CD Pipeline

# ============================================
# Триггеры запуска
# ============================================
on:
  push:
    branches:
      - main
      - develop
      - feature/*
    tags:
      - 'v*'
```

```yaml
  pull_request:
    branches:
      - main
      - develop

  # Ручной запуск
  workflow_dispatch:

# ============================================
# Переменные окружения
# ============================================
env:
  REGISTRY: ghcr.io
  BACKEND_IMAGE_NAME: ${{ github.repository }}/backend
  FRONTEND_IMAGE_NAME: ${{ github.repository }}/frontend
  PYTHON_VERSION: '3.11'
  NODE_VERSION: '18'

# ============================================
# Jobs
# ============================================
jobs:

  # ========================================
  # Job 1: Линтинг (Frontend)
  # ========================================
  lint-frontend:
    name: Lint Frontend
    runs-on: ubuntu-latest
    defaults:
      run:
        working-directory: ./frontend

    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: ${{ env.NODE_VERSION }}
          cache: 'npm'
          cache-dependency-path: './frontend/package-lock.json'

      - name: Install dependencies
        run: npm ci

      - name: Run ESLint
```

```
      run: npm run lint

    - name: Run Prettier check
      run: npm run prettier:check

# ========================================
# Job 2: Линтинг (Backend)
# ========================================
lint-backend:
  name: Lint Backend
  runs-on: ubuntu-latest
  defaults:
    run:
      working-directory: ./backend

  steps:
    - name: Checkout code
      uses: actions/checkout@v4

    - name: Setup Python
      uses: actions/setup-python@v5
      with:
        python-version: ${{ env.PYTHON_VERSION }}
        cache: 'pip'
        cache-dependency-path: './backend/requirements-dev.txt'

    - name: Install dependencies
      run: |
        pip install -r requirements.txt
        pip install flake8 black isort mypy

    - name: Run Flake8
      run: flake8 . --max-line-length=120 --exclude=migrations

    - name: Run Black check
      run: black --check .

    - name: Run isort check
      run: isort --check-only .

    - name: Run mypy
      run: mypy . --ignore-missing-imports

# ========================================
# Job 3: Тестирование (Backend)
# ========================================
test-backend:
  name: Test Backend
```

```yaml
    runs-on: ubuntu-latest
    needs: [lint-frontend, lint-backend]
    services:
      postgres:
        image: postgres:15-alpine
        env:
          POSTGRES_DB: test_db
          POSTGRES_USER: test_user
          POSTGRES_PASSWORD: test_password
        ports:
          - 5432:5432
        options: >-
          --health-cmd pg_isready
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5

      redis:
        image: redis:7-alpine
        ports:
          - 6379:6379
        options: >-
          --health-cmd "redis-cli ping"
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5

    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Setup Python
        uses: actions/setup-python@v5
        with:
          python-version: ${{ env.PYTHON_VERSION }}
          cache: 'pip'
          cache-dependency-path: './backend/requirements-dev.txt'

      - name: Install dependencies
        run: |
          cd backend
          pip install -r requirements-dev.txt

      - name: Run migrations
        run: |
          cd backend
          python manage.py migrate --settings=smart_diary.settings.test
```

```yaml
    - name: Run tests with coverage
      run: |
        cd backend
        pytest --cov=. --cov-report=xml --cov-report=html

    - name: Upload coverage report
      uses: codecov/codecov-action@v4
      with:
        file: ./backend/coverage.xml
        fail_ci_if_error: true

# ========================================
# Job 4: Тестирование (Frontend)
# ========================================
test-frontend:
  name: Test Frontend
  runs-on: ubuntu-latest
  defaults:
    run:
      working-directory: ./frontend

  steps:
    - name: Checkout code
      uses: actions/checkout@v4

    - name: Setup Node.js
      uses: actions/setup-node@v4
      with:
        node-version: ${{ env.NODE_VERSION }}
        cache: 'npm'
        cache-dependency-path: './frontend/package-lock.json'

    - name: Install dependencies
      run: npm ci

    - name: Run unit tests
      run: npm run test:ci

    - name: Run E2E tests
      uses: cypress-io/github-action@v6
      with:
        working-directory: ./frontend
        start: npm run dev
        wait-on: 'http://localhost:5173'
        wait-on-timeout: 120

# ========================================
# Job 5: Сборка Docker образов
```

```yaml
  # ========================================
  build-docker:
    name: Build Docker Images
    runs-on: ubuntu-latest
    needs: [test-backend, test-frontend]
    if: github.event_name == 'push' && (github.ref == 'refs/heads/main' ||
startsWith(github.ref, 'refs/tags/v'))

    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v3

      - name: Log in to Container Registry
        uses: docker/login-action@v3
        with:
          registry: ${{ env.REGISTRY }}
          username: ${{ github.actor }}
          password: ${{ secrets.GITHUB_TOKEN }}

      - name: Extract metadata for Backend
        id: meta-backend
        uses: docker/metadata-action@v5
        with:
          images: ${{ env.REGISTRY }}/${{ env.BACKEND_IMAGE_NAME }}
          tags: |
            type=ref,event=branch
            type=ref,event=pr
            type=semver,pattern={{version}}
            type=sha,format=short
          flavor: |
            latest=true

      - name: Build and push Backend
        uses: docker/build-push-action@v5
        with:
          context: ./backend
          file: ./backend/Dockerfile
          push: true
          tags: ${{ steps.meta-backend.outputs.tags }}
          labels: ${{ steps.meta-backend.outputs.labels }}
          cache-from: type=gha
          cache-to: type=gha,mode=max

      - name: Extract metadata for Frontend
        id: meta-frontend
```

```
      uses: docker/metadata-action@v5
      with:
        images: ${{ env.REGISTRY }}/${{ env.FRONTEND_IMAGE_NAME }}
        tags: |
          type=ref,event=branch
          type=ref,event=pr
          type=semver,pattern={{version}}
          type=sha,format=short
        flavor: |
          latest=true

    - name: Build and push Frontend
      uses: docker/build-push-action@v5
      with:
        context: ./frontend
        file: ./frontend/Dockerfile
        push: true
        tags: ${{ steps.meta-frontend.outputs.tags }}
        labels: ${{ steps.meta-frontend.outputs.labels }}
        cache-from: type=gha
        cache-to: type=gha,mode=max

  # ==========================================
  # Job 6: Деплой на staging (Develop ветка)
  # ==========================================
  deploy-staging:
    name: Deploy to Staging
    runs-on: ubuntu-latest
    needs: [build-docker]
    if: github.ref == 'refs/heads/develop'
    environment:
      name: staging
      url: https://staging.smartdiary.com

    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Install SSH key
        uses: webfactory/ssh-agent@v0.9.0
        with:
          ssh-private-key: ${{ secrets.STAGING_SSH_KEY }}

      - name: Deploy to staging server
        run: |
          ssh -o StrictHostKeyChecking=no
${{ secrets.STAGING_USER }}@${{ secrets.STAGING_HOST }} << 'EOF'
            cd /opt/smartdiary
```

```
            docker-compose pull
            docker-compose up -d --force-recreate
            docker system prune -f
          EOF

      - name: Run database migrations
        run: |
          ssh -o StrictHostKeyChecking=no
${{ secrets.STAGING_USER }}@${{ secrets.STAGING_HOST }} << 'EOF'
            cd /opt/smartdiary
            docker-compose exec backend python manage.py migrate
            docker-compose exec backend python manage.py collectstatic --noinput
          EOF

      - name: Health check
        run: |
          curl --fail --retry 5 --retry-delay 10 https://staging.smartdiary.com/api/health/

  # ========================================
  # Job 7: Деплой на production (Main ветка)
  # ========================================
  deploy-production:
    name: Deploy to Production
    runs-on: ubuntu latest
    needs: [build docker]
    if: github.ref == 'refs/heads/main'
    environment:
      name: production
      url: https //smartdiary.com

    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Install SSH key
        uses: webfactory/ssh agent@v0.9.0
        with:
          ssh-private-key: ${{ secrets.PRODUCTION_SSH_KEY }}

      - name: Deploy to production server
        run: |
          ssh -o StrictHostKeyChecking=no
${{ secrets.PRODUCTION_USER }}@${{ secrets.PRODUCTION_HOST }} << 'EOF'
            cd /opt/smartdiary
            docker-compose pull
            docker-compose up -d --force-recreate
            docker system prune -f
          EOF
```

```
      - name: Run database migrations
        run: |
          ssh -o StrictHostKeyChecking=no
${{ secrets.PRODUCTION_USER }}@${{ secrets.PRODUCTION_HOST }} << 'EOF'
            cd /opt/smartdiary
            docker-compose exec backend python manage.py migrate
            docker-compose exec backend python manage.py collectstatic --noinput
          EOF

      - name: Health check
        run: |
          curl --fail --retry 5 --retry-delay 10 https://smartdiary.com/api/health/

  # ========================================
  # Job 8: Уведомления
  # ========================================
  notify:
    name: Send Notifications
    runs-on: ubuntu-latest
    needs: [deploy-staging, deploy-production]
    if: always()

    steps:
      - name: Send Telegram notification on failure
        if: failure()
        uses: appleboy/telegram-action@master
        with:
          to: ${{ secrets.TELEGRAM_CHAT_ID }}
          token: ${{ secrets.TELEGRAM_BOT_TOKEN }}
          message: |
            ❌ Pipeline failed!
            Repository: ${{ github.repository }}
            Branch: ${{ github.ref_name }}
            Commit: ${{ github.sha }}
            Author: ${{ github.actor }}
            Run:
${{ github.server_url }}/${{ github.repository }}/actions/runs/${{ github.run_id }}

      - name: Send Telegram notification on success
        if: success()
        uses: appleboy/telegram-action@master
        with:
          to: ${{ secrets.TELEGRAM_CHAT_ID }}
          token: ${{ secrets.TELEGRAM_BOT_TOKEN }}
          message: |
            ✅ Pipeline completed successfully!
            Repository: ${{ github.repository }}
```

```
            Branch: ${{ github.ref_name }}
            Deployed to: ${{ github.ref == 'refs/heads/main' && 'Production' || github.ref
== 'refs/heads/develop' && 'Staging' || 'Not deployed' }}
            Run:
${{ github.server_url }}/${{ github.repository }}/actions/runs/${{ github.run_id }}
```

Шаг 2. Создание скриптов для package.json (Frontend)

frontend/package.json (дополнение):

```
{
 "scripts": {
   "dev": "vite",
   "build": "vite build",
   "preview": "vite preview",
   "lint": "eslint . --ext js,jsx --max-warnings 0",
   "lint:fix": "eslint . --ext js,jsx --fix",
   "prettier:check": "prettier --check \"src/**/*.{js,jsx,css}\"",
   "prettier:write": "prettier --write \"src/**/*.{js,jsx,css}\"",
   "test": "vitest",
   "test:ci": "vitest run --coverage",
   "test:e2e": "cypress run"
 }
}
```

Шаг 3. Создание файла зависимостей для разработки

backend/requirements-dev.txt:

```
# Production зависимости
-r requirements.txt

# Линтинг
flake8==7.0.0
black==24.2.0
isort==5.13.2
mypy==1.8.0

# Тестирование
pytest==8.0.0
```

pytest-django==4.8.0

pytest-cov==5.0.0

pytest-xdist==3.5.0

factory-boy==3.3.0

API тестирование

responses==0.25.0

Шаг 4. Создание тестовой конфигурации Django

smart_diary/settings/test.py:

```
from .settings import *

DEBUG = True
SECRET_KEY = 'test-secret-key-not-for-production'

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': os.environ.get('DB_NAME', 'test_db'),
        'USER': os.environ.get('DB_USER', 'test_user'),
        'PASSWORD': os.environ.get('DB_PASSWORD', 'test_password'),
        'HOST': os.environ.get('DB_HOST', 'localhost'),
        'PORT': os.environ.get('DB_PORT', '5432'),
    }
}

PASSWORD_HASHERS = [
    'django.contrib.auth.hashers.MD5PasswordHasher',
]

CELERY_TASK_ALWAYS_EAGER = True

EMAIL_BACKEND = 'django.core.mail.backends.locmem.EmailBackend'
```

Шаг 5. Создание Docker Compose для production

docker-compose.prod.yml:

```
version: '3.8'
```

```yaml
services:
  nginx:
    image: nginx alpine
    container_name: smartdiary_nginx
    restart: always
    volumes:
      - ./nginx/nginx.conf /etc/nginx/conf.d/default.conf ro
      - static_volume /app/staticfiles
      - media_volume /app/mediafiles
    ports:
      - "80:80"
      - "443:443"
    depends_on:
      - backend
      - frontend
    networks:
      - smartdiary_network

  backend:
    image: ghcr.io/username/smartdiary backend latest
    container_name: smartdiary_backend
    restart: always
    environment:
      - DEBUG=False
      - SECRET_KEY=${SECRET_KEY}
      - DB_NAME=${DB_NAME}
      - DB_USER=${DB_USER}
      - DB_PASSWORD=${DB_PASSWORD}
      - DB_HOST=db
      - DB_PORT=5432
      - REDIS_HOST=redis
      - REDIS_PORT=6379
    depends_on:
      - db
      - redis
    volumes:
      - static_volume /app/staticfiles
      - media_volume /app/mediafiles
    networks:
      - smartdiary_network

  frontend:
    image: ghcr.io/username/smartdiary frontend latest
    container_name: smartdiary_frontend
    restart: always
    networks:
      - smartdiary_network
```

```
  db:
    image: postgres 15 alpine
    container_name: smartdiary_db
    restart: always
    environment:
      - POSTGRES_DB=${DB_NAME}
      - POSTGRES_USER=${DB_USER}
      - POSTGRES_PASSWORD=${DB_PASSWORD}
    volumes:
      - postgres_data:/var/lib/postgresql/data
    networks:
      - smartdiary_network

  redis:
    image: redis 7 alpine
    container_name: smartdiary_redis
    restart: always
    command: redis server --appendonly yes
    volumes:
      - redis_data /data
    networks:
      - smartdiary_network

volumes:
  postgres_data:
  redis_data:
  static_volume:
  media_volume:

networks:
  smartdiary_network:
    driver: bridge
```

Шаг 6. Настройка GitHub Secrets

Необходимые секреты для настройки:

Secret Name	Описание
STAGING_SSH_KEY	SSH ключ для доступа к staging серверу
STAGING_HOST	IP или домен staging сервера
STAGING_USER	Имя пользователя на staging сервере

Secret Name	Описание
PRODUCTION_SSH_KEY	SSH ключ для доступа к production серверу
PRODUCTION_HOST	IP или домен production сервера
PRODUCTION_USER	Имя пользователя на production сервере
TELEGRAM_BOT_TOKEN	Токен Telegram бота для уведомлений
TELEGRAM_CHAT_ID	ID чата Telegram для уведомлений

Траектория Б: ASP.NET Core + React

CI/CD пайплайн для ASP.NET Core

.github/workflows/ci-cd-aspnet.yml:

```yaml
name: Smart Diary ASP.NET CI/CD Pipeline

on:
  push:
    branches: [main, develop]
  pull_request:
    branches: [main]
  workflow_dispatch:

env:
  DOTNET_VERSION: '8.0.x'
  NODE_VERSION: '18'

jobs:
  # ========================================
  # Линтинг и форматирование .NET
  # ========================================
  lint-dotnet:
    name: Lint .NET Code
    runs-on: ubuntu latest
    defaults:
      run:
        working-directory: ./backend

    steps:
```

```yaml
    - uses: actions/checkout@v4

    - name: Setup .NET
      uses: actions/setup dotnet@v4
      with:
        dotnet-version: ${{ env.DOTNET_VERSION }}

    - name: Restore tools
      run: dotnet tool restore

    - name: Run dotnet format
      run: dotnet format --verify no changes --severity info

  # ========================================
  # Тестирование .NET
  # ========================================
  test-dotnet:
    name: Test .NET Backend
    runs-on: ubuntu latest
    needs: lint dotnet
    services:
      postgres:
        image: postgres 15 alpine
        env:
          POSTGRES_DB: test_db
          POSTGRES_USER: test_user
          POSTGRES_PASSWORD: test_password
        ports:
          - 5432 5432

    steps:
      - uses: actions/checkout@v4

      - name: Setup .NET
        uses: actions/setup dotnet@v4
        with:
          dotnet-version: ${{ env.DOTNET_VERSION }}

      - name: Restore dependencies
        run: dotnet restore

      - name: Run tests with coverage
        run: |
          dotnet test --collect:"XPlat Code Coverage" --settings coverlet.runsettings

      - name: Upload coverage
        uses: codecov/codecov action@v4
        with:
          file: ./**/coverage.cobertura.xml

  # ========================================
  # Сборка Docker образа
  # ========================================
```

```
  build-docker:
    name: Build Docker Images
    runs-on: ubuntu latest
    needs: test dotnet
    if: github.ref == 'refs/heads/main'

    steps:
      - uses: actions/checkout@v4

      - name: Set up Docker Buildx
        uses: docker/setup buildx action@v3

      - name: Log in to GitHub Container Registry
        uses: docker/login action@v3
        with:
          registry: ghcr.io
          username: ${{ github.actor }}
          password: ${{ secrets.GITHUB_TOKEN }}

      - name: Build and push
        uses: docker/build push action@v5
        with:
          context: ./backend
          push: true
          tags:
ghcr.io/${{ github.repository }}/backend latest ghcr.io/${{ github.repository }}/backend ${{ github.sh
a }}
```

Контрольные вопросы

1. Что такое CI/CD и какие проблемы он решает?
2. В чем разница между Continuous Delivery и Continuous Deployment?
3. Какие этапы включены в пайплайн?
4. Как обрабатываются ошибки в пайплайне?
5. Как организовать безопасное хранение секретов?
6. Как настроить деплой на разные окружения (staging/production)?
7. Что такое GitHub Actions и как он работает?

Критерии оценки

Критерий	Баллы
Настроен CI пайплайн (запуск тестов при пуше)	2
Добавлены этапы линтинга (ESLint, Flake8/dotnet format)	2
Настроена автоматическая сборка Docker-образов	2
Настроен автоматический деплой на staging	2
Настроены уведомления о статусе сборки	1
Пайплайн успешно проходит	1
Итого	10

Результат выполнения

После выполнения лабораторной работы №17 студенты должны получить:

1. CI/CD пайплайн (GitHub Actions / GitLab CI) с этапами:
2. Линтинг фронтенда и бэкенда
3. Запуск тестов
4. Сборка Docker-образов
5. Деплой на staging/production окружения
6. Автоматические уведомления о статусе сборки (Telegram/Slack)
7. Документацию по CI/CD процессу

Лабораторная работа №18

Мониторинг и логирование

Цель работы: Настройка системы мониторинга (Prometheus + Grafana) и централизованного логирования (Loki + Promtail) для отслеживания состояния приложения, сбора метрик и анализа логов.

Задачи

№	Задача
1	Настроить сбор метрик приложения (Prometheus)
2	Настроить визуализацию метрик (Grafana)
3	Настроить централизованный сбор логов (Loki + Promtail)
4	Настроить алерты в Telegram/Slack
5	Создать дашборды для мониторинга ключевых метрик

Теоретическая часть

Мониторинг и логирование

Понятие	Инструмент	Описание
Метрики	Prometheus	Сбор числовых показателей (RPS, latency, ошибки)
Визуализация	Grafana	Дашборды для отображения метрик
Логи	Loki	Хранение и поиск логов
Алерты	Alertmanager	Отправка уведомлений

Prometheus - система мониторинга с открытым исходным кодом.

Ключевые понятия:

1. Metric - числовой показатель (counter, gauge, histogram)
2. Target - endpoint, с которого собираются метрики
3. Scrape - процесс сбора метрик
4. PromQL - язык запросов для метрик

Loki - горизонтально масштабируемая система агрегации логов.

Преимущества:

1. Интеграция с Prometheus
2. Эффективное хранение
3. Быстрый поиск

Grafana - платформа для визуализации данных.

Возможности:

1. Дашборды с графиками
2. Алерты
3. Интеграция с Prometheus, Loki, Elasticsearch

Ход выполнения работы

Траектория A: Django + React

Шаг 1. Установка и настройка django-prometheus

```
pip install django-prometheus
settings.py (дополнение):
python
INSTALLED_APPS = [
    'django_prometheus',
    # ... остальные приложения
]

MIDDLEWARE = [
    'django_prometheus.middleware.PrometheusBeforeMiddleware',
    # ... остальные middleware
    'django_prometheus.middleware.PrometheusAfterMiddleware',
]
```

```
# Настройка метрик
PROMETHEUS_EXPORT_MIGRATIONS = False
PROMETHEUS_LATENCY_BUCKETS = (0.01, 0.025, 0.05, 0.075, 0.1, 0.25, 0.5, 0.75, 1.0, 2.5, 5.0, 7.5, 10.0, float('inf'))

# Custom метрики
from prometheus_client import Counter, Histogram, Gauge

# Счетчик запросов к API
api_requests_total = Counter('api_requests_total', 'Total API requests', ['method', 'endpoint', 'status'])

# Гистограмма времени ответа
api_request_duration = Histogram('api_request_duration_seconds', 'API request duration', ['method', 'endpoint'])

# Датчик активных пользователей
active_users = Gauge('active_users', 'Active users count')
```

Шаг 2. Создание эндпоинта для метрик

urls.py (дополнение):

```
urlpatterns = [
    path('', include('django_prometheus.urls')),
    # ... остальные маршруты
]
```

Шаг 3. Добавление кастомных метрик

tasks/metrics.py:

```
from prometheus_client import Counter, Histogram, Gauge, Info
from django.utils import timezone

# Счетчики задач
tasks_created_total = Counter('tasks_created_total', 'Total tasks created')
tasks_completed_total = Counter('tasks_completed_total', 'Total tasks completed')
tasks_deleted_total = Counter('tasks_deleted_total', 'Total tasks deleted')

# Гистограмма времени выполнения задачи
task_execution_time = Histogram('task_execution_time_seconds', 'Task execution time')
```

```
# Датчик количества активных задач
active_tasks = Gauge('active_tasks', 'Number of active tasks (not completed)')

# Информация о версии приложения
app_info = Info('smartdiary_app', 'Application information')
app_info.info({'version': '1.0.0', 'environment': 'production'})

# Функции для обновления метрик
def track_task_created():
    tasks_created_total.inc()
    update_active_tasks_gauge()

def track_task_completed():
    tasks_completed_total.inc()
    update_active_tasks_gauge()

def track_task_deleted():
    tasks_deleted_total.inc()
    update_active_tasks_gauge()

def update_active_tasks_gauge():
    from .models import Task
    active_tasks.set(Task.objects.exclude(status='Completed').count())
```

Шаг 4. Создание docker-compose для мониторинга

docker-compose.monitoring.yml:

```
version: '3.8'


services:
  # ==========================================
  # Prometheus
  # ==========================================
  prometheus:
    image: prom/prometheus:latest
    container_name: smartdiary_prometheus
    restart: unless-stopped
    volumes:
```

```
      - ./monitoring/prometheus/prometheus.yml:/etc/prometheus/prometheus.yml:ro
      - ./monitoring/prometheus/alert_rules.yml:/etc/prometheus/alert_rules.yml:ro
      - prometheus_data:/prometheus
    command:
      - '--config.file=/etc/prometheus/prometheus.yml'
      - '--storage.tsdb.path=/prometheus'
      - '--web.console.libraries=/usr/share/prometheus/console_libraries'
      - '--web.console.templates=/usr/share/prometheus/consoles'
      - '--web.enable-lifecycle'
    ports:
      - "9090:9090"
    networks:
      - smartdiary_network

  # ============================================
  # Grafana
  # ============================================
  grafana:
    image: grafana/grafana:latest
    container_name: smartdiary_grafana
    restart: unless-stopped
    volumes:
      - ./monitoring/grafana/dashboards:/etc/grafana/provisioning/dashboards:ro
      - ./monitoring/grafana/datasources:/etc/grafana/provisioning/datasources:ro
      - grafana_data:/var/lib/grafana
    environment:
      - GF_SECURITY_ADMIN_PASSWORD=${GRAFANA_PASSWORD:-admin}
      - GF_INSTALL_PLUGINS=grafana-piechart-panel
    ports:
      - "3000:3000"
    depends_on:
```

```
    - prometheus
    - loki
  networks:
    - smartdiary_network

# ==========================================
# Loki
# ==========================================
loki:
  image: grafana/loki:latest
  container_name: smartdiary_loki
  restart: unless-stopped
  volumes:
    - ./monitoring/loki/loki-config.yml:/etc/loki/loki-config.yml:ro
    - loki_data:/loki
  command: -config.file=/etc/loki/loki-config.yml
  ports:
    - "3100:3100"
  networks:
    - smartdiary_network

# ==========================================
# Promtail
# ==========================================
promtail:
  image: grafana/promtail:latest
  container_name: smartdiary_promtail
  restart: unless-stopped
  volumes:
    - ./monitoring/promtail/promtail-config.yml:/etc/promtail/promtail-config.yml:ro
    - /var/lib/docker/containers:/var/lib/docker/containers:ro
    - /var/run/docker.sock:/var/run/docker.sock
```

```
      - ./backend/logs:/var/log/app:ro
    command: -config.file=/etc/promtail/promtail-config.yml
    depends_on:
      - loki
    networks:
      - smartdiary_network

  # ============================================
  # Alertmanager
  # ============================================
  alertmanager:
    image: prom/alertmanager:latest
    container_name: smartdiary_alertmanager
    restart: unless-stopped
    volumes:
      - ./monitoring/alertmanager/alertmanager.yml:/etc/alertmanager/alertmanager.yml:ro
      - alertmanager_data:/alertmanager
    command:
      - '--config.file=/etc/alertmanager/alertmanager.yml'
      - '--storage.path=/alertmanager'
    ports:
      - "9093:9093"
    networks:
      - smartdiary_network

  # ============================================
  # Node Exporter (метрики сервера)
  # ============================================
  node-exporter:
    image: prom/node-exporter:latest
    container_name: smartdiary_node_exporter
```

```
    restart: unless stopped
    volumes:
      - /proc /host/proc ro
      - /sys /host/sys ro
      - / /rootfs ro
    command:
      - '--path.procfs=/host/proc'
      - '--path.sysfs=/host/sys'
      - '--path.rootfs=/rootfs'
      - '--collector.filesystem.mount-points-exclude=^/(sys|proc|dev|host|etc)($$|/)'
    ports:
      - "9100:9100"
    networks:
      - smartdiary_network

  # ============================================
  # cAdvisor (метрики Docker контейнеров)
  # ============================================
  cadvisor:
    image: gcr.io/cadvisor/cadvisor:latest
    container_name: smartdiary_cadvisor
    restart: unless stopped
    volumes:
      - / /rootfs ro
      - /var/run /var/run ro
      - /sys /sys ro
      - /var/lib/docker/ /var/lib/docker ro
      - /dev/disk/ /dev/disk ro
    ports:
      - "8080:8080"
    networks:
      - smartdiary_network
```

```
networks:
  smartdiary_network:
    external: true

volumes:
  prometheus_data:
  grafana_data:
  loki_data:
  alertmanager_data
```

Шаг 5. Конфигурация Prometheus

monitoring/prometheus/prometheus.yml:

```
global:
  scrape_interval: 15s
  evaluation_interval: 15s
  external_labels:
    monitor: 'smartdiary'

# Правила алертов
rule_files:
  - "alert_rules.yml"

# Настройка уведомлений
alerting:
  alertmanagers:
    - static_configs:
        - targets:
            - 'alertmanager:9093'

# Targets для сбора метрик
scrape_configs:
  # Django приложение
  - job_name: 'django'
    static_configs:
      - targets: ['backend:8000']
    metrics_path: '/metrics'
    scrape_interval: 10s

  # PostgreSQL
```

```
  - job_name: 'postgresql'
    static_configs:
      - targets: ['db:9187']
    scrape_interval: 15s

  # Redis
  - job_name: 'redis'
    static_configs:
      - targets: ['redis:9121']
    scrape_interval: 15s

  # Node Exporter (системные метрики)
  - job_name: 'node-exporter'
    static_configs:
      - targets: ['node-exporter:9100']
    scrape_interval: 15s

  # cAdvisor (Docker контейнеры)
  - job_name: 'cadvisor'
    static_configs:
      - targets: ['cadvisor:8080']
    scrape_interval: 15s

  # Nginx
  - job_name: 'nginx'
    static_configs:
      - targets: ['nginx:9113']
    scrape_interval: 15s
```

Шаг 6. Правила алертов

monitoring/prometheus/alert_rules.yml:

```
groups:
  - name: smartdiary_alerts
    interval: 30s
    rules:
      # Высокая задержка API
      - alert: HighAPILatency
        expr: histogram_quantile(0.95
sum(rate(api_request_duration_seconds_bucket[5m])) by (le, endpoint)) > 1
        for: 5m
        labels:
          severity: warning
        annotations:
          summary: "High API latency"
```

```
      description: "API endpoint {{ $labels.endpoint }} has high latency
({{ $value }}s)"

    # Частота ошибок 5xx
    - alert: HighErrorRate
      expr: sum(rate(api_requests_total{status=~"5.."}[5m])) /
sum(rate(api_requests_total[5m])) > 0.05
      for: 5m
      labels:
        severity: critical
      annotations:
        summary: "High error rate"
        description: "Error rate is {{ $value | humanizePercentage }}"

    # Сервер недоступен
    - alert: InstanceDown
      expr: up == 0
      for: 1m
      labels:
        severity: critical
      annotations:
        summary: "Instance {{ $labels.instance }} down"
        description: "{{ $labels.instance }} of job {{ $labels.job }} has been down for
more than 1 minute."

    # Высокое использование CPU
    - alert: HighCPUUsage
      expr: 100 - (avg(rate(node_cpu_seconds_total{mode="idle"}[5m])) * 100) > 80
      for: 10m
      labels:
        severity: warning
      annotations:
        summary: "High CPU usage"
        description: "CPU usage is {{ $value }}%"

    # Низкий свободный объем диска
    - alert: LowDiskSpace
      expr: (node_filesystem_avail_bytes{mountpoint="/"} /
node_filesystem_size_bytes{mountpoint="/"}) * 100 < 10
      for: 5m
      labels:
        severity: critical
      annotations:
        summary: "Low disk space"
        description: "Only {{ $value }}% disk space remaining"

    # Много неудачных логинов
    - alert: ManyFailedLogins
```

```
      expr: sum(increase(django_login_attempts_total{success="false"}[10m])) > 10
      labels:
        severity: warning
      annotations:
        summary: "Many failed login attempts"
        description: "{{ $value }} failed login attempts in last 10 minutes"

    # База данных недоступна
    - alert: DatabaseDown
      expr: pg_up == 0
      for: 30s
      labels:
        severity: critical
      annotations:
        summary: "Database is down"
        description: "PostgreSQL database is not responding"
```

Шаг 7. Конфигурация Loki

monitoring/loki/loki-config.yml:

```
auth_enabled: false

server:
  http_listen_port: 3100

common:
  path_prefix: /loki
  storage:
    filesystem:
      chunks_directory: /loki/chunks
      rules_directory: /loki/rules
  replication_factor: 1
  ring:
    instance_addr: 127.0.0.1
    kvstore:
      store: inmemory

schema_config:
  configs:
    - from: 2020-10-24
      store: boltdb-shipper
      object_store: filesystem
      schema: v11
      index:
        prefix: index_
        period: 24h
```

```
limits_config:
  retention_period: 168h  # 7 дней

ingester:
  lifecycler:
    ring:
      kvstore:
        store: inmemory
      replication_factor: 1
    final_sleep: 0s
  chunk_idle_period: 5m
  chunk_retain_period: 30s

querier:
  query_timeout: 300s

frontend:
  log_queries_longer_than: 5s
  compress_responses: true
```

Шаг 8. Конфигурация Promtail

monitoring/promtail/promtail-config.yml:

```
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:
  # Логи Docker контейнеров
  - 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_service']
```

```
      target_label: 'service'

  # Логи приложения
  - job_name: 'django_app'
    static_configs:
      - targets: ['localhost']
        labels:
          job: 'django'
          service: 'backend'
          __path__: '/var/log/app/*.log'

  # Логи Nginx
  - job_name: 'nginx'
    static_configs:
      - targets: ['localhost']
        labels:
          job: 'nginx'
          service: 'nginx'
          __path__: '/var/log/nginx/*.log'
    pipeline_stages:
      - regex:
          expression: '^(?P<remote_addr>\S+) - (?P<remote_user>\S+)
\[(?P<time_local>[^\]]+)\] "(?P<method>\S+) (?P<request_uri>\S+) (?P<protocol>\S+)"
(?P<status>\d{3}) (?P<body_bytes_sent>\d+) "(?P<http_referer>[^"]*)"
"(?P<http_user_agent>[^"]*)"'
      - labels:
          method: ''
          status: ''
          request_uri: ''
```

Шаг 9. Конфигурация Alertmanager

monitoring/alertmanager/alertmanager.yml:

```
global:
  slack_api_url: 'https://hooks.slack.com/services/YOUR/WEBHOOK/URL'
  telegram_api_url: 'https://api.telegram.org/bot{TOKEN}/sendMessage'

route:
  group_by: ['alertname', 'cluster', 'service']
  group_wait: 10s
  group_interval: 10s
  repeat_interval: 12h
  receiver: 'default-receiver'
  routes:
    - match:
        severity: critical
```

```
    receiver: critical receiver
    continue: true
  - match:
      severity: warning
    receiver: warning receiver

receivers:
  - name: 'default-receiver'
    telegram_configs:
      - bot_token: '${TELEGRAM_BOT_TOKEN}'
        chat_id: $(TELEGRAM_CHAT_ID)
        parse_mode: 'HTML'
        message: |
          <b>[{{ .Status | toUpper }}]</b> {{ .CommonAnnotations.summary }}
          {{ range .Alerts }}
          {{ .Annotations.description }}
          {{ end }}

  - name: 'critical-receiver'
    slack_configs:
      - channel: '#alerts-critical'
        title: '🚨 Critical Alert'
        text: |
          *Alert:* {{ .CommonAnnotations.summary }}
          *Description:* {{ .CommonAnnotations.description }}
          *Severity:* critical
          *Time:* {{ .StartsAt }}
    telegram_configs:
      - bot_token: '${TELEGRAM_BOT_TOKEN}'
        chat_id: $(TELEGRAM_CHAT_ID)
        parse_mode: 'HTML'
        message: |
          🚨 <b>CRITICAL ALERT</b>
          {{ .CommonAnnotations.summary }}
          {{ .CommonAnnotations.description }}

  - name: 'warning-receiver'
    slack_configs:
      - channel: '#alerts-warning'
        title: '⚠ Warning Alert'
        text: |
          *Alert:* {{ .CommonAnnotations.summary }}
          *Description:* {{ .CommonAnnotations.description }}

inhibit_rules:
  - source_match:
      severity: 'critical'
    target_match:
```

```
    severity: 'warning'
  equal: ['alertname', 'cluster', 'service']
```

Шаг 10. Настройка логирования в Django

settings.py (дополнение):

```
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'formatters': {
        'verbose': {
            'format': '{levelname} {asctime} {module} {process:d} {thread:d} {message}',
            'style': '{',
        },
        'json': {
            'format': '{"time": "%(asctime)s", "level": "%(levelname)s", "module": "%(module)s", "message": "%(message)s"}',
            'style': '%',
        },
    },
    'handlers': {
        'console': {
            'class': 'logging.StreamHandler',
            'formatter': 'json',
        },
        'file': {
            'class': 'logging.handlers.RotatingFileHandler',
            'filename': '/var/log/app/smartdiary.log',
            'maxBytes': 10485760,  # 10 MB
            'backupCount': 10,
            'formatter': 'json',
        },
        'prometheus': {
            'class': 'django_prometheus.logging.PrometheusLogHandler',
        },
    },
    'root': {
        'handlers': ['console', 'file', 'prometheus'],
        'level': 'INFO',
    },
    'loggers': {
        'django': {
            'handlers': ['console', 'file'],
            'level': 'INFO',
            'propagate': False,
        },
```

```
        'django.request': {
            'handlers': ['console', 'file'],
            'level': 'WARNING',
            'propagate': False,
        },
        'tasks': {
            'handlers': ['console', 'file', 'prometheus'],
            'level': 'DEBUG',
            'propagate': False,
        },
    },
}
```

Шаг 11. Конфигурация Grafana

monitoring/grafana/datasources/prometheus.yml:

```
apiVersion: 1

datasources:
  - name: Prometheus
    type: prometheus
    access: proxy
    url: http //prometheus:9090
    isDefault: true
    editable: true

  - name: Loki
    type: loki
    access: proxy
    url: http //loki:3100
    editable: true
```

Шаг 12. Дашборд Grafana (JSON)

monitoring/grafana/dashboards/smartdiary-dashboard.json:

```
{
  "title": "Smart Diary Monitoring",
  "uid": "smartdiary-dashboard",
  "panels": [
    {
      "title": "API Requests Rate",
      "type": "graph",
      "targets": [
        {
          "expr": "rate(api_requests_total[5m])",
```

```
        "legendFormat": "{{method}} {{endpoint}}"
      }
    ],
    "gridPos": {"x": 0, "y": 0, "w": 12, "h": 8}
  },
  {
    "title": "API Response Time (p95)",
    "type": "graph",
    "targets": [
      {
        "expr": "histogram_quantile(0.95,
sum(rate(api_request_duration_seconds_bucket[5m])) by (le, endpoint))",
        "legendFormat": "{{endpoint}}"
      }
    ],
    "gridPos": {"x": 12, "y": 0, "w": 12, "h": 8}
  },
  {
    "title": "Error Rate",
    "type": "graph",
    "targets": [
      {
        "expr": "sum(rate(api_requests_total{status=~\"5..\"}[5m])) /
sum(rate(api_requests_total[5m]))",
        "legendFormat": "Error rate"
      }
    ],
    "gridPos": {"x": 0, "y": 8, "w": 8, "h": 6}
  },
  {
    "title": "Active Users",
    "type": "stat",
    "targets": [
      {
        "expr": "active_users"
      }
    ],
    "gridPos": {"x": 8, "y": 8, "w": 4, "h": 6}
  },
  {
    "title": "Active Tasks",
    "type": "stat",
    "targets": [
      {
        "expr": "active_tasks"
      }
    ],
    "gridPos": {"x": 12, "y": 8, "w": 4, "h": 6}
```

```
    },
    {
      "title": "CPU Usage",
      "type": "graph",
      "targets": [
        {
          "expr": "100 - (avg(rate(node_cpu_seconds_total{mode=\"idle\"}[5m])) * 100)",
          "legendFormat": "CPU"
        }
      ],
      "gridPos": {"x": 0, "y": 14, "w": 8, "h": 6}
    },
    {
      "title": "Memory Usage",
      "type": "graph",
      "targets": [
        {
          "expr": "(1 - (node_memory_MemAvailable_bytes /
node_memory_MemTotal_bytes)) * 100",
          "legendFormat": "Memory"
        }
      ],
      "gridPos": {"x": 8, "y": 14, "w": 8, "h": 6}
    },
    {
      "title": "Tasks Created per Day",
      "type": "bargauge",
      "targets": [
        {
          "expr": "sum(increase(tasks_created_total[24h]))"
        }
      ],
      "gridPos": {"x": 0, "y": 20, "w": 12, "h": 8}
    }
  ]
}
```

Шаг 13. Команды для запуска

bash

```
# Запуск всех сервисов мониторинга
docker-compose -f docker-compose.monitoring.yml up -d

# Просмотр логов
docker-compose -f docker-compose.monitoring.yml logs -f
```

```
# Остановка
docker-compose -f docker-compose.monitoring.yml down

# Полная очистка
docker-compose -f docker-compose.monitoring.yml down -v
```

Контрольные вопросы

1. Какие метрики необходимо мониторить в веб-приложении?
2. Чем отличается Prometheus от Loki?
3. Как настроить алерты в Alertmanager?
4. Какие дашборды нужно создать в Grafana?
5. Как централизовать сбор логов с нескольких сервисов?
6. Как интегрировать уведомления в Telegram/Slack?

Критерии оценки

Критерий	Баллы
Настроен Prometheus для сбора метрик	2
Настроена Grafana с дашбордами	2
Настроен Loki + Promtail для логов	2
Настроены алерты в Alertmanager	2
Настроены уведомления в Telegram/Slack	1
Созданы дашборды для ключевых метрик	1
Итого	10

Результат выполнения

После выполнения лабораторной работы №18 студенты должны получить:

1. Систему мониторинга (Prometheus) для сбора метрик
2. Визуализацию (Grafana) с дашбордами
3. Централизованное логирование (Loki)
4. Систему алертов с уведомлениями в Telegram/Slack
5. Дашборды для отслеживания ключевых метрик

Лабораторная работа №19

PWA (Progressive Web App) и офлайн-режим

Цель работы: Превращение веб-приложения в Progressive Web App (PWA) с возможностью работы офлайн, получения push-уведомлений и установки на устройство пользователя.

Задачи

№	Задача
1	Создать манифест (manifest.json) для PWA
2	Настроить Service Worker для кэширования статических ресурсов
3	Реализовать офлайн-режим с кэшированием данных в IndexedDB
4	Добавить возможность "установки" приложения на устройство
5	Настроить push-уведомления через сервис-воркер

Теоретическая часть

PWA =веб-приложение, которое ведёт себя как нативное: устанавливается на устройство, работает офлайн, получает push-уведомления.

Три ключевых компонента PWA:

Компонент	**Описание**
HTTPS	Обязательное требование для PWA
Web App Manifest	JSON-файл с метаданными приложения
Service Worker	JavaScript-скрипт, работающий в фоне

Service Worker - это программный прокси-сервер между браузером и сетью.

Возможности:

1. Перехват сетевых запросов
2. Кэширование ресурсов
3. Push-уведомления
4. Фоновая синхронизация

Жизненный цикл Service Worker:

Register → Install → Activate → Fetch/Message → Update → SkipWaiting

Web App Manifest

Manifest - JSON-файл, определяющий, как приложение выглядит при установке.

Основные поля:

1. name - полное имя приложения
2. short_name - короткое имя для рабочего стола
3. start_url - стартовая страница
4. display - режим отображения (standalone, fullscreen, minimal-ui)
5. icons - иконки разных размеров

6. theme_color - цвет темы
7. background_color - цвет фона при загрузке

Кэш-стратегии

Стратегия	Описание	Применение
Cache First	Сначала кэш, затем сеть	Статические ресурсы
Network First	Сначала сеть, затем кэш	API запросы
Stale While Revalidate	Кэш + фоновая отправка запроса	Аватары, картинки
Network Only	Только сеть	Формы, платежи
Cache Only	Только кэш	Офлайн-контент

Ход выполнения работы

Шаг 1. Создание манифеста PWA

public/manifest.json:

```
{
  "name": "Smart Diary - Умный ежедневник",
  "short_name": "Smart Diary",
  "description": "Приложение для управления задачами, проектами и личной продуктивностью",
  "start_url": "/",
  "display": "standalone",
  "theme_color": "#2c3e50",
  "background_color": "#ffffff",
  "orientation": "portrait-primary",
  "scope": "/",
  "categories": ["productivity", "lifestyle", "utilities"],
  "lang": "ru-RU",
  "dir": "ltr",
  "icons": [
    {
      "src": "/icons/icon-72.png",
      "sizes": "72x72",
```

```
    "type": "image/png",
    "purpose": "any"
  },
  {
    "src": "/icons/icon-96.png",
    "sizes": "96x96",
    "type": "image/png",
    "purpose": "any"
  },
  {
    "src": "/icons/icon-128.png",
    "sizes": "128x128",
    "type": "image/png",
    "purpose": "any"
  },
  {
    "src": "/icons/icon-144.png",
    "sizes": "144x144",
    "type": "image/png",
    "purpose": "any"
  },
  {
    "src": "/icons/icon-152.png",
    "sizes": "152x152",
    "type": "image/png",
    "purpose": "any"
  },
  {
    "src": "/icons/icon-192.png",
    "sizes": "192x192",
    "type": "image/png",
    "purpose": "any"
  },
  {
    "src": "/icons/icon-384.png",
    "sizes": "384x384",
    "type": "image/png",
    "purpose": "any"
  },
  {
    "src": "/icons/icon-512.png",
    "sizes": "512x512",
    "type": "image/png",
    "purpose": "any"
  },
  {
    "src": "/icons/maskable-icon-192.png",
    "sizes": "192x192",
```

```
      "type": "image/png",
      "purpose": "maskable"
    },
    {
      "src": "/icons/maskable-icon-512.png",
      "sizes": "512x512",
      "type": "image/png",
      "purpose": "maskable"
    }
  ],
  "screenshots": [
    {
      "src": "/screenshots/desktop.png",
      "sizes": "1280x720",
      "type": "image/png",
      "form_factor": "wide",
      "label": "Главный экран"
    },
    {
      "src": "/screenshots/mobile.png",
      "sizes": "750x1334",
      "type": "image/png",
      "form_factor": "narrow",
      "label": "Мобильный вид"
    }
  ],
  "shortcuts": [
    {
      "name": "Новая задача",
      "short_name": "Новая задача",
      "description": "Быстрое создание задачи",
      "url": "/tasks/create",
      "icons": [
        {
          "src": "/icons/task-icon.png",
          "sizes": "96x96"
        }
      ]
    },
    {
      "name": "Календарь",
      "short_name": "Календарь",
      "url": "/calendar",
      "icons": [
        {
          "src": "/icons/calendar-icon.png",
          "sizes": "96x96"
        }
```

```
    ]
   }
  ],
  "related_applications": [],
  "prefer_related_applications": false
}
```

Шаг 2. Подключение манифеста в HTML

index.html (дополнение):

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="UTF-8" />
  <link rel="icon" type="image/png" href="/icons/icon-192.png" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0,
viewport-fit=cover" />
  <meta name="theme-color" content="#2c3e50" />
  <meta name="apple-mobile-web-app-capable" content="yes" />
  <meta name="apple-mobile-web-app-status-bar-style" content="black-
translucent" />
  <meta name="apple-mobile-web-app-title" content="Smart Diary" />
  <meta name="application-name" content="Smart Diary" />
  <meta name="msapplication-TileColor" content="#2c3e50" />
  <meta name="msapplication-TileImage" content="/icons/icon-144.png" />

  <link rel="manifest" href="/manifest.json" />
  <link rel="apple-touch-icon" href="/icons/icon-192.png" />

  <title>Smart Diary - Умный ежедневник</title>
</head>
<body>
  <div id="root"></div>
  <script type="module" src="/src/main.jsx"></script>
</body>
</html>
```

Шаг 3. Создание Service Worker

public/sw.js:

```
// ============================================
// Service Worker для Smart Diary
// ============================================
```

```
// Имя кэша и версия
const CACHE_NAME = 'smartdiary-cache-v1.0.0';
const DATA_CACHE_NAME = 'smartdiary-data-v1';

// Ресурсы для кэширования при установке
const STATIC_ASSETS = [
  '/',
  '/index.html',
  '/manifest.json',
  '/icons/icon-72.png',
  '/icons/icon-96.png',
  '/icons/icon-128.png',
  '/icons/icon-144.png',
  '/icons/icon-152.png',
  '/icons/icon-192.png',
  '/icons/icon-384.png',
  '/icons/icon-512.png',
  '/icons/maskable-icon-192.png',
  '/icons/maskable-icon-512.png',
  '/offline.html'
];

// API эндпоинты для кэширования
const API_ENDPOINTS = [
  '/api/tasks/',
  '/api/projects/',
  '/api/tags/'
];

// ============================================
// Установка Service Worker
// ============================================
self.addEventListener('install', (event) => {
  console.log('[SW] Installing...');

  event.waitUntil(
    caches.open(CACHE_NAME)
      .then((cache) => {
        console.log('[SW] Caching static assets');
        return cache.addAll(STATIC_ASSETS);
      })
      .then(() => {
        console.log('[SW] Installation complete');
        return self.skipWaiting();
      })
  );
});
```

```
// ============================================
// Активация Service Worker
// ============================================
self.addEventListener('activate', (event) => {
  console.log('[SW] Activating...');

  event.waitUntil(
    caches.keys().then((cacheNames) => {
      return Promise.all(
        cacheNames.map((cacheName) => {
          if (cacheName !== CACHE_NAME && cacheName !== DATA_CACHE_NAME) {
            console.log('[SW] Deleting old cache:', cacheName);
            return caches.delete(cacheName);
          }
        })
      );
    }).then(() => {
      console.log('[SW] Activation complete');
      return self.clients.claim();
    })
  );
});

// ============================================
// Стратегии кэширования
// ============================================

// Cache First с обновлением в фоне
const cacheFirstWithRefresh = async (request) => {
  const cachedResponse = await caches.match(request);

  // Отправляем запрос на сервер в фоне
  const fetchPromise = fetch(request).then(async (networkResponse) => {
    const cache = await caches.open(DATA_CACHE_NAME);
    cache.put(request, networkResponse.clone());
    return networkResponse;
  });

  return cachedResponse || fetchPromise;
};

// Network First с падением на кэш
const networkFirstWithCacheFallback = async (request) => {
  try {
    const networkResponse = await fetch(request);
    const cache = await caches.open(DATA_CACHE_NAME);
    cache.put(request, networkResponse.clone());
```

```
    return networkResponse;
  } catch (error) {
    const cachedResponse = await caches.match(request);
    if (cachedResponse) {
      return cachedResponse;
    }
    // Если нет кэша и нет сети - возвращаем страницу офлайн
    if (request.headers.get('Accept').includes('text/html')) {
      return caches.match('/offline.html');
    }
    throw error;
  }
};

// Stale While Revalidate
const staleWhileRevalidate = async (request) => {
  const cache = await caches.open(DATA_CACHE_NAME);
  const cachedResponse = await cache.match(request);

  const fetchPromise = fetch(request).then(async (networkResponse) => {
    cache.put(request, networkResponse.clone());
    return networkResponse;
  });

  return cachedResponse || fetchPromise;
};

// ============================================
# Обработка запросов
# ============================================
self.addEventListener('fetch', (event) => {
  const url = new URL(event.request.url);

  // API запросы (Network First)
  if (url.pathname.startsWith('/api/')) {
    event.respondWith(networkFirstWithCacheFallback(event.request));
    return;
  }

  // Статические ресурсы (Cache First)
  if (STATIC_ASSETS.some(asset => url.pathname.includes(asset))) {
    event.respondWith(cacheFirstWithRefresh(event.request));
    return;
  }

  // Изображения (Stale While Revalidate)
  if (url.pathname.match(/\.(png|jpg|jpeg|svg|gif|ico)$/)) {
    event.respondWith(staleWhileRevalidate(event.request));
```

```
    return;
  }

  // Остальные запросы (Network First с падением на кэш)
  event.respondWith(networkFirstWithCacheFallback(event.request));
});

// ============================================
# Push-уведомления
# ============================================
self.addEventListener('push', (event) => {
  console.log('[SW] Push received', event);

  let data = {
    title: 'Smart Diary',
    body: 'У вас новое уведомление!',
    icon: '/icons/icon-192.png',
    badge: '/icons/badge-72.png',
    vibrate: [200, 100, 200],
    tag: 'notification'
  };

  if (event.data) {
    try {
      data = { ...data, ...event.data.json() };
    } catch (e) {
      data.body = event.data.text();
    }
  }

  const options = {
    body: data.body,
    icon: data.icon,
    badge: data.badge,
    vibrate: data.vibrate,
    tag: data.tag,
    data: {
      url: data.url || '/',
      dateOfArrival: Date.now()
    },
    actions: [
      {
        action: 'open',
        title: 'Открыть'
      },
      {
        action: 'dismiss',
        title: 'Отмена'
```

```
    }
   ]
  };

  event.waitUntil(self.registration.showNotification(data.title, options));
});

// Обработка клика по уведомлению
self.addEventListener('notificationclick', (event) => {
  console.log('[SW] Notification click', event);

  event.notification.close();

  const urlToOpen = event.notification.data?.url || '/';

  event.waitUntil(
    clients.matchAll({ type: 'window', includeUncontrolled: true })
      .then((windowClients) => {
        for (const client of windowClients) {
          if (client.url === urlToOpen && 'focus' in client) {
            return client.focus();
          }
        }
        if (clients.openWindow) {
          return clients.openWindow(urlToOpen);
        }
      })
  );
});
```

Шаг 4. Создание страницы офлайн-режима

public/offline.html:

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Нет подключения - Smart Diary</title>
  <style>
    * {
      margin: 0;
      padding: 0;
      box-sizing: border-box;
    }
```

```
body {
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto, sans-serif;
  background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
  min-height: 100vh;
  display: flex;
  align-items: center;
  justify-content: center;
  padding: 20px;
}

.offline-container {
  max-width: 500px;
  width: 100%;
  background: white;
  border-radius: 24px;
  padding: 40px;
  text-align: center;
  box-shadow: 0 20px 40px rgba(0,0,0,0.1);
}

.offline-icon {
  width: 120px;
  height: 120px;
  margin-bottom: 24px;
}

h1 {
  font-size: 28px;
  color: #2c3e50;
  margin-bottom: 16px;
}

p {
  color: #7f8c8d;
  line-height: 1.6;
  margin-bottom: 24px;
}

.try-again-btn {
  background: #3498db;
  color: white;
  border: none;
  padding: 12px 32px;
  border-radius: 50px;
  font-size: 16px;
  cursor: pointer;
  transition: background 0.3s;
}
```

```
        .try-again-btn:hover {
          background: #2980b9;
        }

        .saved-tasks {
          margin-top: 32px;
          padding-top: 24px;
          border-top: 1px solid #ecf0f1;
        }

        .saved-tasks h3 {
          font-size: 16px;
          color: #34495e;
          margin-bottom: 16px;
        }

        .task-list {
          text-align: left;
          max-height: 300px;
          overflow-y: auto;
        }

        .task-item {
          background: #f8f9fa;
          border-radius: 12px;
          padding: 12px;
          margin-bottom: 8px;
        }

        .task-title {
          font-weight: 500;
          color: #2c3e50;
          margin-bottom: 4px;
        }

        .task-status {
          font-size: 12px;
          color: #7f8c8d;
        }
      </style>
    </head>
    <body>
      <div class="offline-container">
        <svg class="offline-icon" viewBox="0 0 24 24" fill="none"
xmlns="http://www.w3.org/2000/svg">
```

```
        <path d="M12 2C6.48 2 2 6.48 2 12s4.48 10 10 10 10-4.48 10-10S17.52 2 12 2zm0 4c1.1 0 2 .9 2 2s-.9 2-2 2-2-.9-2-2 .9-2 2-2zm0 13c-2.33 0-4.31-1.46-5.11-3.5h10.22c-.8 2.04-2.78 3.5-5.11 3.5z" fill="#e74c3c"/>
      </svg>

      <h1>Нет подключения к Интернету</h1>

      <p>
        Проверьте ваше интернет-соединение.<br>
        Вы можете продолжать работу с ранее загруженными данными.
      </p>

      <button class="try-again-btn" onclick="window.location.reload()">
        Попробовать снова
      </button>

      <div class="saved-tasks" id="saved-tasks-section" style="display: none;">
        <h3>Сохранённые задачи (офлайн)</h3>
        <div class="task-list" id="offline-tasks-list"></div>
      </div>
    </div>

    <script>
      // Загрузка сохранённых задач из IndexedDB
      if ('indexedDB' in window) {
        const request = indexedDB.open('SmartDiaryDB', 1);

        request.onsuccess = (event) => {
          const db = event.target.result;
          const transaction = db.transaction(['tasks'], 'readonly');
          const store = transaction.objectStore('tasks');
          const getAll = store.getAll();

          getAll.onsuccess = () => {
            if (getAll.result && getAll.result.length > 0) {
              const section = document.getElementById('saved-tasks-section');
              const container = document.getElementById('offline-tasks-list');
              section.style.display = 'block';

              getAll.result.slice(0, 10).forEach(task => {
                const taskEl = document.createElement('div');
                taskEl.className = 'task-item';
                taskEl.innerHTML = `
                  <div class="task-title">${task.title}</div>
                  <div class="task-status">Статус: ${task.status || 'Новая'}</div>
                `;
                container.appendChild(taskEl);
              });
```

```
        }
      };
    };
  }
 </script>
</body>
</html>
```

Шаг 5. Регистрация Service Worker в React

src/serviceWorkerRegistration.js:

```
// ========================================
// Регистрация Service Worker
// ========================================

const isLocalhost = Boolean(
 window.location.hostname === 'localhost' ||
  window.location.hostname === '[::1]' ||
  window.location.hostname.match(/^127(?:\.\d+){0,2}\.\d+$/)
);

export function register(config) {
 if (process.env.NODE_ENV === 'production' && 'serviceWorker' in navigator) {
  const publicUrl = new URL(process.env.PUBLIC_URL, window.location.href);
  if (publicUrl.origin !== window.location.origin) {
   return;
  }

  window.addEventListener('load', () => {
   const swUrl = `${process.env.PUBLIC_URL}/sw.js`;

   if (isLocalhost) {
    checkValidServiceWorker(swUrl, config);
    navigator.serviceWorker.ready.then(() => {
     console.log('Service Worker is ready');
    });
   } else {
    registerValidSW(swUrl, config);
   }
  });
 }
}

function registerValidSW(swUrl, config) {
 navigator.serviceWorker
  .register(swUrl)
```

```
    .then((registration) => {
      console.log('Service Worker registered:', registration);

      registration.onupdatefound = () => {
        const installingWorker = registration.installing;
        if (installingWorker == null) {
          return;
        }
        installingWorker.onstatechange = () => {
          if (installingWorker.state === 'installed') {
            if (navigator.serviceWorker.controller) {
              console.log('New content is available; please refresh.');
              if (config && config.onUpdate) {
                config.onUpdate(registration);
              }
            } else {
              console.log('Content is cached for offline use.');
              if (config && config.onSuccess) {
                config.onSuccess(registration);
              }
            }
          }
        };
      };
    })
    .catch((error) => {
      console.error('Error during service worker registration:', error);
    });
}

function checkValidServiceWorker(swUrl, config) {
  fetch(swUrl, {
    headers: { 'Service-Worker': 'script' },
  })
    .then((response) => {
      const contentType = response.headers.get('content-type');
      if (
        response.status === 404 ||
        (contentType != null && contentType.indexOf('javascript') === -1)
      ) {
        navigator.serviceWorker.ready.then((registration) => {
          registration.unregister().then(() => {
            window.location.reload();
          });
        });
      } else {
        registerValidSW(swUrl, config);
      }
```

```
    })
    .catch(() => {
      console.log('No internet connection found. App is running in offline mode.');
    });
}

export function unregister() {
  if ('serviceWorker' in navigator) {
    navigator.serviceWorker.ready
      .then((registration) => {
        registration.unregister();
      })
      .catch((error) => {
        console.error(error.message);
      });
  }
}
```

Шаг 6. Компонент PWA установки

src/components/pwa/PWAInstallPrompt.jsx:

```
import React, { useState, useEffect } from 'react';
import {
  Box,
  Button,
  Paper,
  Typography,
  IconButton,
  Slide,
} from '@mui/material';
import {
  Close as CloseIcon,
  GetApp as InstallIcon,
} from '@mui/icons-material';

const PWAInstallPrompt = () => {
  const [deferredPrompt, setDeferredPrompt] = useState(null);
  const [showPrompt, setShowPrompt] = useState(false);

  useEffect(() => {
    const handler = (e) => {
      e.preventDefault();
      setDeferredPrompt(e);
      // Показываем промпт не сразу, а через некоторое время
      setTimeout(() => {
```

```
      // Проверяем, не установили ли уже приложение
      const isInstalled = localStorage getItem('pwa-installed');
      if (!isInstalled) {
        setShowPrompt(true);
      }
    }, 30000); // Через 30 секунд после загрузки
  };

  window addEventListener('beforeinstallprompt', handler);

  // Проверка, открыто ли приложение как установленное
  if (window.matchMedia('(display-mode: standalone)').matches) {
    localStorage.setItem('pwa-installed', 'true');
  }

  return () => {
    window.removeEventListener('beforeinstallprompt', handler);
  };
}, []);

const handleInstall = async () => {
  if (!deferredPrompt) return;

  deferredPrompt.prompt();

  const { outcome } = await deferredPrompt.userChoice;
  console.log(`User response to the install prompt: ${outcome}`);

  if (outcome === 'accepted') {
    localStorage.setItem('pwa-installed', 'true');
    setShowPrompt(false);
  }

  setDeferredPrompt(null);
};

const handleDismiss = () => {
  setShowPrompt(false);
  localStorage.setItem('pwa-installed', 'dismissed');
};

if (!showPrompt) return null;

return (
  <Slide direction="up" in={showPrompt} mountOnEnter unmountOnExit>
    <Paper
      elevation={6}
      sx={{
```

```
                position: 'fixed',
                bottom: 20,
                left: 20,
                right: 20,
                maxWidth: 400,
                mx: 'auto',
                zIndex: 1300,
                borderRadius: 3,
                overflow: 'hidden',
              }}
            >
              <Box sx={{ p: 2, bgcolor: '#3498db', color: 'white', display: 'flex', justifyContent: 'space-between', alignItems: 'center' }}>
                <Typography variant="subtitle1" fontWeight="bold">
                  Установить приложение
                </Typography>
                <IconButton size="small" onClick={handleDismiss} sx={{ color: 'white' }}>
                  <CloseIcon />
                </IconButton>
              </Box>

              <Box sx={{ p: 2 }}>
                <Box display="flex" gap={2} mb={2}>
                  <img
                    src="/icons/icon-192.png"
                    alt="Smart Diary"
                    style={{ width: 48, height: 48, borderRadius: 12 }}
                  />
                  <Box>
                    <Typography variant="body1" fontWeight="500">
                      Smart Diary
                    </Typography>
                    <Typography variant="caption" color="text.secondary">
                      Управляйте задачами быстрее с установленным приложением
                    </Typography>
                  </Box>
                </Box>

                <Box display="flex" gap={2}>
                  <Button
                    variant="contained"
                    color="primary"
                    startIcon={<InstallIcon />}
                    onClick={handleInstall}
                    fullWidth
                  >
                    Установить
                  </Button>
```

```
          <Button
            variant="outlined"
            onClick={handleDismiss}
          >
            Не сейчас
          </Button>
        </Box>
      </Box>
    </Paper>
  </Slide>
 );
};

export default PWAInstallPrompt;
```

Шаг 7. Настройка push-уведомлений

src/services/pushService.js:

```
// ============================================
# Push-уведомления
# ============================================

export const pushService = {
 // Запрос разрешения на уведомления
 async requestPermission() {
  if (!('Notification' in window)) {
   console.log('This browser does not support notifications');
   return false;
  }

  if (Notification.permission === 'granted') {
   return true;
  }

  if (Notification.permission !== 'denied') {
   const permission = await Notification.requestPermission();
   return permission === 'granted';
  }

  return false;
 },

 // Подписка на push-уведомления
 async subscribeToPush() {
  if (!('serviceWorker' in navigator) || !('PushManager' in window)) {
   console.log('Push messaging is not supported');
```

```
      return null;
    }

    const registration = await navigator.serviceWorker.ready;
    const subscription = await registration.pushManager.subscribe({
      userVisibleOnly: true,
      applicationServerKey: this.urlBase64ToUint8Array(
        process.env.REACT_APP_VAPID_PUBLIC_KEY
      ),
    });

    // Отправляем подписку на сервер
    await this.sendSubscriptionToServer(subscription);

    return subscription;
  },

  // Отправка подписки на сервер
  async sendSubscriptionToServer(subscription) {
    try {
      const response = await fetch('/api/push/subscribe/', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
          'Authorization': `Bearer ${localStorage.getItem('access_token')}`,
        },
        body: JSON.stringify(subscription),
      });
      return response.ok;
    } catch (error) {
      console.error('Error sending subscription:', error);
      return false;
    }
  },

  // Отписка от push-уведомлений
  async unsubscribe() {
    const registration = await navigator.serviceWorker.ready;
    const subscription = await registration.pushManager.getSubscription();

    if (subscription) {
      await subscription.unsubscribe();
      await fetch('/api/push/unsubscribe/', {
        method: 'POST',
        headers: {
          'Authorization': `Bearer ${localStorage.getItem('access_token')}`,
        },
      });
```

```
  }
 },

 // Конвертация ключа VAPID
 urlBase64ToUint8Array(base64String) {
  const padding = '='.repeat((4 - base64String.length % 4) % 4);
  const base64 = (base64String + padding)
   .replace(/\-/g, '+')
   .replace(/_/g, '/');
  const rawData = window.atob(base64);
  const outputArray = new Uint8Array(rawData.length);
  for (let i = 0; i < rawData.length; ++i) {
   outputArray[i] = rawData.charCodeAt(i);
  }
  return outputArray;
 },
};
```

Шаг 8. Настройка IndexedDB для офлайн-кэша

src/services/indexedDBService.js:

```
// ============================================
# IndexedDB для офлайн-хранения данных
# ============================================

const DB_NAME = 'SmartDiaryDB';
const DB_VERSION = 1;

export const dbService = {
 async openDB() {
  return new Promise((resolve, reject) => {
   const request = indexedDB.open(DB_NAME, DB_VERSION);

   request.onerror = () => reject(request.error);
   request.onsuccess = () => resolve(request.result);

   request.onupgradeneeded = (event) => {
    const db = event.target.result;

    // Хранилище для задач
    if (!db.objectStoreNames.contains('tasks')) {
     const taskStore = db.createObjectStore('tasks', { keyPath: 'id' });
     taskStore.createIndex('status', 'status', { unique: false });
     taskStore.createIndex('deadline', 'deadline', { unique: false });
    }

    // Хранилище для проектов
```

```
      if (!db.objectStoreNames.contains('projects')) {
        const projectStore = db.createObjectStore('projects', { keyPath: 'id' });
        projectStore.createIndex('name', 'name', { unique: false });
      }

      // Хранилище для тегов
      if (!db.objectStoreNames.contains('tags')) {
        db.createObjectStore('tags', { keyPath: 'id' });
      }

      // Хранилище для пользователя
      if (!db.objectStoreNames.contains('user')) {
        db.createObjectStore('user', { keyPath: 'id' });
      }

      // Хранилище для очереди запросов (offline mutations)
      if (!db.objectStoreNames.contains('syncQueue')) {
        const syncStore = db.createObjectStore('syncQueue', {
          keyPath: 'id',
          autoIncrement: true
        });
        syncStore.createIndex('status', 'status', { unique: false });
        syncStore.createIndex('createdAt', 'createdAt', { unique: false });
      }
    };
  });
},

// Сохранение задач
async saveTasks(tasks) {
  const db = await this.openDB();
  const transaction = db.transaction(['tasks'], 'readwrite');
  const store = transaction.objectStore('tasks');

  for (const task of tasks) {
    store.put({ ...task, _cachedAt: Date.now() });
  }

  return new Promise((resolve, reject) => {
    transaction.oncomplete = () => resolve();
    transaction.onerror = () => reject(transaction.error);
  });
},

// Получение всех задач
async getTasks() {
  const db = await this.openDB();
  return new Promise((resolve, reject) => {
```

```
    const transaction = db.transaction(['tasks'], 'readonly');
    const store = transaction.objectStore('tasks');
    const request = store.getAll();

    request.onsuccess = () => resolve(request.result);
    request.onerror = () => reject(request.error);
  });
},

// Сохранение задачи
async saveTask(task) {
  const db = await this.openDB();
  const transaction = db.transaction(['tasks'], 'readwrite');
  const store = transaction.objectStore('tasks');
  store.put({ ...task, _cachedAt: Date.now() });

  return new Promise((resolve, reject) => {
    transaction.oncomplete = () => resolve();
    transaction.onerror = () => reject(transaction.error);
  });
},

// Удаление задачи
async deleteTask(taskId) {
  const db = await this.openDB();
  const transaction = db.transaction(['tasks'], 'readwrite');
  const store = transaction.objectStore('tasks');
  store.delete(taskId);

  return new Promise((resolve, reject) => {
    transaction.oncomplete = () => resolve();
    transaction.onerror = () => reject(transaction.error);
  });
},

// Добавление в очередь синхронизации
async addToSyncQueue(mutation) {
  const db = await this.openDB();
  const transaction = db.transaction(['syncQueue'], 'readwrite');
  const store = transaction.objectStore('syncQueue');

  const queuedItem = {
    ...mutation,
    status: 'pending',
    createdAt: Date.now(),
    retryCount: 0,
  };
```

```
    store.add(queuedItem);

    return new Promise((resolve, reject) => {
      transaction.oncomplete = () => resolve();
      transaction.onerror = () => reject(transaction.error);
    });
  },

  // Получение очереди синхронизации
  async getSyncQueue() {
    const db = await this.openDB();
    return new Promise((resolve, reject) => {
      const transaction = db.transaction(['syncQueue'], 'readonly');
      const store = transaction.objectStore('syncQueue');
      const index = store.index('status');
      const request = index.getAll('pending');

      request.onsuccess = () => resolve(request.result);
      request.onerror = () => reject(request.error);
    });
  },

  // Удаление из очереди синхронизации
  async removeFromSyncQueue(id) {
    const db = await this.openDB();
    const transaction = db.transaction(['syncQueue'], 'readwrite');
    const store = transaction.objectStore('syncQueue');
    store.delete(id);

    return new Promise((resolve, reject) => {
      transaction.oncomplete = () => resolve();
      transaction.onerror = () => reject(transaction.error);
    });
  },
};

// Обработка офлайн-режима
export const offlineHandler = {
  async syncTask(task) {
    await dbService.saveTask(task);
    await dbService.addToSyncQueue({
      type: 'CREATE_TASK',
      payload: task,
      url: '/api/tasks/',
      method: 'POST',
    });
  },
```

```
  async syncUpdateTask(taskId, updates) {
    await dbService.saveTask({ id: taskId, ...updates });
    await dbService.addToSyncQueue({
      type: 'UPDATE_TASK',
      payload: { id: taskId, ...updates },
      url: `/api/tasks/${taskId}/`,
      method: 'PUT',
    });
  },

  async syncDeleteTask(taskId) {
    await dbService.deleteTask(taskId);
    await dbService.addToSyncQueue({
      type: 'DELETE_TASK',
      payload: { id: taskId },
      url: `/api/tasks/${taskId}/`,
      method: 'DELETE',
    });
  },
};
```

Шаг 9. Регистрация Service Worker в main.jsx

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';
import * as serviceWorkerRegistration from './serviceWorkerRegistration';
import './styles/global.css';

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <Provider store={store}>
      <App />
    </Provider>
  </React.StrictMode>
);

// Регистрация Service Worker
serviceWorkerRegistration.register({
  onSuccess: (registration) => {
    console.log('Service Worker registered successfully');
  },
  onUpdate: (registration) => {
```

```
      console.log('New version available, reloading...');
      if (registration.waiting) {
        registration.waiting.postMessage({ type: 'SKIP_WAITING' });
      }
      window.location.reload();
    },
  });
```

Шаг 10. Обновление Navbar для отображения статуса PWA

src/components/common/Navbar.jsx (дополнение):

```
import { usePWA } from '../../hooks/usePWA';

// Внутри компонента
const { isInstallable, installPrompt } = usePWA();

// ... в панели инструментов
{isInstallable && (
  <IconButton color="inherit" onClick={installPrompt} title="Установить приложение">
    <GetAppIcon />
  </IconButton>
)}
```

Контрольные вопросы

1. Что такое PWA и каковы его основные компоненты?
2. Как работает Service Worker и каков его жизненный цикл?
3. Какие существуют стратегии кэширования в Service Worker?
4. Как реализовать офлайн-режим в PWA?
5. Как работают push-уведомления в PWA?
6. Что такое Web App Manifest и для чего он нужен?

Критерии оценки

Критерий	Баллы
Создан manifest.json с иконками	2
Настроен Service Worker для кэширования статики	2

Критерий	Баллы
Реализован офлайн-режим	2
Реализована возможность установки приложения	2
Настроены push-уведомления	1
Создан компонент PWAInstallPrompt	1
Итого	10

Результат выполнения

После выполнения лабораторной работы №19 студенты должны получить:

1. PWA-совместимое приложение с манифестом и сервис-воркером
2. Офлайн-режим с кэшированием статики и данных
3. Возможность установки приложения на устройство
4. Push-уведомления с подтверждением подписки
5. Очередь синхронизации для офлайн-мутаций данных

Лабораторная работа №20

Интеграция AI/ML (Искусственный интеллект)

Цель работы: Добавление «умных» функций с использованием AI: рекомендательная система для приоритетов задач, автоматическая категоризация задачи по проекту, интеграция OpenAI API для генерации описаний и помощи в планировании.

Задачи

№	Задача
1	Интегрировать OpenAI API для генерации описания задачи и рекомендаций
2	Реализовать автоматическую категоризацию задачи по проекту (классификация)
3	Реализовать рекомендательную систему для приоритета задачи на основе текста
4	Добавить AI-помощника для планирования дня
5	Реализовать кэширование результатов AI-запросов для оптимизации

Теоретическая часть

Искусственный интеллект в веб-приложениях

Подход	**Описание**	**Применение**
Внешнее AI API	Использование OpenAI, Hugging Face, Yandex GPT	Генерация текста, классификация
Встраивание модели	Локальное выполнение TensorFlow.js, ONNX	Офлайн-инференс
Микросервис ML	Отдельный сервис для модели (FastAPI + PyTorch)	Производительность

OpenAI API - сервис для работы с моделями GPT.

Основные эндпоинты:

1. /v1/chat/completions — диалоговые модели

2. /v1/completions — текстовые модели
3. /v1/embeddings — векторные представления

Промпт-инжиниринг

Промпт - инструкция для AI-модели.

Рекомендации:

1. Будьте конкретны
2. Давайте примеры (few-shot)
3. Ограничивайте ответ (max_tokens)
4. Задавайте формат вывода (JSON)

Ход выполнения работы

Шаг 1. Установка необходимых пакетов

```
# Бэкенд
pip install openai
pip install tenacity  # для повторных попыток
pip install redis  # для кэширования

# Фронтенд
npm install @mui/lab
```

Шаг 2. Создание AI-сервиса в Django

tasks/services/ai_service.py:

```
import json
import logging
from typing import Dict, List, Optional, Tuple
from django.conf import settings
from django.core.cache import cache
from openai import OpenAI
from tenacity import retry, stop_after_attempt, wait_exponential
```

```
logger = logging getLogger(__name__)


class AIService:
    """Сервис для работы с AI (OpenAI API)"""

    def __init__(self):
        self client = OpenAI(api_key=settings OPENAI_API_KEY)
        self model = "gpt-3.5-turbo"

    # ==========================================
    # Генерация описания задачи
    # ==========================================
    @retry(stop=stop_after_attempt(3), wait=wait_exponential(multiplier=1, min=2,
max=10))
    async def generate_task_description(self, title: str, project_name: str = None) ->
Optional[str]:
        """Генерация описания задачи на основе заголовка"""
        cache_key = f"ai:desc:{title}:{project_name or ''}"
        cached = cache get(cache_key)
        if cached:
            return cached

        prompt = f"""
        Ты — помощник по планированию задач. Напиши подробное описание
для задачи.

        Заголовок задачи: "{title}"
        """

        if project_name
            prompt += f"\nПроект: {project_name}\n"

        prompt += """
        Описание должно содержать:
        1. Конкретные шаги для выполнения задачи
        2. Рекомендуемые ресурсы (если применимо)
        3. Ожидаемый результат
        4. Возможные риски или подводные камни

        Ответ должен быть на русском языке, информативным и практичным.
        """

        try:
            response = self client chat completions create(
                model=self model,
                messages=[
```

```
                    {"role": "system", "content": "Ты — эксперт по продуктивности и управлению задачами."},
                    {"role": "user", "content": prompt}
                ],
                temperature=0.7,
                max_tokens=500
            )

            description = response.choices[0].message.content.strip()

            # Кэшируем на 1 час
            cache.set(cache_key, description, timeout=3600)
            return description

        except Exception as e:
            logger.error(f"Error generating task description: {e}")
            return None

    # ============================================
    # Рекомендация приоритета задачи
    # ============================================
    @retry(stop=stop_after_attempt(3), wait=wait_exponential(multiplier=1, min=2, max=10))
    async def suggest_priority(self, title: str, description: str = None) -> Dict:
        """Рекомендация приоритета на основе заголовка и описания"""
        cache_key = f"ai:priority:{title}:{description[:100] if description else ''}"
        cached = cache.get(cache_key)
        if cached:
            return cached

        prompt = f"""
        Оцени приоритет задачи по шкале: Low (низкий), Medium (средний), High (высокий).

        Заголовок: "{title}"
        """

        if description:
            prompt += f"\nОписание: {description[:500]}\n"

        prompt += """
        Учитывай:
        - Срочность и важность задачи
        - Влияние на другие задачи
        - Дедлайны (если указаны)

        Ответ должен быть в формате JSON:
        {
```

```
            "priority": "High/Medium/Low",
            "reason": "краткое обоснование выбора"
        }
        """

        try:
            response = self.client.chat.completions.create(
                model=self.model,
                messages=[
                    {"role": "system", "content": "Ты — эксперт по приоритизации задач."},
                    {"role": "user", "content": prompt}
                ],
                temperature=0.3,
                max_tokens=200,
                response_format={"type": "json_object"}
            )

            result = json.loads(response.choices[0].message.content)
            cache.set(cache_key, result, timeout=3600)
            return result

        except Exception as e:
            logger.error(f"Error suggesting priority: {e}")
            return {"priority": "Medium", "reason": "Рекомендация недоступна"}

    # ============================================
    # Автоматическая категоризация по проекту
    # ============================================
    @retry(stop=stop_after_attempt(3), wait=wait_exponential(multiplier=1, min=2,
max=10))
    async def categorize_task(self, title: str, available_projects: List[str]) ->
Optional[int]:
        """Определение проекта для задачи на основе заголовка"""
        cache_key = f"ai:category:{title}:{','.join(available_projects)}"
        cached = cache.get(cache_key)
        if cached:
            return cached

        if not available_projects:
            return None

        prompt = f"""
        Определи, к какому проекту относится задача.

        Задача: "{title}"

        Доступные проекты: {', '.join(available_projects)}
```

```
            Если ни один проект не подходит, ответь "None".
            Ответ должен содержать только название проекта или None.
            """

            try:
                response = self.client.chat.completions.create(
                    model=self.model,
                    messages=[
                        {"role": "system", "content": "Ты — системный аналитик."},
                        {"role": "user", "content": prompt}
                    ],
                    temperature=0.2,
                    max_tokens=50
                )

                result = response.choices[0].message.content.strip()
                result = result if result != "None" else None

                cache.set(cache_key, result, timeout=3600)
                return result

            except Exception as e:
                logger.error(f"Error categorizing task: {e}")
                return None

        # ============================================
        # AI-помощник для планирования дня
        # ============================================
        @retry(stop=stop_after_attempt(3), wait=wait_exponential(multiplier=1, min=2,
max=10))
        async def plan_day(self, tasks: List[Dict]) -> Dict:
            """Создание плана на день на основе списка задач"""
            cache_key = f"ai:plan:{hash(str(sorted([t['title'] for t in tasks])))}"
            cached = cache.get(cache_key)
            if cached:
                return cached

            tasks_text = "\n".join([
                f"- [{t.get('priority', 'Medium')}] {t['title']}"
                for t in tasks[:20]
            ])

            prompt = f"""
            Составь оптимальный план выполнения задач на сегодня.

            Список задач:
            {tasks_text}
```

```
            Ответ должен содержать:
            1. Рекомендуемый порядок выполнения задач (с обоснованием)
            2. Оценку времени на каждую задачу
            3. Советы по продуктивности

            Формат ответа: JSON
            {
               "order": ["задача1", "задача2", ...],
               "estimates": {"задача1": "30мин", ...},
               "tips": ["совет1", "совет2", ...],
               "total_estimated_time": "3ч 30м"
            }
            """

            try:
               response = self.client.chat.completions.create(
                  model=self.model,
                  messages=[
                     {"role": "system", "content": "Ты — персональный коуч по продуктивности."},
                     {"role": "user", "content": prompt}
                  ],
                  temperature=0.5,
                  max_tokens=800,
                  response_format={"type": "json_object"}
               )

               result = json.loads(response.choices[0].message.content)
               cache.set(cache_key, result, timeout=3600)
               return result

            except Exception as e:
               logger.error(f"Error planning day: {e}")
               return {
                  "order": [t['title'] for t in tasks],
                  "estimates": {},
                  "tips": ["Начните с самой важной задачи", "Используйте технику Pomodoro"],
                  "total_estimated_time": "Не определено"
               }

         # ==========================================
         # Генерация мотивационной цитаты
         # ==========================================
         @retry(stop=stop_after_attempt(3), wait=wait_exponential(multiplier=1, min=2, max=10))
         async def get_motivation(self) -> Dict:
            """Генерация мотивационной цитаты"""
```

```
        cache_key = "ai:motivation:daily"
        cached = cache.get(cache_key)
        if cached:
            return cached

        try:
            response = self.client.chat.completions.create(
                model=self.model,
                messages=[
                    {"role": "system", "content": "Ты — источник вдохновения и мотивации."},
                    {"role": "user", "content": "Напиши короткую мотивационную цитату о продуктивности и достижении целей (не более 100 символов). Формат: JSON {'quote': 'цитата', 'author': 'автор'}"}
                ],
                temperature=0.8,
                max_tokens=150,
                response_format={"type": "json_object"}
            )

            result = json.loads(response.choices[0].message.content)
            cache.set(cache_key, result, timeout=86400)  # на 24 часа
            return result

        except Exception as e:
            logger.error(f"Error getting motivation: {e}")
            return {"quote": "Начинай делать то, что нужно. Затем делай то, что возможно. И внезапно ты начнёшь делать невозможное.", "author": "Франциск Ассизский"}

    # ============================================
    # Умная фильтрация задач
    # ============================================
    async def smart_filter(self, tasks: List[Dict], query: str) -> List[int]:
        """Умный поиск задач по смыслу (семантический)"""
        cache_key = f"ai:search:{query}:{hash(str([t['id'] for t in tasks]))}"
        cached = cache.get(cache_key)
        if cached:
            return cached

        tasks_text = "\n".join([
            f"{t['id']}: {t['title']} - {t.get('description', '')[:100]}"
            for t in tasks[:50]
        ])

        prompt = f"""
        Найди задачи, наиболее соответствующие запросу: "{query}"
```

```
        Список задач:
        {tasks_text}

        Ответ должен содержать только ID задач через запятую.
        Например: "1, 5, 12"
        Если задачи не найдены, ответь "None"
        """

        try:
            response = self.client.chat.completions.create(
                model=self.model,
                messages=[
                    {"role": "system", "content": "Ты — помощник по поиску."},
                    {"role": "user", "content": prompt}
                ],
                temperature=0.1,
                max_tokens=200
            )

            result = response.choices[0].message.content.strip()

            if result and result != "None":
                task_ids = [int(id.strip()) for id in result.split(',')]
                cache.set(cache_key, task_ids, timeout=3600)
                return task_ids

            return []

        except Exception as e:
            logger.error(f"Error in smart filter: {e}")
            return []
```

Шаг 3. Создание API эндпоинтов для AI

tasks/views/ai_views.py:

```
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework.permissions import IsAuthenticated
from drf_spectacular.utils import extend_schema
from ..services.ai_service import AIService
from ..models import Task, Project

ai_service = AIService()


class GenerateDescriptionView(APIView):
```

```
    """Генерация описания задачи"""
    permission_classes = [IsAuthenticated]

    @extend_schema(summary="Генерация описания задачи AI")
    async def post(self, request):
        title = request.data.get('title')
        project_id = request.data.get('project_id')

        if not title:
            return Response({'error': 'Title is required'}, status=400)

        project_name = None
        if project_id:
            project = await Project.objects.filter(id=project_id,
owner=request.user).afirst()
            project_name = project.name if project else None

        description = await ai_service.generate_task_description(title, project_name)

        if description:
            return Response({'description': description})
        return Response({'error': 'Failed to generate description'}, status=500)


class SuggestPriorityView(APIView):
    """Рекомендация приоритета задачи"""
    permission_classes = [IsAuthenticated]

    @extend_schema(summary="Рекомендация приоритета задачи AI")
    async def post(self, request):
        title = request.data.get('title')
        description = request.data.get('description', '')

        if not title:
            return Response({'error': 'Title is required'}, status=400)

        result = await ai_service.suggest_priority(title, description)
        return Response(result)


class CategorizeTaskView(APIView):
    """Автоматическая категоризация задачи по проекту"""
    permission_classes = [IsAuthenticated]

    @extend_schema(summary="Автоматическая категоризация задачи")
    async def post(self, request):
        title = request.data.get('title')
```

```
        if not title:
            return Response({'error': 'Title is required'}, status=400)

        projects = await Project.objects.filter(owner=request.user).values_list('name', flat=True)
        projects_list = list(projects)

        if not projects_list:
            return Response({'suggested_project': None})

        suggested = await ai_service.categorize_task(title, projects_list)

        # Находим ID проекта по названию
        project_id = None
        if suggested:
            project = await Project.objects.filter(name=suggested, owner=request.user).afirst()
            project_id = project.id if project else None

        return Response({'suggested_project_id': project_id, 'suggested_project_name': suggested})


class PlanDayView(APIView):
    """AI-помощник для планирования дня"""
    permission_classes = [IsAuthenticated]

    @extend_schema(summary="Планирование дня с AI")
    async def get(self, request):
        # Получаем активные задачи пользователя
        tasks = await Task.objects.filter(
            user=request.user
        ).exclude(
            status='Completed'
        ).values('id', 'title', 'priority')[:20]

        tasks_list = list(tasks)

        if not tasks_list:
            return Response({
                'order': [],
                'estimates': {},
                'tips': ['У вас нет активных задач! Отличный день для отдыха.'],
                'total_estimated_time': '0ч'
            })

        plan = await ai_service.plan_day(tasks_list)
        return Response(plan)
```

```
class MotivationView(APIView):
    """Мотивационная цитата"""
    permission_classes = [IsAuthenticated]

    @extend_schema(summary="Мотивационная цитата")
    async def get(self, request):
        quote = await ai_service.get_motivation()
        return Response(quote)


class SmartSearchView(APIView):
    """Умный поиск задач"""
    permission_classes = [IsAuthenticated]

    @extend_schema(summary="Умный поиск задач")
    async def post(self, request):
        query = request.data.get('query')

        if not query or len(query) < 3:
            return Response({'task_ids': []})

        tasks = await Task.objects.filter(
            user=request.user
        ).values('id', 'title', 'description')[:100]

        tasks_list = list(tasks)
        task_ids = await ai_service.smart_filter(tasks_list, query)

        return Response({'task_ids': task_ids})
```

Шаг 4. Добавление маршрутов

tasks/urls.py (дополнение):

```
from .views.ai_views import (
    GenerateDescriptionView, SuggestPriorityView, CategorizeTaskView,
    PlanDayView, MotivationView, SmartSearchView
)

urlpatterns += [
    path('api/ai/generate-description/', GenerateDescriptionView.as_view(), name='ai-generate-desc'),
    path('api/ai/suggest-priority/', SuggestPriorityView.as_view(), name='ai-suggest-priority'),
    path('api/ai/categorize/', CategorizeTaskView.as_view(), name='ai-categorize'),
    path('api/ai/plan-day/', PlanDayView.as_view(), name='ai-plan-day'),
    path('api/ai/motivation/', MotivationView.as_view(), name='ai-motivation'),
```

```
    path('api/ai/smart-search/', SmartSearchView.as_view(), name='ai-smart-search'),
]
```

Шаг 5. Настройка переменных окружения

.env (дополнение):

```
# OpenAI
OPENAI_API_KEY=your-openai-api-key-here
```

settings.py (дополнение):

python

```
OPENAI_API_KEY = os.environ.get('OPENAI_API_KEY', '')
```

Шаг 6. Создание AI-компонентов на фронтенде

src/components/ai/AIAssistant.jsx:

```
import React, { useState } from 'react';
import {
  Box,
  Fab,
  Drawer,
  List,
  ListItem,
  ListItemText,
  ListItemIcon,
  Divider,
  Typography,
  CircularProgress,
  Card,
  CardContent,
  IconButton,
  TextField,
  Button,
  Chip
} from '@mui/material';
import {
  SmartToy as AiIcon,
  Close as CloseIcon,
  AutoAwesome as AutoAwesomeIcon,
  Lightbulb as LightbulbIcon,
  Schedule as ScheduleIcon,
  Search as SearchIcon,
} from '@mui/icons-material';
```

```
import apiClient from '../../api/client';

const AIAssistant = () => {
  const [open, setOpen] = useState(false);
  const [loading, setLoading] = useState(false);
  const [plan, setPlan] = useState(null);
  const [motivation, setMotivation] = useState(null);
  const [searchQuery, setSearchQuery] = useState('');
  const [searchResults, setSearchResults] = useState([]);

  const fetchMotivation = async () => {
    try {
      const response = await apiClient.get('/ai/motivation/');
      setMotivation(response.data);
    } catch (error) {
      console.error('Error fetching motivation:', error);
    }
  };

  const fetchDayPlan = async () => {
    setLoading(true);
    try {
      const response = await apiClient.get('/ai/plan-day/');
      setPlan(response.data);
    } catch (error) {
      console.error('Error fetching plan:', error);
    } finally {
      setLoading(false);
    }
  };

  const handleSmartSearch = async () => {
    if (!searchQuery.trim()) return;

    setLoading(true);
    try {
      const response = await apiClient.post('/ai/smart-search/', {
        query: searchQuery
      });
      setSearchResults(response.data.task_ids || []);
    } catch (error) {
      console.error('Error in smart search:', error);
    } finally {
      setLoading(false);
    }
  };

  const handleOpen = () => {
```

```
    setOpen(true);
    if (!motivation) fetchMotivation();
  };

  return (
    <>
      <Fab
        color="secondary"
        sx={{ position: 'fixed', bottom: 20, right: 20 }}
        onClick={handleOpen}
      >
        <AiIcon />
      </Fab>

      <Drawer anchor="right" open={open} onClose={() => setOpen(false)}>
        <Box sx={{ width: 400, p: 2 }}>
          <Box display="flex" justifyContent="space-between" alignItems="center"
mb={2}>
            <Typography variant="h6" display="flex" alignItems="center" gap={1}>
              <AutoAwesomeIcon color="secondary" />
              AI-ассистент
            </Typography>
            <IconButton onClick={() => setOpen(false)}>
              <CloseIcon />
            </IconButton>
          </Box>

          <Divider sx={{ mb: 2 }} />

          {/* Мотивация */}
          {motivation && (
            <Card sx={{ mb: 2, bgcolor: '#f0f7ff' }}>
              <CardContent>
                <Typography variant="body2" fontStyle="italic">
                  «{motivation.quote}»
                </Typography>
                <Typography variant="caption" color="text.secondary" mt={1}
display="block">
                  — {motivation.author}
                </Typography>
              </CardContent>
            </Card>
          )}

          {/* Планирование дня */}
          <Button
            variant="outlined"
            startIcon={<ScheduleIcon />}
```

```
          onClick={fetchDayPlan}
          fullWidth
          sx={{ mb: 2 }}
          disabled={loading}
        >
          План на сегодня
        </Button>

        {loading && <CircularProgress size={24} sx={{ display: 'block', mx: 'auto', my:
2 }} />}

        {plan && (
          <Card sx={{ mb: 2 }}>
            <CardContent>
              <Typography variant="subtitle2" gutterBottom>
                Рекомендуемый порядок:
              </Typography>
              <ol style={{ margin: 0, paddingLeft: 20 }}>
                {plan.order?.map((task, idx) => (
                  <li key={idx}>
                    <Typography variant="body2">
                      {task}
                      {plan.estimates?.[task] && (
                        <Chip size="small" label={plan.estimates[task]} sx={{ ml: 1 }} />
                      )}
                    </Typography>
                  </li>
                ))}
              </ol>

              <Typography variant="subtitle2" mt={2} gutterBottom>
                Советы:
              </Typography>
              <ul style={{ margin: 0, paddingLeft: 20 }}>
                {plan.tips?.map((tip, idx) => (
                  <li key={idx}>
                    <Typography variant="body2">{tip}</Typography>
                  </li>
                ))}
              </ul>

              <Typography variant="caption" color="text.secondary" mt={1}
display="block">
                Общее время: {plan.total_estimated_time}
              </Typography>
            </CardContent>
          </Card>
        )}
```

```
            {/* Умный поиск */}
            <Typography variant="subtitle2" gutterBottom>
              Умный поиск
            </Typography>
            <Box display="flex" gap={1} mb={2}>
              <TextField
                size="small"
                placeholder="Поиск по смыслу..."
                value={searchQuery}
                onChange={(e) => setSearchQuery(e.target.value)}
                fullWidth
                onKeyPress={(e) => e.key === 'Enter' && handleSmartSearch()}
              />
              <Button variant="contained" onClick={handleSmartSearch}
disabled={loading}>
                <SearchIcon />
              </Button>
            </Box>

            {searchResults.length > 0 && (
              <Typography variant="body2" color="text.secondary">
                Найдено задач: {searchResults.length}
              </Typography>
            )}
          </Box>
        </Drawer>
      </>
    );
  };

  export default AIAssistant;
```

Шаг 7. AI-кнопки в форме задачи

src/components/tasks/AIAssistButtons.jsx:

```
  import React, { useState } from 'react';
  import { Button, CircularProgress, Tooltip } from '@mui/material';
  import { AutoAwesome as AutoAwesomeIcon } from '@mui/icons-material';
  import apiClient from '../../api/client';

  const AIAssistButtons = ({ title, projectId, onDescriptionGenerated,
onPrioritySuggested }) => {
    const [loadingDesc, setLoadingDesc] = useState(false);
    const [loadingPriority, setLoadingPriority] = useState(false);
```

```
const generateDescription = async () => {
  if (!title) {
    alert('Сначала введите заголовок задачи');
    return;
  }

  setLoadingDesc(true);
  try {
    const response = await apiClient.post('/ai/generate-description/', {
      title,
      project_id: projectId,
    });
    if (response.data.description && onDescriptionGenerated) {
      onDescriptionGenerated(response.data.description);
    }
  } catch (error) {
    console.error('Error generating description:', error);
  } finally {
    setLoadingDesc(false);
  }
};

const suggestPriority = async () => {
  if (!title) {
    alert('Сначала введите заголовок задачи');
    return;
  }

  setLoadingPriority(true);
  try {
    const response = await apiClient.post('/ai/suggest-priority/', {
      title,
    });
    if (response.data.priority && onPrioritySuggested) {
      onPrioritySuggested(response.data.priority);
    }
  } catch (error) {
    console.error('Error suggesting priority:', error);
  } finally {
    setLoadingPriority(false);
  }
};

return (
  <>
    <Tooltip title="Сгенерировать описание с помощью AI">
      <Button
        size="small"
```

```
            startIcon={loadingDesc ? <CircularProgress size={16} /> : <AutoAwesomeIcon
/>}
            onClick={generateDescription}
            disabled={loadingDesc || !title}
          >
            AI описание
          </Button>
        </Tooltip>

        <Tooltip title="Рекомендовать приоритет">
          <Button
            size="small"
            startIcon={loadingPriority ? <CircularProgress size={16} /> :
<AutoAwesomeIcon />}
            onClick={suggestPriority}
            disabled={loadingPriority || !title}
          >
            AI приоритет
          </Button>
        </Tooltip>
      </>
    );
};

export default AIAssistButtons
```

Шаг 8. Интеграция AI в TaskForm

src/components/tasks/TaskForm.jsx (дополнение):

```
import AIAssistButtons from './AIAssistButtons';

// Внутри компонента TaskForm добавьте:
const [generatedDescription, setGeneratedDescription] = useState('');

// Функции для AI
const handleDescriptionGenerated = (description) => {
  setGeneratedDescription(description);
  formik.setFieldValue('description', description);
};

const handlePrioritySuggested = (priority) => {
  formik.setFieldValue('priority', priority);
};

// В JSX, рядом с полями описания и приоритета:
<Box display="flex" alignItems="flex-start" gap={1}>
```

```
  <TextField fullWidth ... />
  <AIAssistButtons
   title={formik values title}
   projectId={formik values project_id}
   onDescriptionGenerated={handleDescriptionGenerated}
   onPrioritySuggested={handlePrioritySuggested}
  />
</Box>
```

Шаг 9. Настройка Redis для кэширования AI-запросов

```
# Установка django-redis
pip install django-redis
```

settings.py (дополнение):

```
CACHES = {
    'default': {
        'BACKEND': 'django_redis.cache.RedisCache',
        'LOCATION': 'redis://localhost:6379/1',
        'OPTIONS': {
            'CLIENT_CLASS': 'django_redis.client.DefaultClient',
        }
    }
}

# Время жизни кэша AI-запросов (в секундах)
AI_CACHE_TIMEOUT = 3600  # 1 час
```

Контрольные вопросы

1. Какие подходы к интеграции AI в веб-приложения существуют?
2. Как работает промпт-инжиниринг для OpenAI API?
3. Зачем нужно кэшировать результаты AI-запросов?
4. Как реализована автоматическая категоризация задач?
5. Какие меры безопасности следует учитывать при использовании AI API?

Критерии оценки

Критерий	Баллы
Интегрирован OpenAI API	2
Реализована генерация описания задачи	2
Реализована рекомендация приоритета	2
Реализован AI-помощник для планирования дня	2
Реализовано кэширование AI-запросов	1
Создан компонент AIAssistant	1
Итого	10

Результат выполнения

После выполнения лабораторной работы №20 студенты должны получить:

1. Интеграцию с OpenAI API для AI-функций
2. Генерацию описания задачи AI
3. Рекомендацию приоритета на основе текста задачи
4. AI-помощника для планирования дня
5. Умный поиск задач по смыслу
6. Кэширование AI-запросов для оптимизации

Лабораторная работа №21

Оптимизация производительности

Цель работы: Анализ и улучшение производительности веб-приложения: оптимизация бэкенда (запросы к БД, кэширование), фронтенда (бандл, рендеринг) и инфраструктуры.

Задачи

№	Задача
1	Провести аудит производительности с использованием Lighthouse, Django Debug Toolbar
2	Оптимизировать запросы к БД (устранение N+1, добавление индексов)
3	Оптимизировать фронтенд (code splitting, lazy loading, мемоизация)
4	Внедрить кэширование на бэкенде (Redis)
5	Оптимизировать загрузку бандла (анализ, уменьшение размера)

Теоретическая часть

Метрики производительности

Метрика	**Описание**	**Хорошее значение**
FCP (First Contentful Paint)	Первая отрисовка контента	< 1.8 с
LCP (Largest Contentful Paint)	Отрисовка основного контента	< 2.5 с
TTI (Time to Interactive)	Время до интерактивности	< 3.8 с
TBT (Total Blocking Time)	Общее время	< 300 мс

Метрика	Описание	Хорошее значение
	блокировки	
CLS (Cumulative Layout Shift)	Сдвиги макета	< 0.1

Проблема N+1 запросов

```
# Проблема: N+1 запрос
tasks = Task objects.all()
for task in tasks:
    print(task project name)  # Каждый раз новый запрос к БД

# Решение: select_related / prefetch_related
tasks = Task objects select_related('project').all()
```

Оптимизация React

Техника	Описание
React.memo	Кэширование рендеринга компонента
useMemo	Кэширование результатов вычислений
useCallback	Кэширование функций
React.lazy + Suspense	Ленивая загрузка компонентов

Ход выполнения работы

Шаг 1. Установка инструментов для аудита

```
# Бэкенд (Django)
pip install django-debug-toolbar django-silk

# Фронтенд
npm install --save-dev webpack-bundle-analyzer
```

Шаг 2. Настройка Django Debug Toolbar

settings.py:

```
INSTALLED_APPS = [
    # ...
    'debug_toolbar',
]

MIDDLEWARE = [
    'debug_toolbar.middleware.DebugToolbarMiddleware',
    # ...
]

INTERNAL_IPS = [
    '127.0.0.1',
]

# Настройка панели
DEBUG_TOOLBAR_PANELS = [
    'debug_toolbar.panels.history.HistoryPanel',
    'debug_toolbar.panels.versions.VersionsPanel',
    'debug_toolbar.panels.timer.TimerPanel',
    'debug_toolbar.panels.settings.SettingsPanel',
    'debug_toolbar.panels.headers.HeadersPanel',
    'debug_toolbar.panels.request.RequestPanel',
    'debug_toolbar.panels.sql.SQLPanel',
    'debug_toolbar.panels.staticfiles.StaticFilesPanel',
    'debug_toolbar.panels.templates.TemplatesPanel',
    'debug_toolbar.panels.cache.CachePanel',
    'debug_toolbar.panels.signals.SignalsPanel',
    'debug_toolbar.panels.logging.LoggingPanel',
    'debug_toolbar.panels.redirects.RedirectsPanel',
    'debug_toolbar.panels.profiling.ProfilingPanel',
]
```

urls.py:

```
from django urls import include, path


urlpatterns = [
    path('__debug__/', include('debug_toolbar.urls')),
    # ...
]
```

Шаг 3. Оптимизация запросов к БД

tasks/services/optimized_service.py:

```
from django db models import Prefetch, Count, Q
from django core paginator import Paginator
from django utils decorators import method_decorator
from django views decorators cache import cache_page
from django core cache import cache
from . models import Task, Project, Tag, TaskTag

class OptimizedTaskService:
    """Сервис с оптимизированными запросами к БД"""

    @staticmethod
    def get_user_tasks_optimized(user_id, page=1, page_size=20):
        """Оптимизированное получение задач пользователя"""
        # Используем select_related для связей один-к-одному (Project)
        # Используем prefetch_related для связей многие-ко-многим (Tags)
        queryset = Task objects.filter(
            user_id=user_id
        ) select_related(
            'project'  # Убирает N+1 для project
        ) prefetch_related(
            Prefetch('task_tags', queryset=TaskTag objects select_related('tag')),
            'subtasks'  # Для подзадач
        ) only(  # Выбираем только нужные поля
            'id', 'title', 'description', 'status', 'priority',
            'deadline', 'created_at', 'project_id', 'parent_id'
        ) order_by('-created_at')

        # Пагинация
        paginator = Paginator(queryset, page_size)
        return paginator.get_page(page)
```

```python
    @staticmethod
    def get_task_statistics_optimized(user_id):
        """Оптимизированная статистика (один запрос)"""
        from django.db.models import Count, Q, Case, When, IntegerField, Sum

        stats = Task.objects.filter(user_id=user_id).aggregate(
            total=Count('id'),
            completed=Count('id', filter=Q(status='Completed')),
            in_progress=Count('id', filter=Q(status='InProgress')),
            new=Count('id', filter=Q(status='New')),
            overdue=Count('id', filter=Q(deadline__lt=timezone.now()) &
~Q(status='Completed')),
            high_priority=Count('id', filter=Q(priority='High')),
            medium_priority=Count('id', filter=Q(priority='Medium')),
            low_priority=Count('id', filter=Q(priority='Low')),
        )

        # Добавляем вычисляемые поля
        stats['completion_rate'] = round(
            stats['completed'] / stats['total'] * 100, 1
        ) if stats['total'] > 0 else 0

        return stats

    @staticmethod
    def get_user_projects_with_tasks_count(user_id):
        """Проекты с количеством задач (один запрос)"""
        return Project.objects.filter(
            owner_id=user_id
        ).annotate(
            tasks_count=Count('tasks'),
            completed_tasks=Count('tasks', filter=Q(tasks__status='Completed'))
        ).prefetch_related(
            Prefetch('tasks', queryset=Task.objects.only('id', 'title', 'status'))
        )

    @staticmethod
    def bulk_create_tasks(tasks_data):
        """Массовое создание задач (одна операция)"""
        tasks = [Task(**data) for data in tasks_data]
        return Task.objects.bulk_create(tasks)

    @staticmethod
    def bulk_update_tasks(task_ids, updates):
        """Массовое обновление задач"""
        return Task.objects.filter(id__in=task_ids).update(**updates)
```

```
class CacheService:
    """Сервис для работы с кэшем"""

    @staticmethod
    def get_or_cache(key, callback, timeout=3600):
        """Получение данных из кэша или выполнение callback"""
        cached_value = cache.get(key)
        if cached_value is not None:
            return cached_value

        value = callback()
        cache.set(key, value, timeout)
        return value

    @staticmethod
    def invalidate_user_cache(user_id):
        """Инвалидация кэша пользователя"""
        cache.delete_pattern(f'user_{user_id}_*')
        cache.delete(f'user_stats_{user_id}')
        cache.delete(f'user_tasks_{user_id}')

    @staticmethod
    def cache_task_list(user_id, page, callback):
        """Кэширование списка задач с учетом пагинации"""
        cache_key = f'user_tasks_{user_id}_page_{page}'
        return CacheService.get_or_cache(cache_key, callback, timeout=300)


# Добавление индексов в модели (models.py)
"""
class Task(models.Model):
    # ...
    class Meta:
        indexes = [
            models.Index(fields=['user_id', 'status']),
            models.Index(fields=['user_id', 'deadline']),
            models.Index(fields=['user_id', 'priority']),
            models.Index(fields=['user_id', '-created_at']),
            models.Index(fields=['project_id', 'status']),
            models.Index(fields=['parent_id']),
        ]
"""
```

Шаг 4. Настройка кэширования в Django

settings.py:

```python
# Настройка Redis кэша
CACHES = {
    'default': {
        'BACKEND': 'django_redis.cache.RedisCache',
        'LOCATION': 'redis://localhost:6379/1',
        'OPTIONS': {
            'CLIENT_CLASS': 'django_redis.client.DefaultClient',
            'PARSER_CLASS': 'redis.connection.HiredisParser',
            'CONNECTION_POOL_CLASS': 'redis.BlockingConnectionPool',
            'CONNECTION_POOL_CLASS_KWARGS': {
                'max_connections': 50,
                'timeout': 20,
            },
            'MAX_CONNECTIONS': 1000,
            'PICKLE_VERSION': -1,
        },
        'KEY_PREFIX': 'smartdiary',
        'TIMEOUT': 86400,  # 24 часа
    }
}

# Кэширование представлений
CACHE_MIDDLEWARE_SECONDS = 300
CACHE_MIDDLEWARE_KEY_PREFIX = 'smartdiary'

# Сессии в Redis
SESSION_ENGINE = 'django.contrib.sessions.backends.cache'
SESSION_CACHE_ALIAS = 'default'
```

Шаг 5. Кэширование представлений

tasks/views/optimized_views.py:

python

```python
from django views decorators cache import cache_page
from django views decorators vary import vary_on_headers
from django utils decorators import method_decorator
from rest_framework views import APIView
from rest_framework response import Response
from . services optimized_service import OptimizedTaskService, CacheService


@method_decorator(cache_page(300), name='dispatch')
@method_decorator(vary_on_headers('Authorization'), name='dispatch')
class OptimizedTaskListView(APIView):
    """Оптимизированное получение списка задач с кэшированием"""

    def get(self, request):
        user_id = request user.id
        page = int(request query_params get('page', 1))
```

```
    def fetch_tasks():
        return OptimizedTaskService.get_user_tasks_optimized(user_id, page)

    tasks = CacheService.get_or_cache(
        f'user_tasks_{user_id}_page_{page}',
        fetch_tasks,
        timeout=300
    )

    return Response({
        'tasks': tasks.object_list,
        'pagination': {
            'page': tasks.number,
            'page_size': tasks.paginator.per_page,
            'total': tasks.paginator.count,
            'total_pages': tasks.paginator.num_pages,
        }
    })


@method_decorator(cache_page(600), name='dispatch')
class OptimizedStatisticsView(APIView):
    """Оптимизированная статистика с кэшированием"""

    def get(self, request):
        user_id = request.user.id

        def fetch_stats():
            return OptimizedTaskService.get_task_statistics_optimized(user_id)

        stats = CacheService.get_or_cache(
            f'user_stats_{user_id}',
            fetch_stats,
            timeout=600
        )

        return Response(stats)
```

Шаг 6. Оптимизация фронтенда (React)

src/optimizations/RouteLazyLoading.jsx:

```
import React, { lazy, Suspense } from 'react';
import { CircularProgress, Box } from '@mui/material';

// Ленивая загрузка компонентов
const TaskList = lazy(() => import('../pages/TaskListPage'));
```

```
const TaskForm = lazy(() => import('../pages/TaskCreatePage'));
const TaskDetail = lazy(() => import('../pages/TaskDetailPage'));
const StatisticsPage = lazy(() => import('../pages/StatisticsPage'));
const ProfilePage = lazy(() => import('../pages/ProfilePage'));

// Компонент-обёртка с Suspense
const LazyRoute = ({ children }) => (
  <Suspense
   fallback={
     <Box display="flex" justifyContent="center" alignItems="center"
minHeight="200px">
       <CircularProgress />
     </Box>
   }
  >
   {children}
  </Suspense>
);

export const routes = [
  { path: '/', element: () => <TaskList />, lazy: true },
  { path: '/tasks', element: () => <TaskList />, lazy: true },
  { path: '/tasks/create', element: () => <TaskForm />, lazy: true },
  { path: '/tasks/:id', element: () => <TaskDetail />, lazy: true },
  { path: '/statistics', element: () => <StatisticsPage />, lazy: true },
  { path: '/profile', element: () => <ProfilePage />, lazy: true },
];
```

Шаг 7. Оптимизация React-компонентов

src/optimizations/TaskCardOptimized.jsx:

```
import React, { memo, useCallback, useMemo } from 'react';
import { Card, CardContent, Typography, Chip, Box, Checkbox } from
'@mui/material';
import { format, isToday, isTomorrow, differenceInDays } from 'date-fns';

const priorityColors = {
  Low: '#2ecc71',
  Medium: '#f39c12',
  High: '#e74c3c',
};

const statusColors = {
  New: '#3498db',
  InProgress: '#f39c12',
  Completed: '#2ecc71',
```

```
};

// useMemo для форматирования даты
const useFormattedDate = (deadline) => {
  return useMemo(() => {
    if (!deadline) return null;
    const date = new Date(deadline);
    if (isToday(date)) return 'Сегодня';
    if (isTomorrow(date)) return 'Завтра';
    const days = differenceInDays(date, new Date());
    if (days < 7) return `Через ${days} дня`;
    return format(date, 'dd.MM.yyyy');
  }, [deadline]);
};

// React.memo для предотвращения лишних рендеров
const TaskCardOptimized = memo(({ task, onStatusChange, onEdit, onDelete }) => {
  const formattedDeadline = useFormattedDate(task.deadline);

  // useCallback для мемоизации обработчиков
  const handleStatusChange = useCallback((e) => {
    const newStatus = e.target.checked ? 'Completed' : 'New';
    onStatusChange?.(task.id, newStatus);
  }, [task.id, onStatusChange]);

  const handleEdit = useCallback(() => {
    onEdit?.(task.id);
  }, [task.id, onEdit]);

  const handleDelete = useCallback(() => {
    onDelete?.(task.id);
  }, [task.id, onDelete]);

  return (
    <Card sx={{ mb: 2, transition: 'box-shadow 0.2s', '&:hover': { boxShadow: 3 } }}>
      <CardContent>
        <Box display="flex" alignItems="center">
          <Checkbox
            checked={task.status === 'Completed'}
            onChange={handleStatusChange}
            sx={{ mr: 1 }}
          />
          <Typography
            variant="h6"
            sx={{
              flex: 1,
              textDecoration: task.status === 'Completed' ? 'line-through' : 'none',
              opacity: task.status === 'Completed' ? 0.7 : 1,
```

```
          }}
        >
          {task.title}
        </Typography>
      </Box>

      <Box display="flex" flexWrap="wrap" gap={1} ml={5}>
        <Chip
          size="small"
          label={task.priority}
          sx={{ bgcolor: priorityColors[task.priority], color: 'white' }}
        />
        <Chip
          size="small"
          label={task.status}
          sx={{ bgcolor: statusColors[task.status], color: 'white' }}
        />
        {formattedDeadline && (
          <Chip size="small" label={formattedDeadline} variant="outlined" />
        )}
      </Box>
    </CardContent>
  </Card>
 );
});

TaskCardOptimized.displayName = 'TaskCardOptimized';

export default TaskCardOptimized;
```

Шаг 8. Оптимизация бандла

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',
   open: true,
   gzipSize: true,
   brotliSize: true,
  }),
```

```
  ],
  build: {
    rollupOptions: {
      output: {
        manualChunks: {
          'react-vendor': ['react', 'react-dom', 'react-router-dom'],
          'mui-vendor': ['@mui/material', '@emotion/react', '@emotion/styled'],
          'redux-vendor': ['@reduxjs/toolkit', 'react-redux'],
          'date-vendor': ['date-fns', 'dayjs'],
        },
      },
    },
    target: 'es2020',
    minify: 'terser',
    terserOptions: {
      compress: {
        drop_console: true,
        drop_debugger: true,
      },
    },
    chunkSizeWarningLimit: 500,
  },
  optimizeDeps: {
    include: ['react', 'react-dom', '@mui/material', '@reduxjs/toolkit'],
  },
});
```

Шаг 9. Оптимизация изображений

src/utils/imageOptimization.js:

```
export const getOptimizedImageUrl = (url, width = 800, quality = 80) => {
  if (!url) return '';

  // Для изображений, загруженных на сервер
  if (url.includes('/media/')) {
    return `${url}?w=${width}&q=${quality}`;
  }

  return url;
};

// Компонент оптимизированного изображения
export const OptimizedImage = React.memo(({ src, alt, width, height, className }) =>
{
  const [loaded, setLoaded] = useState(false);
  const optimizedSrc = getOptimizedImageUrl(src, width);
```

```
  return (
    <div style={{ position: 'relative', width, height }}>
      {!loaded && <div className="image-placeholder" />}
      <img
        src={optimizedSrc}
        alt={alt}
        width={width}
        height={height}
        className={className}
        loading="lazy"
        onLoad={() => setLoaded(true)}
        style={{ opacity: loaded ? 1 : 0, transition: 'opacity 0.3s' }}
      />
    </div>
  );
});
```

Шаг 10. Оптимизация API запросов (React Query)

src/hooks/useOptimizedTasks.js:

```
import { useQuery, useMutation, useQueryClient } from '@tanstack/react-query';
import apiClient from '../api/client';

// Ключи кэша
const taskKeys = {
  all: ['tasks'],
  lists: () => [...taskKeys.all, 'list'],
  list: (filters) => [...taskKeys.lists(), { ...filters }],
  details: () => [...taskKeys.all, 'detail'],
  detail: (id) => [...taskKeys.details(), id],
};

// Хук для получения задач с оптимизированным кэшированием
export const useOptimizedTasks = (filters = {}, options = {}) => {
  return useQuery({
    queryKey: taskKeys.list(filters),
    queryFn: async () => {
      const { data } = await apiClient.get('/tasks/optimized/', { params: filters });
      return data;
    },
    staleTime: 5 * 60 * 1000, // 5 минут
    cacheTime: 10 * 60 * 1000, // 10 минут
    refetchOnWindowFocus: false,
    keepPreviousData: true,
    ...options,
```

```
  });
};

// Хук для получения деталей задачи
export const useOptimizedTaskDetail = (taskId) => {
  return useQuery({
    queryKey: taskKeys.detail(taskId),
    queryFn: async () => {
      const { data } = await apiClient.get(`/tasks/${taskId}/`);
      return data;
    },
    enabled: !!taskId,
    staleTime: 5 * 60 * 1000,
  });
};

// Хук для мутации создания задачи
export const useCreateTask = () => {
  const queryClient = useQueryClient();

  return useMutation({
    mutationFn: async (newTask) => {
      const { data } = await apiClient.post('/tasks/', newTask);
      return data;
    },
    onSuccess: () => {
      // Инвалидация списка задач
      queryClient.invalidateQueries({ queryKey: taskKeys.lists() });
    },
  });
};

// Хук для мутации обновления задачи
export const useUpdateTask = () => {
  const queryClient = useQueryClient();

  return useMutation({
    mutationFn: async ({ id, ...updates }) => {
      const { data } = await apiClient.put(`/tasks/${id}/`, updates);
      return data;
    },
    onSuccess: (data, variables) => {
      queryClient.invalidateQueries({ queryKey: taskKeys.detail(variables.id) });
      queryClient.invalidateQueries({ queryKey: taskKeys.lists() });
    },
  });
};
```

Шаг 11. Проверка производительности с Lighthouse

```
# Установка Lighthouse CLI
npm install -g lighthouse

# Запуск аудита
lighthouse http://localhost:3000 --view --preset=desktop

# Запуск с сохранением отчета
lighthouse http://localhost:3000 --output html --output-path ./lighthouse-report.html
```

Шаг 12. Оптимизация Nginx

nginx/nginx.conf (оптимизированный):

```
user nginx;
worker_processes auto;
worker_rlimit_nofile 65535;

events {
    worker_connections 4096;
    use epoll;
    multi_accept on;
}

http {
    # Базовые настройки
    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    keepalive_timeout 65;
    types_hash_max_size 2048;
    client_max_body_size 10M;

    # Gzip сжатие
    gzip on;
    gzip_vary on;
    gzip_proxied any
```

```
        gzip_comp_level 6;
        gzip_types text/plain text/css application/json application/javascript text/xml
application/xml application/xml+rss text/javascript;

        # Кэширование статики
        location ~* \.(jpg|jpeg|png|gif|ico|css|js)$ {
            expires 1y;
            add_header Cache-Control "public, immutable";
        }

        # Брокирование API
        location /api/ {
            proxy_pass http://backend:8000;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection 'upgrade';
            proxy_set_header Host $host;
            proxy_cache_bypass $http_upgrade;
            proxy_buffering off;
            proxy_cache api_cache;
            proxy_cache_valid 200 1m;
            add_header X-Cache-Status $upstream_cache_status;
        }

        # WebSocket
        location /ws/ {
            proxy_pass http://backend:8000;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection "upgrade";
            proxy_set_header Host $host;
            proxy_read_timeout 86400;
        }
    }
```

Контрольные вопросы

1. Какие метрики производительности веб-приложения вы знаете?
2. Что такое проблема N+1 запросов и как её решить?
3. Как работает кэширование в Django и React?
4. Какие техники оптимизации React-компонентов вы знаете?
5. Как уменьшить размер бандла приложения?
6. Что такое code splitting и lazy loading?
7. Как настроить кэширование статических файлов в Nginx?

Критерии оценки

Критерий	Баллы
Проведен аудит производительности (Lighthouse, Debug Toolbar)	2
Оптимизированы запросы к БД (устранение N+1, индексы)	2
Внедрено кэширование на бэкенде (Redis)	2
Реализован code splitting и lazy loading	2
Произведена оптимизация бандла	1
Настроено кэширование статики в Nginx	1
Итого	10

Результат выполнения

После выполнения лабораторной работы №21 студенты должны получить:

1. Аудит производительности с рекомендациями
2. Оптимизированные запросы к БД (решение N+1, индексы)
3. Кэширование на бэкенде и фронтенде
4. Уменьшенный размер бандла (code splitting, lazy loading)
5. Настроенное кэширование статических файлов в Nginx
6. Улучшенные метрики Lighthouse (целевые значения)

Лабораторная работа №22

Административная панель и дашборды

Цель работы: Создание панели администратора для управления системой: управление пользователями, просмотр системных логов, отображение метрик, настройка системных параметров.

Задачи

№	Задача
1	Создать административную панель с дашбордом (React + Django Admin)
2	Реализовать управление пользователями (блокировка, роли, просмотр)
3	Добавить просмотр системных логов и активности
4	Реализовать дашборд с бизнес-метриками
5	Добавить управление системными настройками

Теоретическая часть

Django Admin vs Кастомная админ-панель

Подход	**Преимущества**	**Недостатки**
Django Admin	Быстро, бесплатно, ORM-интеграция	Ограниченная кастомизация
Кастомная панель	Полный контроль, UX, React	Требует времени
Гибридный подход	Django Admin для данных + React для метрик	Сложность поддержки

Роли и права

Роль	Доступ к админ-панели	Возможности
Superuser	Полный доступ	Всё
Staff	Ограниченный доступ	Назначенные модели
Admin	Кастомная панель	Управление пользователями, просмотр метрик

Метрики для дашборда

Категория	Метрики
Пользователи	Всего, новые за день/неделю, активные, заблокированные
Задачи	Всего, создано за день, выполнено, просрочено
Производительность	Среднее время выполнения задачи, заполненность
Система	Использование CPU/RAM, количество запросов, ошибки

Ход выполнения работы

Шаг 1. Расширение модели пользователя

users/models.py (создать новое приложение):

python manage.py startapp users

settings.py:

```
INSTALLED_APPS = [
    'users',
```

```
    # ...
]

AUTH_USER_MODEL = 'users.User'
users/models.py:
python
from django.contrib.auth.models import AbstractUser
from django.db import models


class User(AbstractUser):
    """Расширенная модель пользователя"""

    ROLE_CHOICES = (
        ('user', 'Пользователь'),
        ('moderator', 'Модератор'),
        ('admin', 'Администратор'),
    )

    avatar = models.ImageField(upload_to='avatars/', null=True, blank=True)
    phone = models.CharField(max_length=20, blank=True)
    role = models.CharField(max_length=20, choices=ROLE_CHOICES, default='user')
    is_blocked = models.BooleanField(default=False)
    blocked_at = models.DateTimeField(null=True, blank=True)
    blocked_by = models.ForeignKey(
        'self',
        on_delete=models.SET_NULL,
        null=True,
        blank=True,
        related_name='blocked_users'
    )
    last_login_ip = models.GenericIPAddressField(null=True, blank=True)
    email_verified = models.BooleanField(default=False)
    email_verified_at = models.DateTimeField(null=True, blank=True)

    class Meta:
        ordering = ['-date_joined']
        indexes = [
            models.Index(fields=['role']),
            models.Index(fields=['is_blocked']),
            models.Index(fields=['date_joined']),
        ]

    def __str__(self):
        return self.username

    @property
    def is_admin_user(self):
```

```
        return self.role == 'admin' or self.is_superuser

    @property
    def is_moderator(self):
        return self.role == 'moderator' or self.is_staff
```

Шаг 2. Создание миграций

python manage.py makemigrations users

python manage.py migrate

Шаг 3. Кастомный Django Admin

users/admin.py:

```
from django.contrib import admin
from django.contrib.auth.admin import UserAdmin as BaseUserAdmin
from django.utils.html import format_html
from .models import User


@admin.register(User)
class UserAdmin(BaseUserAdmin):
    list_display = ('username', 'email', 'role', 'is_blocked', 'date_joined',
'avatar_preview')
    list_filter = ('role', 'is_blocked', 'is_active', 'date_joined')
    search_fields = ('username', 'email', 'phone')

    fieldsets = BaseUserAdmin.fieldsets + (
        ('Дополнительная информация', {
            'fields': ('avatar', 'phone', 'role', 'is_blocked', 'blocked_reason', 'last_login_ip')
        }),
    )

    def avatar_preview(self, obj):
        if obj.avatar:
            return format_html('<img src="{}" width="50" height="50" style="border-
radius: 50%;" />', obj.avatar.url)
        return '-'
    avatar_preview.short_description = 'Аватар'

    actions = ['block_users', 'unblock_users', 'make_moderator']

    def block_users(self, request, queryset):
```

```
        queryset.update(is_blocked=True)
    block_users.short_description = 'Заблокировать выбранных пользователей'

    def unblock_users(self, request, queryset):
        queryset.update(is_blocked=False)
    unblock_users.short_description = 'Разблокировать выбранных пользователей'

    def make_moderator(self, request, queryset):
        queryset.update(role='moderator')
    make_moderator.short_description = 'Назначить модератором'
```

Шаг 4. Создание API для админ-панели

users/views/admin_views.py:

```
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework.permissions import IsAdminUser
from django.db.models import Count, Q, Sum, Avg
from django.utils import timezone
from datetime import timedelta
from ..models import User
from tasks.models import Task, ActivityLog, Comment


class AdminStatsView(APIView):
    """Статистика для админ-панели"""
    permission_classes = [IsAdminUser]

    def get(self, request):
        now = timezone.now()
        today = now.date()
        week_ago = now - timedelta(days=7)
        month_ago = now - timedelta(days=30)

        # Пользователи
        total_users = User.objects.count()
        new_users_today = User.objects.filter(date_joined__date=today).count()
        new_users_week = User.objects.filter(date_joined__gte=week_ago).count()
        new_users_month = User.objects.filter(date_joined__gte=month_ago).count()
        blocked_users = User.objects.filter(is_blocked=True).count()
        active_users = User.objects.filter(is_active=True, is_blocked=False).count()

        # Задачи
        total_tasks = Task.objects.count()
        tasks_created_today = Task.objects.filter(created_at__date=today).count()
        tasks_created_week = Task.objects.filter(created_at__gte=week_ago).count()
```

```
tasks_completed_today = Task.objects.filter(
    status='Completed',
    completed_at__date=today
).count()
overdue_tasks = Task.objects.filter(
    deadline__lt=now,
    status__in=['New', 'InProgress']
).count()

# Комментарии
total_comments = Comment.objects.count()
comments_today = Comment.objects.filter(created_at__date=today).count()

# Активность
total_activities = ActivityLog.objects.count()
activities_today = ActivityLog.objects.filter(created_at__date=today).count()

# Распределение по ролям
roles_distribution = User.objects.values('role').annotate(count=Count('id'))

# Динамика по дням (последние 7 дней)
daily_stats = []
for i in range(7):
    date = (now - timedelta(days=i)).date()
    daily_stats.append({
        'date': date.isoformat(),
        'new_users': User.objects.filter(date_joined__date=date).count(),
        'new_tasks': Task.objects.filter(created_at__date=date).count(),
        'completed_tasks': Task.objects.filter(
            status='Completed',
            completed_at__date=date
        ).count(),
    })

return Response({
    'users': {
        'total': total_users,
        'new_today': new_users_today,
        'new_week': new_users_week,
        'new_month': new_users_month,
        'blocked': blocked_users,
        'active': active_users,
    },
    'tasks': {
        'total': total_tasks
        'created_today': tasks_created_today,
        'created_week': tasks_created_week,
        'completed_today': tasks_completed_today,
```

```
            'overdue': overdue_tasks,
        },
        'engagement': {
            'total_comments': total_comments,
            'comments_today': comments_today,
            'total_activities': total_activities,
            'activities_today': activities_today,
        },
        'roles_distribution': roles_distribution,
        'daily_stats': daily_stats,
    })


class AdminUsersView(APIView):
    """Управление пользователями для админа"""
    permission_classes = [IsAdminUser]

    def get(self, request):
        users = User.objects.all().order_by('-date_joined')

        # Поиск
        search = request.query_params.get('search')
        if search:
            users = users.filter(
                Q(username__icontains=search) |
                Q(email__icontains=search) |
                Q(phone__icontains=search)
            )

        # Фильтрация
        role = request.query_params.get('role')
        if role:
            users = users.filter(role=role)

        is_blocked = request.query_params.get('is_blocked')
        if is_blocked is not None:
            users = users.filter(is_blocked=is_blocked == 'true')

        # Пагинация
        page = int(request.query_params.get('page', 1))
        page_size = int(request.query_params.get('page_size', 20))
        start = (page - 1) * page_size
        end = start + page_size

        total = users.count()

        return Response({
            'users': list(users[start:end].values(
```

```
                'id', 'username', 'email', 'role', 'is_blocked',
                'is_active', 'date_joined', 'last_login', 'phone'
            )),
            'pagination': {
                'page': page,
                'page_size': page_size,
                'total': total,
                'total_pages': (total + page_size - 1) // page_size,
            }
        })

    def patch(self, request, user_id):
        """Обновление пользователя (блокировка, роль)"""
        try:
            user = User.objects.get(id=user_id)
        except User.DoesNotExist:
            return Response({'error': 'User not found'}, status=404)

        if 'is_blocked' in request.data:
            user.is_blocked = request.data['is_blocked']
            if user.is_blocked:
                user.blocked_at = timezone.now()
                user.blocked_by_id = request.user.id
        if 'role' in request.data:
            user.role = request.data['role']

        user.save()
        return Response({'message': 'User updated'})


class AdminLogsView(APIView):
    """Просмотр системных логов"""
    permission_classes = [IsAdminUser]

    def get(self, request):
        # Используем ActivityLog для отображения активности
        logs = ActivityLog.objects.select_related('user', 'task').order_by('-created_at')

        # Фильтрация
        user_id = request.query_params.get('user_id')
        if user_id:
            logs = logs.filter(user_id=user_id)

        action_type = request.query_params.get('action_type')
        if action_type:
            logs = logs.filter(action_type=action_type)

        # Пагинация
```

```
        page = int(request.query_params.get('page', 1))
        page_size = int(request.query_params.get('page_size', 50))
        start = (page - 1) * page_size
        end = start + page_size

        total = logs.count()

        return Response({
            'logs': [
                {
                    'id': log.id,
                    'user': log.user.username if log.user else None,
                    'action_type': log.action_type,
                    'old_value': log.old_value,
                    'new_value': log.new_value,
                    'task_id': log.task_id,
                    'task_title': log.task.title if log.task else None,
                    'created_at': log.created_at.isoformat(),
                }
                for log in logs[start:end]
            ],
            'pagination': {
                'page': page,
                'page_size': page_size,
                'total': total,
                'total_pages': (total + page_size - 1) // page_size,
            }
        })


class AdminSettingsView(APIView):
    """Управление системными настройками"""
    permission_classes = [IsAdminUser]

    # Системные настройки по умолчанию
    DEFAULT_SETTINGS = {
        'site_name': 'Smart Diary',
        'site_description': 'Умный ежедневник для управления задачами',
        'maintenance_mode': False,
        'registration_enabled': True,
        'default_task_priority': 'Medium',
        'reminder_hours': 24,
        'max_tasks_per_project': 100,
        'max_tags_per_task': 10,
        'enable_email_notifications': True,
        'enable_push_notifications': True,
        'session_timeout_minutes': 60,
    }
```

```
    def get(self, request):
        from django.core.cache import cache

        settings = cache.get('system_settings')
        if not settings:
            settings = self.DEFAULT_SETTINGS.copy()

            # Загрузка из модели Settings (если есть)
            from .models import SystemSetting
            for setting in SystemSetting.objects.all():
                settings[setting.key] = setting.value

        return Response(settings)

    def post(self, request):
        from django.core.cache import cache
        from .models import SystemSetting

        for key, value in request.data.items():
            if key in self.DEFAULT_SETTINGS:
                SystemSetting.objects.update_or_create(
                    key=key,
                    defaults={'value': value}
                )

        cache.set('system_settings', request.data, timeout=3600)

        return Response({'message': 'Settings saved'})
```

Шаг 5. Модель системных настроек

users/models.py (дополнение):

```
class SystemSetting(models.Model):
    """Системные настройки"""
    key = models.CharField(max_length=100, unique=True)
    value = models.JSONField()
    updated_at = models.DateTimeField(auto_now=True)
    updated_by = models.ForeignKey(
        User,
        on_delete=models.SET_NULL,
        null=True,
        blank=True,
        related_name='updated_settings'
    )
```

```
        class Meta:
            ordering = ['key']

        def __str__(self):
            return f"{self.key}: {self.value}"
```

Шаг 6. Добавление маршрутов

users/urls.py:

python

```
from django.urls import path
from .views.admin_views import AdminStatsView, AdminUsersView, AdminLogsView, AdminSettingsView

urlpatterns = [
    path('api/admin/stats/', AdminStatsView.as_view(), name='admin-stats'),
    path('api/admin/users/', AdminUsersView.as_view(), name='admin-users'),
    path('api/admin/users/<int:user_id>/', AdminUsersView.as_view(), name='admin-user-update'),
    path('api/admin/logs/', AdminLogsView.as_view(), name='admin-logs'),
    path('api/admin/settings/', AdminSettingsView.as_view(), name='admin-settings'),
]
```

Шаг 7. Фронтенд: компонент AdminDashboard

src/pages/admin/AdminDashboard.jsx:

```
import React, { useState, useEffect } from 'react';
import {
    Container,
    Grid,
    Paper,
    Typography,
    Box,
    Card,
    CardContent,
    IconButton,
    Table,
    TableBody,
    TableCell,
    TableContainer,
    TableHead,
    TableRow,
    Chip,
    TextField,
    InputAdornment,
    Select,
    MenuItem,
    FormControl,
```

```
    InputLabel,
    Pagination,
    Switch,
    FormControlLabel,
    Tab,
    Tabs,
    CircularProgress,
} from '@mui/material';
import {
    People as PeopleIcon,
    Task as TaskIcon,
    Comment as CommentIcon,
    Block as BlockIcon,
    Search as SearchIcon,
    Refresh as RefreshIcon,
    Delete as DeleteIcon,
    Edit as EditIcon,
} from '@mui/icons-material';
import { Line } from 'react-chartjs-2';
import { Chart as ChartJS, CategoryScale, LinearScale, PointElement, LineElement,
Title, Tooltip, Legend } from 'chart.js';
import apiClient from '../../api/client';

ChartJS.register(CategoryScale, LinearScale, PointElement, LineElement, Title,
Tooltip, Legend);

const StatCard = ({ title, value, icon, color }) => (
    <Card sx={{ height: '100%' }}>
        <CardContent>
            <Box display="flex" justifyContent="space-between" alignItems="center">
                <Box>
                    <Typography color="text.secondary" variant="body2">
                        {title}
                    </Typography>
                    <Typography variant="h4" component="div" sx={{ mt: 1 }}>
                        {value}
                    </Typography>
                </Box>
                <Box sx={{ bgcolor: color, borderRadius: 2, p: 1 }}>
                    {icon}
                </Box>
            </Box>
        </CardContent>
    </Card>
);

const AdminDashboard = () => {
    const [stats, setStats] = useState(null);
```

```
const [users, setUsers] = useState([]);
const [logs, setLogs] = useState([]);
const [settings, setSettings] = useState({});
const [loading, setLoading] = useState(true);
const [tabValue, setTabValue] = useState(0);
const [userPage, setUserPage] = useState(1);
const [userSearch, setUserSearch] = useState('');
const [userRoleFilter, setUserRoleFilter] = useState('');

useEffect(() => {
  fetchAllData();
}, [userPage, userSearch, userRoleFilter]);

const fetchAllData = async () => {
  setLoading(true);
  try {
    const [statsRes, usersRes, logsRes, settingsRes] = await Promise.all([
      apiClient.get('/admin/stats/'),
      apiClient.get('/admin/users/', {
        params: {
          page: userPage,
          search: userSearch,
          role: userRoleFilter,
        }
      }),
      apiClient.get('/admin/logs/'),
      apiClient.get('/admin/settings/'),
    ]);
    setStats(statsRes.data);
    setUsers(usersRes.data);
    setLogs(logsRes.data);
    setSettings(settingsRes.data);
  } catch (error) {
    console.error('Error fetching admin data:', error);
  } finally {
    setLoading(false);
  }
};

const handleBlockUser = async (userId, isBlocked) => {
  try {
    await apiClient.patch(`/admin/users/${userId}/`, { is_blocked: !isBlocked });
    fetchAllData();
  } catch (error) {
    console.error('Error blocking user:', error);
  }
};
```

```
  const handleUpdateRole = async (userId, role) => {
    try {
      await apiClient.patch(`/admin/users/${userId}/`, { role });
      fetchAllData();
    } catch (error) {
      console.error('Error updating role:', error);
    }
  };

  const handleSaveSettings = async () => {
    try {
      await apiClient.post('/admin/settings/', settings);
      alert('Настройки сохранены');
    } catch (error) {
      console.error('Error saving settings:', error);
    }
  };

  if (loading) {
    return (
      <Box display="flex" justifyContent="center" alignItems="center"
minHeight="400px">
        <CircularProgress />
      </Box>
    );
  }

  // Данные для графика
  const chartData = {
    labels: stats?.daily_stats?.map(d => d.date.slice(5)) || [],
    datasets: [
      {
        label: 'Новые пользователи',
        data: stats?.daily_stats?.map(d => d.new_users) || [],
        borderColor: '#3498db',
        backgroundColor: 'rgba(52, 152, 219, 0.1)',
        fill: true,
      },
      {
        label: 'Новые задачи',
        data: stats?.daily_stats?.map(d => d.new_tasks) || [],
        borderColor: '#2ecc71',
        backgroundColor: 'rgba(46, 204, 113, 0.1)',
        fill: true,
      },
      {
        label: 'Выполненные задачи',
        data: stats?.daily_stats?.map(d => d.completed_tasks) || [],
```

```
            borderColor: '#f39c12',
            backgroundColor: 'rgba(243, 156, 18, 0.1)',
            fill: true,
          },
        ],
      };

      return (
        <Container maxWidth="xl" sx={{ mt: 4, mb: 4 }}>
          <Box display="flex" justifyContent="space-between" alignItems="center"
mb={3}>
            <Typography variant="h4">Административная панель</Typography>
            <IconButton onClick={fetchAllData}>
              <RefreshIcon />
            </IconButton>
          </Box>

          {/* Статистические карточки */}
          <Grid container spacing={3} sx={{ mb: 4 }}>
            <Grid item xs={12} sm={6} md={3}>
              <StatCard
                title="Всего пользователей"
                value={stats?.users?.total || 0}
                icon={<PeopleIcon sx={{ fontSize: 32, color: 'white' }} />}
                color="#3498db"
              />
            </Grid>
            <Grid item xs={12} sm={6} md={3}>
              <StatCard
                title="Всего задач"
                value={stats?.tasks?.total || 0}
                icon={<TaskIcon sx={{ fontSize: 32, color: 'white' }} />}
                color="#2ecc71"
              />
            </Grid>
            <Grid item xs={12} sm={6} md={3}>
              <StatCard
                title="Заблокировано"
                value={stats?.users?.blocked || 0}
                icon={<BlockIcon sx={{ fontSize: 32, color: 'white' }} />}
                color="#e74c3c"
              />
            </Grid>
            <Grid item xs={12} sm={6} md={3}>
              <StatCard
                title="Комментариев"
                value={stats?.engagement?.total_comments || 0}
                icon={<CommentIcon sx={{ fontSize: 32, color: 'white' }} />}
```

```
                    color="#9b59b6"
                  />
                </Grid>
              </Grid>

              {/* График динамики */}
              <Paper sx={{ p: 2, mb: 4 }}>
                <Typography variant="h6" gutterBottom>Динамика за последние 7
дней</Typography>
                <Line data={chartData} options={{ responsive: true }} />
              </Paper>

              {/* Tabs */}
              <Paper sx={{ p: 2 }}>
                <Tabs value={tabValue} onChange={(e, v) => setTabValue(v)} sx={{ mb: 2 }}>
                  <Tab label="Пользователи" />
                  <Tab label="Логи активности" />
                  <Tab label="Системные настройки" />
                </Tabs>

                {/* Вкладка: Пользователи */}
                {tabValue === 0 && (
                  <>
                    <Box display="flex" gap={2} sx={{ mb: 2 }}>
                      <TextField
                        size="small"
                        placeholder="Поиск пользователей..."
                        value={userSearch}
                        onChange={(e) => setUserSearch(e target value)}
                        InputProps={{
                          startAdornment: (
                            <InputAdornment position="start">
                              <SearchIcon />
                            </InputAdornment>
                          ),
                        }}
                        sx={{ flex: 1 }}
                      />
                      <FormControl size="small" sx={{ minWidth: 150 }}>
                        <InputLabel>Роль</InputLabel>
                        <Select
                          value={userRoleFilter}
                          onChange={(e) => setUserRoleFilter(e target value)}
                          label="Роль"
                        >
                          <MenuItem value="">Все</MenuItem>
                          <MenuItem value="user">Пользователь</MenuItem>
                          <MenuItem value="moderator">Модератор</MenuItem>
```

```
                        <MenuItem value="admin">Администратор</MenuItem>
                      </Select>
                    </FormControl>
                  </Box>

                  <TableContainer>
                    <Table>
                      <TableHead>
                        <TableRow>
                          <TableCell>ID</TableCell>
                          <TableCell>Имя пользователя</TableCell>
                          <TableCell>Email</TableCell>
                          <TableCell>Роль</TableCell>
                          <TableCell>Статус</TableCell>
                          <TableCell>Дата регистрации</TableCell>
                          <TableCell>Действия</TableCell>
                        </TableRow>
                      </TableHead>
                      <TableBody>
                        {users?.users?.map((user) => (
                          <TableRow key={user.id}>
                            <TableCell>{user.id}</TableCell>
                            <TableCell>{user.username}</TableCell>
                            <TableCell>{user.email}</TableCell>
                            <TableCell>
                              <Select
                                size="small"
                                value={user.role}
                                onChange={(e) => handleUpdateRole(user.id,
e.target.value)}
                                sx={{ minWidth: 120 }}
                              >
                                <MenuItem value="user">Пользователь</MenuItem>
                                <MenuItem
value="moderator">Модератор</MenuItem>
                                <MenuItem
value="admin">Администратор</MenuItem>
                              </Select>
                            </TableCell>
                            <TableCell>
                              <Chip
                                label={user.is_blocked ? 'Заблокирован' : 'Активен'}
                                color={user.is_blocked ? 'error' : 'success'}
                                size="small"
                              />
                            </TableCell>
                            <TableCell>
                              {new Date(user.date_joined).toLocaleDateString('ru-RU')}
```

```
                                    </TableCell>
                                    <TableCell>
                                      <Switch
                                        checked={user.is_blocked}
                                        onChange={() => handleBlockUser(user.id,
user.is_blocked)}
                                        color="error"
                                      />
                                    </TableCell>
                                  </TableRow>
                                ))}
                              </TableBody>
                            </Table>
                          </TableContainer>

                          <Box display="flex" justifyContent="center" mt={2}>
                            <Pagination
                              count={users?.pagination?.total_pages || 1}
                              page={userPage}
                              onChange={(e, p) => setUserPage(p)}
                              color="primary"
                            />
                          </Box>
                        </>
                      )}

                      {/* Вкладка: Логи активности */}
                      {tabValue === 1 && (
                        <TableContainer>
                          <Table size="small">
                            <TableHead>
                              <TableRow>
                                <TableCell>Время</TableCell>
                                <TableCell>Пользователь</TableCell>
                                <TableCell>Действие</TableCell>
                                <TableCell>Задача</TableCell>
                              </TableRow>
                            </TableHead>
                            <TableBody>
                              {logs?.logs?.map((log) => (
                                <TableRow key={log.id}>
                                  <TableCell>
                                    {new Date(log.created_at).toLocaleString('ru-RU')}
                                  </TableCell>
                                  <TableCell>{log.user || 'Система'}</TableCell>
                                  <TableCell>{log.action_type}</TableCell>
                                  <TableCell>{log.task_title || '-'}</TableCell>
                                </TableRow>
```

```
                ))}
              </TableBody>
            </Table>
          </TableContainer>
        )}

        {/* Вкладка: Системные настройки */}
        {tabValue === 2 && (
          <Box>
            <Grid container spacing={3}>
              <Grid item xs={12} md={6}>
                <TextField
                  label="Название сайта"
                  fullWidth
                  value={settings.site_name || ''}
                  onChange={(e) => setSettings({ ...settings, site_name:
e.target.value })}
                  sx={{ mb: 2 }}
                />
                <TextField
                  label="Описание сайта"
                  fullWidth
                  multiline
                  rows={2}
                  value={settings.site_description || ''}
                  onChange={(e) => setSettings({ ...settings, site_description:
e.target.value })}
                  sx={{ mb: 2 }}
                />
                <FormControlLabel
                  control={
                    <Switch
                      checked={settings.maintenance_mode || false}
                      onChange={(e) => setSettings({ ...settings,
maintenance_mode: e.target.checked })}
                    />
                  }
                  label="Режим обслуживания"
                />
                <FormControlLabel
                  control={
                    <Switch
                      checked={settings.registration_enabled !== false}
                      onChange={(e) => setSettings({ ...settings,
registration_enabled: e.target.checked })}
                    />
                  }
                  label="Регистрация новых пользователей"
```

```
                />
              </Grid>
              <Grid item xs={12} md={6}>
                <FormControl fullWidth sx={{ mb: 2 }}>
                  <InputLabel>Приоритет по умолчанию</InputLabel>
                  <Select
                    value={settings.default_task_priority || 'Medium'}
                    onChange={(e) => setSettings({ ...settings,
default_task_priority: e.target.value })}
                    label="Приоритет по умолчанию"
                  >
                    <MenuItem value="Low">Низкий</MenuItem>
                    <MenuItem value="Medium">Средний</MenuItem>
                    <MenuItem value="High">Высокий</MenuItem>
                  </Select>
                </FormControl>
                <TextField
                  label="Часов до напоминания"
                  type="number"
                  fullWidth
                  value={settings.reminder_hours || 24}
                  onChange={(e) => setSettings({ ...settings, reminder_hours:
parseInt(e.target.value) })}
                  sx={{ mb: 2 }}
                />
                <FormControlLabel
                  control={
                    <Switch
                      checked={settings.enable_email_notifications !== false}
                      onChange={(e) => setSettings({ ...settings,
enable_email_notifications: e.target.checked })}
                    />
                  }
                  label="Email-уведомления"
                />
                <FormControlLabel
                  control={
                    <Switch
                      checked={settings.enable_push_notifications !== false}
                      onChange={(e) => setSettings({ ...settings,
enable_push_notifications: e.target.checked })}
                    />
                  }
                  label="Push-уведомления"
                />
              </Grid>
            </Grid>
            <Box display="flex" justifyContent="flex-end" mt={3}>
```

```
                    <Button variant="contained" onClick={handleSaveSettings}>
                      Сохранить настройки
                    </Button>
                  </Box>
                </Box>
              )}
            </Paper>
          </Container>
        );
};

export default AdminDashboard
```

Контрольные вопросы

1. Какие подходы к созданию админ-панели существуют?
2. Как организовано разграничение доступа к админ-панели?
3. Какие метрики важны для бизнес-дашборда?
4. Как реализована блокировка пользователей?
5. Как хранятся и изменяются системные настройки?
6. Какие инструменты используются для визуализации метрик?

Критерии оценки

Критерий	Баллы
Создана админ-панель с дашбордом	2
Реализовано управление пользователями (блокировка, роли)	2
Реализован просмотр системных логов	2
Создан дашборд с графиками метрик	2
Реализовано управление системными настройками	1
Настроена защита (только для администраторов)	1

Критерий	Баллы
Итого	10

Результат выполнения

После выполнения лабораторной работы №22 студенты должны получить:

1. Административную панель с полным функционалом
2. Управление пользователями (блокировка, изменение ролей)
3. Просмотр системных логов активности
4. Дашборд с бизнес-метриками и графиками
5. Системные настройки с возможностью изменения

Лабораторная работа №23

Развертывание в облако (Yandex Cloud / AWS)

Цель работы: Развертывание веб-приложения в облачной среде (Yandex Cloud / AWS) с настройкой managed базы данных, S3-хранилища для медиафайлов, HTTPS и CI/CD.

Задачи

№	Задача
1	Выбрать и настроить облачного провайдера (Yandex Cloud / AWS)
2	Настроить виртуальную машину для бэкенда
3	Настроить managed базу данных PostgreSQL
4	Настроить S3-совместимое хранилище (Object Storage) для

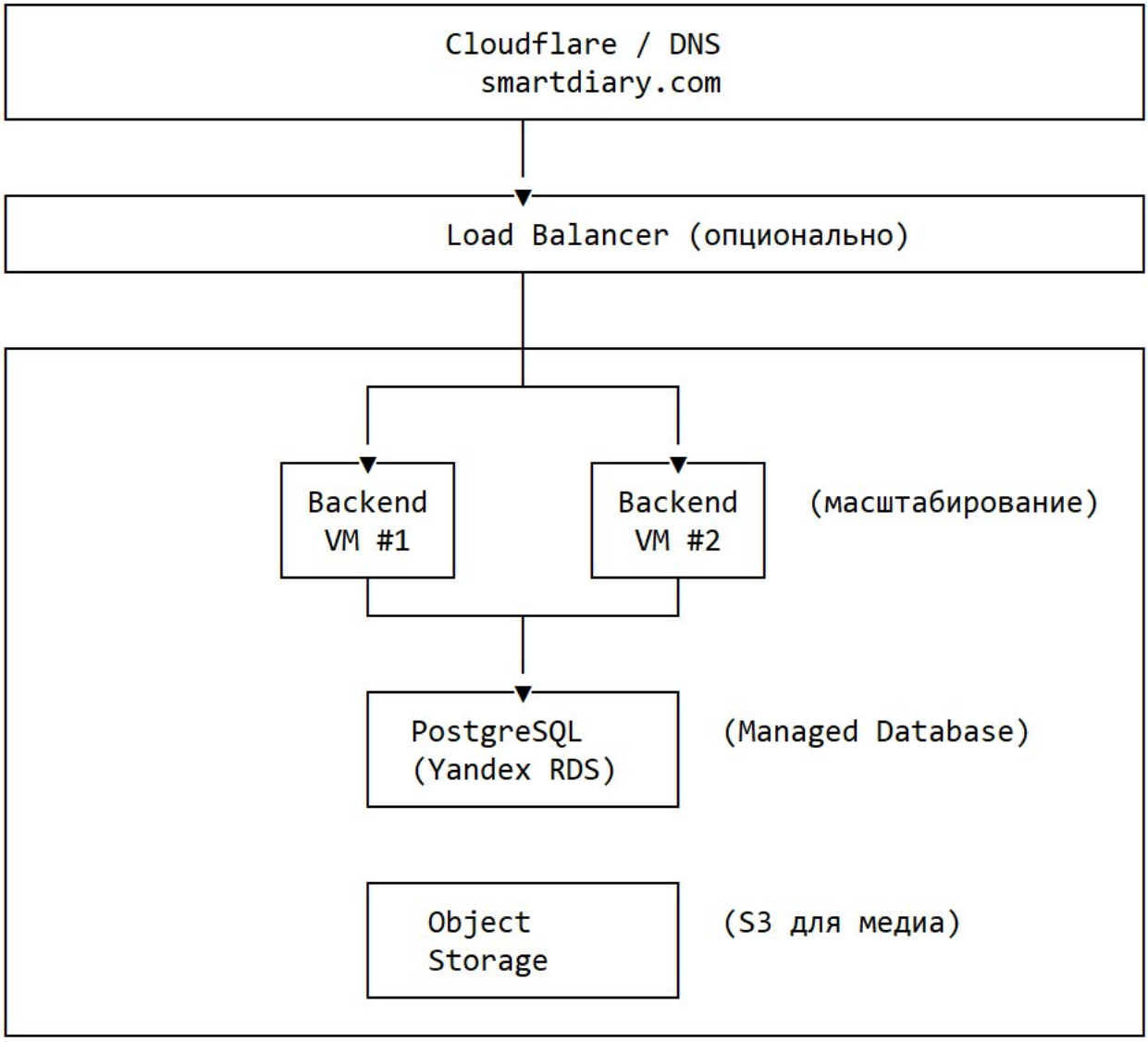
№	Задача
	медиафайлов
5	Настроить домен и SSL-сертификат (HTTPS)
6	Настроить CI/CD для автоматического деплоя

Теоретическая часть

Облачные провайдеры

Провайдер	Бесплатный тариф	Managed БД	Object Storage	Цена
Yandex Cloud	4000₽ на 60 дней	PostgreSQL, Redis	S3	Локальная
AWS	12 месяцев (ограничено)	RDS	S3	Доллары
Render	750 часов/месяц	PostgreSQL	Нет	Простота

Архитектура развертывания



Terraform / Infrastructure as Code

Terraform - инструмент для описания инфраструктуры как код (IaC).

Преимущества:

1. Воспроизводимость окружений
2. Версионирование инфраструктуры
3. Автоматизация создания/удаления ресурсов

Ход выполнения работы

Шаг 1. Настройка Yandex Cloud CLI

```
# Установка Yandex Cloud CLI
```

```
curl -sSL https://storage.yandexcloud.net/yandexcloud-yc/install.sh | bash
source ~/.bashrc

# Аутентификация
yc init

# Создание сервисного аккаунта
yc iam service-account create --name smartdiary-sa

# Создание ключа для сервисного аккаунта
yc iam key create --service-account-name smartdiary-sa --output key.json

# Настройка переменных окружения
export YC_TOKEN=$(yc iam create-token)
export YC_CLOUD_ID=$(yc config get cloud-id)
export YC_FOLDER_ID=$(yc config get folder-id)
```

Шаг 2. Создание Terraform конфигурации

terraform/main.tf:

hcl

```
terraform {
  required_providers {
    yandex = {
      source  = "yandex-cloud/yandex"
      version = "~> 0.87"
    }
  }
}

provider "yandex" {
  token     = var.yc_token
  cloud_id  = var.cloud_id
  folder_id = var.folder_id
  zone      = var.zone
}
```

```
# ============================================
# VPC и сети
# ============================================

resource "yandex_vpc_network" "smartdiary_network" {
  name = "smartdiary-network"
}

resource "yandex_vpc_subnet" "smartdiary_subnet" {
  name           = "smartdiary-subnet"
  zone           = var.zone
  network_id     = yandex_vpc_network.smartdiary_network.id
  v4_cidr_blocks = ["10.0.1.0/24"]
}

# ============================================
# Managed Database PostgreSQL
# ============================================

resource "yandex_mdb_postgresql_cluster" "smartdiary_db" {
  name        = "smartdiary-db"
  environment = "PRODUCTION"
  network_id  = yandex_vpc_network.smartdiary_network.id

  config {
    version = 15
    resources {
      resource_preset_id = "s2.micro"
      disk_type_id       = "network-ssd"
      disk_size          = 20
    }
    postgresql_config = {
      max_connections      = 100
      shared_buffers       = "256MB"
      effective_cache_size = "768MB"
    }
  }

  host {
    zone      = var.zone
    subnet_id = yandex_vpc_subnet.smartdiary_subnet.id
    assign_public_ip = false
  }

  databases {
    name = "smartdiary"
  }
```

```
  user {
    name     = "smartdiary_user"
    password = var.db_password
    permission {
      database_name = "smartdiary"
    }
  }
}

# ============================================
# Object Storage (S3) для медиафайлов
# ============================================

resource "yandex_iam_service_account" "storage_sa" {
  name = "storage-sa"
}

resource "yandex_resourcemanager_folder_iam_member" "storage_sa_editor" {
  folder_id = var.folder_id
  role      = "storage.editor"
  member    = "serviceAccount:${yandex_iam_service_account.storage_sa.id}"
}

resource "yandex_iam_service_account_static_access_key" "storage_sa_key" {
  service_account_id = yandex_iam_service_account.storage_sa.id
}

resource "yandex_storage_bucket" "smartdiary_media" {
  bucket = "smartdiary-media-${var.bucket_suffix}"
  acl    = "public-read"

  anonymous_access_flags {
    read = true
    list = false
  }

  cors_rule {
    allowed_headers = ["*"]
    allowed_methods = ["GET", "PUT", "POST", "DELETE"]
    allowed_origins = ["https://smartdiary.com", "https://www.smartdiary.com"]
    max_age_seconds = 3000
  }

  lifecycle_rule {
    id      = "media-lifecycle"
    enabled = true

    expiration {
```

```
      days = 365
    }
  }
}

# ============================================
# Compute Instance (BM) для Backend
# ============================================

resource "yandex_compute_image" "ubuntu_image" {
  source_family = "ubuntu-2204-lts"
}

resource "yandex_compute_instance" "smartdiary_backend" {
  name        = "smartdiary-backend"
  zone        = var.zone
  platform_id = "standard-v3"

  resources {
    cores  = 2
    memory = 4
  }

  boot_disk {
    initialize_params {
      image_id = yandex_compute_image.ubuntu_image.id
      size     = 30
      type     = "network-ssd"
    }
  }

  network_interface {
    subnet_id = yandex_vpc_subnet.smartdiary_subnet.id
    nat       = true
  }

  metadata = {
    ssh-keys = "ubuntu:${file(var.ssh_public_key)}"
    user-data = templatefile("cloud-init.yml", {
      db_host     = yandex_mdb_postgresql_cluster.smartdiary_db.host[0].fqdn
      db_name     = "smartdiary"
      db_user     = "smartdiary_user"
      db_password = var.db_password
      s3_key      =
yandex_iam_service_account_static_access_key.storage_sa_key.access_key
      s3_secret   =
yandex_iam_service_account_static_access_key.storage_sa_key.secret_key
      bucket_name = yandex_storage_bucket.smartdiary_media.bucket
```

```
      secret_key  = var.app_secret_key
    })
  }
}

# ============================================
# DNS и SSL
# ============================================

resource "yandex_dns_zone" "smartdiary_zone" {
  name   = "smartdiary-zone"
  zone   = "smartdiary.com."
  public = true
}

resource "yandex_dns_recordset" "smartdiary_a" {
  zone_id = yandex_dns_zone.smartdiary_zone.id
  name    = "smartdiary.com."
  type    = "A"
  ttl     = 300
  data    =
[yandex_compute_instance.smartdiary_backend.network_interface[0].nat_ip_address]
}

resource "yandex_dns_recordset" "www_smartdiary_a" {
  zone_id = yandex_dns_zone.smartdiary_zone.id
  name    = "www.smartdiary.com."
  type    = "A"
  ttl     = 300
  data    =
[yandex_compute_instance.smartdiary_backend.network_interface[0].nat_ip_address]
}

# ============================================
# Outputs
# ============================================

output "backend_ip" {
  value =
yandex_compute_instance.smartdiary_backend.network_interface[0].nat_ip_address
}

output "database_host" {
  value = yandex_mdb_postgresql_cluster.smartdiary_db.host[0].fqdn
}

output "bucket_name" {
  value = yandex_storage_bucket.smartdiary_media.bucket
```

```
}

output "s3_access_key" {
 value     =
yandex_iam_service_account_static_access_key.storage_sa_key.access_key
 sensitive = true
}
```

Шаг 3. Cloud-Init конфигурация для ВМ

terraform/cloud-init.yml:

```
#cloud-config
write_files:
  - path: /etc/environment
    content: |
      DJANGO_SECRET_KEY=${secret_key}
      DB_HOST=${db_host}
      DB_NAME=${db_name}
      DB_USER=${db_user}
      DB_PASSWORD=${db_password}
      AWS_ACCESS_KEY_ID=${s3_key}
      AWS_SECRET_ACCESS_KEY=${s3_secret}
      AWS_STORAGE_BUCKET_NAME=${bucket_name}
      DEBUG=False
      ALLOWED_HOSTS=smartdiary.com,www.smartdiary.com,${backend_ip}

  - path: /home/ubuntu/docker compose.yml
    content: |
      version: '3.8'
      services:
        nginx:
          image: nginx:alpine
          container_name: nginx
          restart: always
          volumes:
            - ./nginx.conf:/etc/nginx/conf.d/default.conf:ro
            - ./ssl:/etc/nginx/ssl:ro
          ports:
            - "80:80"
            - "443:443"
          depends_on:
            - backend
            - frontend

        backend:
          image: ghcr.io/username/smartdiary-backend latest
          container_name: backend
```

```
    restart: always
    env_file:
      - .env
    volumes:
      - ./staticfiles:/app/staticfiles
    depends_on:
      - db

  frontend:
    image: ghcr.io/username/smartdiary-frontend:latest
    container_name: frontend
    restart: always

- path: /home/ubuntu/nginx.conf
  content: |
    server {
        listen 80;
        server_name smartdiary.com www.smartdiary.com;
        return 301 https://$server_name$request_uri;
    }

    server {
        listen 443 ssl http2;
        server_name smartdiary.com www.smartdiary.com;

        ssl_certificate /etc/nginx/ssl/fullchain.pem;
        ssl_certificate_key /etc/nginx/ssl/privkey.pem;

        location /api/ {
            proxy_pass http://backend:8080;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
        }

        location /static/ {
            alias /app/staticfiles/;
        }

        location /media/ {
            proxy_pass https://storage.yandexcloud.net/smartdiary-media/;
            proxy_set_header Host storage.yandexcloud.net;
        }

        location / {
            proxy_pass http://frontend:80;
        }
    }
```

```
  - path: /home/ubuntu/.env
    content: |
      SECRET_KEY=${secret_key}
      DB_HOST=${db_host}
      DB_NAME=${db_name}
      DB_USER=${db_user}
      DB_PASSWORD=${db_password}
      AWS_ACCESS_KEY_ID=${s3_key}
      AWS_SECRET_ACCESS_KEY=${s3_secret}
      AWS_STORAGE_BUCKET_NAME=${bucket_name}

  - path: /home/ubuntu/deploy.sh
    content: |
      #!/bin/bash
      cd /home/ubuntu
      docker-compose pull
      docker-compose up -d --force-recreate
      docker system prune -f
    permissions: '0755'

runcmd:
  - apt get update
  - apt get install -y docker.io docker compose nginx certbot python3 certbot nginx
  - systemctl enable docker
  - systemctl start docker
  - cd /home/ubuntu
  - docker login ghcr.io -u username -p ${github_token}
  - ./deploy.sh
  - certbot --nginx -d smartdiary.com -d www.smartdiary.com --non interactive --
agree tos --email admin@smartdiary.com
```

IIIаг 4. Terraform переменные

terraform/variables.tf:

hcl

```
variable "yc_token" {
  description = "Yandex Cloud IAM token"
  type        = string
  sensitive   = true
}

variable "cloud_id" {
  description = "Yandex Cloud ID"
```

```
  type        = string
}

variable "folder_id" {
  description = "Yandex Cloud Folder ID"
  type        = string
}

variable "zone" {
  description = "Availability zone"
  type        = string
  default     = "ru-central1-a"
}

variable "bucket_suffix" {
  description = "Suffix for unique bucket name"
  type        = string
}

variable "db_password" {
  description = "Database password"
  type        = string
  sensitive   = true
}

variable "app_secret_key" {
  description = "Django/Flask secret key"
  type        = string
  sensitive   = true
}

variable "ssh_public_key" {
  description = "Path to SSH public key"
  type        = string
  default     = "~/.ssh/id_rsa.pub"
}

variable "github_token" {
  description = "GitHub token for container registry"
  type        = string
  sensitive   = true
}
```

Шаг 5. Настройка GitHub Actions для деплоя в облако

.github/workflows/deploy-cloud.yml:

yaml

```yaml
name: Deploy to Yandex Cloud

on:
  push:
    branches: [main]
  workflow_dispatch:

jobs:
  deploy:
    name: Deploy to Production
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v3

      - name: Log in to GitHub Container Registry
        uses: docker/login-action@v3
        with:
          registry: ghcr.io
          username: ${{ github.actor }}
          password: ${{ secrets.GITHUB_TOKEN }}

      - name: Build and push Backend
        uses: docker/build-push-action@v5
        with:
          context: ./backend
          push: true
          tags: ghcr.io/${{ github.repository }}/backend:latest

      - name: Build and push Frontend
        uses: docker/build-push-action@v5
        with:
          context: ./frontend
          push: true
          tags: ghcr.io/${{ github.repository }}/frontend:latest

      - name: Deploy to Yandex Cloud
        uses: appleboy/ssh-action@v1.0.0
        with:
          host: ${{ secrets.YC_HOST }}
          username: ubuntu
          key: ${{ secrets.YC_SSH_PRIVATE_KEY }}
          script: |
```

```
cd /home/ubuntu
docker login ghcr.io -u ${{ github.actor }} -p ${{ secrets.GITHUB_TOKEN }}
docker-compose pull
docker-compose up -d --force-recreate
docker system prune -f
```

Шаг 6. Настройка Django для работы с S3

settings.py (дополнение):

```
# Настройка S3 для медиафайлов
if not DEBUG:
    DEFAULT_FILE_STORAGE = 'storages.backends.s3boto3.S3Boto3Storage'
    AWS_ACCESS_KEY_ID = os.environ.get('AWS_ACCESS_KEY_ID')
    AWS_SECRET_ACCESS_KEY = os.environ.get('AWS_SECRET_ACCESS_KEY')
    AWS_STORAGE_BUCKET_NAME = os.environ.get('AWS_STORAGE_BUCKET_NAME')
    AWS_S3_ENDPOINT_URL = 'https://storage.yandexcloud.net'
    AWS_S3_REGION_NAME = 'ru-central1'
    AWS_S3_OBJECT_PARAMETERS = {
        'CacheControl': 'max-age=86400',
    }
    AWS_DEFAULT_ACL = 'public-read'
    AWS_QUERYSTRING_AUTH = False
```

Шаг 7. Установка зависимостей

pip install boto3 django-storages

Шаг 8. AWS альтернатива (Terraform для AWS)

terraform-aws/main.tf:

hcl

```
provider "aws" {
  region = var.aws_region
}

# VPC
resource "aws_vpc" "smartdiary_vpc" {
  cidr_block = "10.0.0.0/16"
  enable_dns_hostnames = true
  enable_dns_support = true
}
```

```
# Public Subnet
resource "aws_subnet" "smartdiary_subnet" {
  vpc_id            = aws_vpc.smartdiary_vpc.id
  cidr_block        = "10.0.1.0/24"
  availability_zone = "${var.aws_region}a"
  map_public_ip_on_launch = true
}

# Internet Gateway
resource "aws_internet_gateway" "smartdiary_igw" {
  vpc_id = aws_vpc.smartdiary_vpc.id
}

# Route Table
resource "aws_route_table" "smartdiary_rt" {
  vpc_id = aws_vpc.smartdiary_vpc.id
  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.smartdiary_igw.id
  }
}

resource "aws_route_table_association" "smartdiary_rta" {
  subnet_id      = aws_subnet.smartdiary_subnet.id
  route_table_id = aws_route_table.smartdiary_rt.id
}

# Security Group
resource "aws_security_group" "smartdiary_sg" {
  name        = "smartdiary-sg"
  vpc_id      = aws_vpc.smartdiary_vpc.id

  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
    description = "SSH"
  }

  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
    description = "HTTP"
  }
```

```
  ingress {
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
    description = "HTTPS"
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

# EC2 Instance
resource "aws_instance" "smartdiary_backend" {
  ami                    = data.aws_ami.ubuntu.id
  instance_type          = "t3.micro"
  subnet_id              = aws_subnet.smartdiary_subnet.id
  vpc_security_group_ids = [aws_security_group.smartdiary_sg.id]
  key_name               = var.key_name

  user_data = templatefile("cloud-init-aws.yml", {
    db_host     = aws_db_instance.smartdiary_db.address
    db_name     = "smartdiary"
    db_user     = "smartdiary_user"
    db_password = var.db_password
  })
}

# RDS PostgreSQL
resource "aws_db_instance" "smartdiary_db" {
  identifier     = "smartdiary-db"
  engine         = "postgres"
  engine_version = "15"
  instance_class = "db.t3.micro"

  allocated_storage     = 20
  storage_type          = "gp3"
  storage_encrypted     = true

  db_name  = "smartdiary"
  username = "smartdiary_user"
  password = var.db_password

  vpc_security_group_ids = [aws_security_group.smartdiary_sg.id]
```

```
  db_subnet_group_name   = aws_db_subnet_group.smartdiary_db_subnet.name

  backup_retention_period = 7
  backup_window          = "03:00-04:00"
  maintenance_window     = "sun:04:00-sun:05:00"

  skip_final_snapshot = true
  publicly_accessible = false
}

# S3 Bucket
resource "aws_s3_bucket" "smartdiary_media" {
  bucket = "smartdiary-media-${var.bucket_suffix}"

  tags = {
    Name = "Smart Diary Media"
  }
}

resource "aws_s3_bucket_public_access_block" "smartdiary_media_block" {
  bucket = aws_s3_bucket.smartdiary_media.id

  block_public_acls       = false
  block_public_policy     = false
  ignore_public_acls      = false
  restrict_public_buckets = false
}

resource "aws_s3_bucket_policy" "smartdiary_media_policy" {
  bucket = aws_s3_bucket.smartdiary_media.id
  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Sid       = "PublicReadGetObject"
        Effect    = "Allow"
        Principal = "*"
        Action    = "s3:GetObject"
        Resource  = "${aws_s3_bucket.smartdiary_media.arn}/*"
      }
    ]
  })
}

# Route 53
data "aws_route53_zone" "domain" {
  name = var.domain_name
}
```

```
resource "aws_route53_record" "smartdiary_a" {
  zone_id = data.aws_route53_zone.domain.zone_id
  name    = var.domain_name
  type    = "A"
  ttl     = 300
  records = [aws_eip.smartdiary_eip.public_ip]
}

# Elastic IP
resource "aws_eip" "smartdiary_eip" {
  instance = aws_instance.smartdiary_backend.id
  domain   = "vpc"
}
```

Шаг 9. Скрипт для проверки развертывания

scripts/verify_deployment.sh:

```
#!/bin/bash

# Цвета для вывода
GREEN='\033[0;32m'
RED='\033[0;31m'
YELLOW='\033[1;33m'
NC='\033[0m'

echo -e "${YELLOW}Проверка развертывания Smart Diary...${NC}\n"

# Проверка 1: Доступность сайта
echo -n "1. Проверка доступности сайта... "
if curl -s -o /dev/null -w "%{http_code}" https://smartdiary.com | grep -q "200\|301\|302"; then
    echo -e "${GREEN}OK${NC}"
else
    echo -e "${RED}FAIL${NC}"
fi
```

```
# Проверка 2: API health
echo -n "2. Проверка API health... "
if curl -s https://smartdiary.com/api/health/ | grep -q "ok"; then
    echo -e "${GREEN}OK${NC}"
else
    echo -e "${RED}FAIL${NC}"
fi

# Проверка 3: База данных
echo -n "3. Проверка подключения к БД... "
if docker exec backend python manage.py check --database default 2>/dev/null; then
    echo -e "${GREEN}OK${NC}"
else
    echo -e "${RED}FAIL${NC}"
fi

# Проверка 4: S3 хранилище
echo -n "4. Проверка S3 хранилища... "
if aws s3 ls s3://smartdiary-media --endpoint-url https://storage.yandexcloud.net
&>/dev/null; then
    echo -e "${GREEN}OK${NC}"
else
    echo -e "${RED}FAIL${NC}"
fi

# Проверка 5: SSL сертификат
echo -n "5. Проверка SSL сертификата... "
if openssl s_client -connect smartdiary.com:443 -servername smartdiary.com
2>/dev/null | openssl x509 -noout -dates | grep -q "notAfter"; then
    echo -e "${GREEN}OK${NC}"
else
```

```
    echo -e "${RED}FAIL${NC}"
fi

# Проверка 6: Docker контейнеры
echo -n "6. Проверка статуса контейнеров... "
RUNNING_COUNT=$(docker ps --format "table {{.Names}}" | grep -c "smartdiary")
if [ $RUNNING_COUNT -ge 3 ]; then
    echo -e "${GREEN}OK${NC} ($RUNNING_COUNT контейнеров)"
else
    echo -e "${RED}FAIL${NC} (только $RUNNING_COUNT контейнеров)"
fi

echo -e "\n${YELLOW}Детали развертывания:${NC}"
echo "  Backend IP: $(yc compute instance get smartdiary-backend --format json | jq -r '.network_interfaces[0].primary_v4_address.one_to_one_nat.address')"
echo "  DB Host: $(yc managed-postgresql cluster get smartdiary-db --format json | jq -r '.hosts[0].fqdn')"
echo "  S3 Bucket: smartdiary-media"
```

Контрольные вопросы

1. Какие преимущества даёт использование managed баз данных?
2. Зачем нужно S3-хранилище для медиафайлов?
3. Как работает процесс получения SSL-сертификата через Let's Encrypt?
4. Что такое Infrastructure as Code и в чём его преимущества?
5. Как настроить CI/CD для автоматического деплоя в облако?
6. Какие метрики нужно мониторить в production-среде?

Критерии оценки

Критерий	Баллы
Создана и настроена ВМ в облаке	2
Настроена managed база данных PostgreSQL	2
Настроено S3-хранилище для медиафайлов	1
Настроен домен и SSL-сертификат (HTTPS)	2
Настроен CI/CD для автоматического деплоя	2
Приложение успешно развернуто и работает	1
Итого	10

Результат выполнения

После выполнения лабораторной работы №23 студенты должны получить:

1. Работающее приложение в облаке (Yandex Cloud / AWS)
2. Managed базу данных PostgreSQL
3. S3-хранилище для медиафайлов
4. HTTPS с валидным SSL-сертификатом
5. Автоматический CI/CD пайплайн для деплоя
6. Документацию по развертыванию

Лабораторная работа №24

Безопасность. Защита от уязвимостей

Цель работы: Аудит безопасности веб-приложения и устранение типовых уязвимостей: SQL-инъекции, XSS, CSRF, CORS, а также настройка безопасности на уровне инфраструктуры.

Задачи

№	Задача
1	Провести статический анализ безопасности кода (SAST)
2	Провести динамический анализ безопасности (DAST) с OWASP ZAP
3	Устранить SQL-инъекции (использовать ORM/параметризованные запросы)
4	Устранить XSS-уязвимости (санитизация вывода)
5	Настроить CSRF-защиту и CORS политики
6	Внедрить rate limiting для защиты от брутфорса и DDoS

Теоретическая часть

OWASP Top 10

Уязвимость	**Описание**	**Рейтинг**
A01:2021 – Broken Access Control	Нарушение контроля доступа	1
A02:2021 – Cryptographic Failures	Криптографические ошибки	2
A03:2021 – Injection	SQL, NoSQL, OS injection	3
A04:2021 – Insecure Design	Небезопасный дизайн	4
A05:2021 – Security	Ошибки конфигурации	5

Уязвимость	Описание	Рейтинг
Misconfiguration		
A06:2021 – Vulnerable Components	Уязвимые компоненты	6
A07:2021 – Identification Failures	Ошибки идентификации	7
A08:2021 – Software Integrity Failures	Нарушение целостности ПО	8
A09:2021 – Monitoring Failures	Отсутствие мониторинга	9
A10:2021 – SSRF	Server-Side Request Forgery	10

SAST vs DAST

Тип	Инструменты	Описание
SAST	SonarQube, Bandit, Semgrep	Анализ исходного кода
DAST	OWASP ZAP, Burp Suite	Анализ работающего приложения

Ход выполнения работы

Шаг 1. Установка инструментов безопасности

```
# SAST для Python/Django
pip install bandit safety
```

```
# SAST для JavaScript
npm install -D eslint-plugin-security

# SAST для .NET
dotnet tool install --global SecurityCodeScan.Tool

# DAST (OWASP ZAP)
# Скачать с https://www.zaproxy.org/download.html
```

Шаг 2. Настройка защиты от SQL-инъекций

Использование ORM вместо сырых SQL-запросов:

```
# ✕ НЕПРАВИЛЬНО: риск SQL-инъекции
def get_task_unsafe(request, task_id):
    query = f"SELECT * FROM tasks_task WHERE id = {task_id}"
    # ...

# ✓ ПРАВИЛЬНО: использование ORM
def get_task_safe(request, task_id):
    task = Task.objects.filter(id=task_id).first()
    # ...

# ✕ НЕПРАВИЛЬНО: конкатенация в сыром запросе
def search_unsafe(request):
    search = request.GET.get('search')
    cursor.execute(f"SELECT * FROM tasks_task WHERE title LIKE '%{search}%'")

# ✓ ПРАВИЛЬНО: параметризованный запрос
def search_safe(request):
    search = request.GET.get('search')
    cursor.execute("SELECT * FROM tasks_task WHERE title LIKE %s", [f'%{search}%'])
```

Шаг 3. Настройка защиты от XSS

Django (автоматическая защита):

```
# ✓ Django автоматически экранирует вывод в шаблонах
```

```
# {{ user_input }}  ->  <script>alert(1)</script>  экранируется


# ⚠ Осторожно с |safe
# {{ user_input|safe }}  # Только для доверенного контента


# ✅ Для разрешённого HTML используйте bleach
```

pip install bleach

```
import bleach

ALLOWED_TAGS = ['b', 'i', 'u', 'strong', 'em', 'p', 'br', 'ul', 'ol', 'li']
ALLOWED_ATTRIBUTES = {'a': ['href', 'title']}

def clean_html_content(content):
    return bleach.clean(content, tags=ALLOWED_TAGS,
attributes=ALLOWED_ATTRIBUTES)
```

React (автоматическая защита):

jsx

```
// ✅ React автоматически экранирует значения
<div>{userInput}</div>

// ⚠ Осторожно с dangerouslySetInnerHTML
<div dangerouslySetInnerHTML={{__html: sanitizedHtml}} />
```

Шаг 4. Настройка CSRF-защиты

Django:

```
# settings.py
CSRF_COOKIE_SECURE = True
CSRF_COOKIE_HTTPONLY = True
CSRF_COOKIE_SAMESITE = 'Strict'
CSRF_TRUSTED_ORIGINS = ['https://smartdiary.com',
'https://www.smartdiary.com']

# В шаблонах
<form method="post">
    {% csrf_token %}
    ...
</form>
```

```
# Для API с JWT
# JWT не требует CSRF, так как токен хранится в заголовке
```
React + Django:
```
javascript
// Настройка axios для передачи CSRF токена
apiClient.interceptors.request.use(config => {
    const csrfToken = document.cookie
        .split('; ')
        .find(row => row.startsWith('csrftoken='))
        ?.split('=')[1];

    if (csrfToken) {
        config.headers['X-CSRFToken'] = csrfToken;
    }
    return config;
});
```
Шаг 5. Настройка CORS

Django:

```
# settings.py
INSTALLED_APPS = [
    'corsheaders',
]

MIDDLEWARE = [
    'corsheaders.middleware.CorsMiddleware',
]

CORS_ALLOWED_ORIGINS = [
    'https://smartdiary.com',
    'https://www.smartdiary.com',
]

CORS_ALLOW_CREDENTIALS = True

CORS_ALLOW_METHODS = [
    'GET', 'POST', 'PUT', 'PATCH', 'DELETE', 'OPTIONS'
]

CORS_ALLOW_HEADERS = [
    'accept',
    'accept-encoding',
    'authorization',
    'content-type',
    'x-csrftoken',
```

```
]
```

Шаг 6. Настройка Rate Limiting

Django (django-ratelimit):

```
pip install django-ratelimit
```

```
# settings.py
RATELIMIT_ENABLE = True
RATELIMIT_USE_CACHE = 'default'
RATELIMIT_CACHE = 'default'

# В представлениях
from django_ratelimit.decorators import ratelimit

@ratelimit(key='ip', rate='5/m', method='POST', block=True)
def login_view(request):
    # Ограничение: 5 попыток в минуту с одного IP
    pass

@ratelimit(key='user', rate='100/h', method='POST', block=True)
def create_task(request):
    # Ограничение: 100 задач в час от пользователя
    pass
```

Nginx:

```
# Ограничение частоты запросов
limit_req_zone $binary_remote_addr zone=api:10m rate=10r/s;

server {
    location /api/ {
        limit_req zone=api burst=20 nodelay;
        limit_req_status 429;
        proxy_pass http://backend:8000;
    }
}
```

Шаг 7. Настройка Security Headers

Django (django-csp и django-security):

```
pip install django-csp django-security
```

```
# settings.py
# HSTS
SECURE_HSTS_SECONDS = 31536000  # 1 год
SECURE_HSTS_INCLUDE_SUBDOMAINS = True
SECURE_HSTS_PRELOAD = True

# HTTPS
SECURE_SSL_REDIRECT = True
SECURE_PROXY_SSL_HEADER = ('HTTP_X_FORWARDED_PROTO', 'https')

# Cookies
SESSION_COOKIE_SECURE = True
SESSION_COOKIE_HTTPONLY = True
SESSION_COOKIE_SAMESITE = 'Strict'

CSRF_COOKIE_SECURE = True
CSRF_COOKIE_HTTPONLY = True
CSRF_COOKIE_SAMESITE = 'Strict'

# Content Security Policy (CSP)
CSP_DEFAULT_SRC = ("'self'",)
CSP_SCRIPT_SRC = ("'self'", "'unsafe-inline'", "https://cdn.jsdelivr.net")
CSP_STYLE_SRC = ("'self'", "'unsafe-inline'", "https://fonts.googleapis.com")
CSP_IMG_SRC = ("'self'", "data:", "https://storage.yandexcloud.net")

# X-Frame-Options
X_FRAME_OPTIONS = 'DENY'

# Referrer Policy
SECURE_REFERRER_POLICY = 'strict-origin-when-cross-origin'
```

Шаг 8. Защита паролей

```
# settings.py
# Использование сильного хэширования паролей
PASSWORD_HASHERS = [
    'django.contrib.auth.hashers.Argon2PasswordHasher',
    'django.contrib.auth.hashers.PBKDF2PasswordHasher',
    'django.contrib.auth.hashers.PBKDF2SHA1PasswordHasher',
]

# Требования к паролю
AUTH_PASSWORD_VALIDATORS = [
```

```
        {
            'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
        },
        {
            'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
            'OPTIONS': {
                'min_length': 8,
            }
        },
        {
            'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
        },
        {
            'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
        },
    ]
```

Шаг 9. Безопасная работа с JWT

```
# settings.py
from datetime import timedelta

SIMPLE_JWT = {
    'ACCESS_TOKEN_LIFETIME': timedelta(minutes=15),
    'REFRESH_TOKEN_LIFETIME': timedelta(days=7),
    'ROTATE_REFRESH_TOKENS': True,
    'BLACKLIST_AFTER_ROTATION': True,
    'ALGORITHM': 'HS256',
    'SIGNING_KEY': SECRET_KEY,
    'VERIFYING_KEY': None,
    'AUTH_HEADER_TYPES': ('Bearer',),
    'USER_ID_FIELD': 'id',
    'USER_ID_CLAIM': 'user_id',
    'AUTH_TOKEN_CLASSES': ('rest_framework_simplejwt.tokens.AccessToken',),
    'TOKEN_TYPE_CLAIM': 'token_type',
}
```

Шаг 10. Безопасность файловых загрузок

```
# validation.py
import magic
from django core exceptions import ValidationError

ALLOWED_MIME_TYPES = [
    'image/jpeg', 'image/png', 'image/gif', 'image/webp',
    'application/pdf',
]
```

```
ALLOWED_EXTENSIONS = ['.jpg', '.jpeg', '.png', '.gif', '.webp', '.pdf']
MAX_FILE_SIZE = 5 * 1024 * 1024  # 5 MB

def validate_file(file):
    # Проверка размера
    if file.size > MAX_FILE_SIZE:
        raise ValidationError(f'Файл слишком большой. Максимум {MAX_FILE_SIZE // 1024 // 1024} MB')

    # Проверка MIME-типа (библиотека magic)
    mime = magic.from_buffer(file.read(1024), mime=True)
    if mime not in ALLOWED_MIME_TYPES:
        raise ValidationError(f'Недопустимый тип файла')

    # Проверка расширения
    ext = os.path.splitext(file.name)[1].lower()
    if ext not in ALLOWED_EXTENSIONS:
        raise ValidationError(f'Недопустимое расширение файла')

    return file
```

Шаг 11. SAST с Bandit (Python)

```
# Установка
pip install bandit

# Запуск проверки
bandit -r ./tasks -f html -o bandit_report.html

# Конфигурация .bandit
.bandit:

yaml
exclude: tests, migrations
skips:
  - B311  # random module usage
  - B601  # paramiko calls
```

Шаг 12. SAST с ESLint Security (JavaScript)

.eslintrc.js:
javascript

```
module exports = {
  extends: [
    'plugin:security/recommended'
  ],
  plugins: [
    'security'
  ],
  rules: {
    'security/detect-object-injection': 'off',
    'security/detect-non-literal-fs-filename': 'warn',
    'security/detect-eval-with-expression': 'error',
    'security/detect-unsafe-regex': 'warn',
  }
};
```

Шаг 13. DAST с OWASP ZAP (автоматизация)

zap_scan.py:

```
from zapv2 import ZAPv2
import time
import json

ZAP_ADDRESS = 'localhost'
ZAP_PORT = 8080
API_KEY = 'your-api-key'
TARGET_URL = 'http://localhost:8000'

zap = ZAPv2(apikey=API_KEY, proxies={'http': f'http://{ZAP_ADDRESS}:{ZAP_PORT}',
'https': f'http://{ZAP_ADDRESS}:{ZAP_PORT}'})

# Spider (обход сайта)
print("Запуск обхода (spider)...")
spider_id = zap.spider.scan(TARGET_URL)

while int(zap.spider.status(spider_id)) < 100:
    print(f"Прогресс обхода: {zap.spider.status(spider_id)}%")
    time.sleep(2)

print("Spider завершён")
```

```
# Активное сканирование
print("Запуск активного сканирования...")
scan_id = zap.ascan.scan(TARGET_URL)

while int(zap.ascan.status(scan_id)) < 100:
    print(f"Прогресс сканирования: {zap.ascan.status(scan_id)}%")
    time.sleep(5)

print("Сканирование завершено")

# Получение результатов
alerts = zap.core.alerts()
with open('zap_report.json', 'w') as f:
    json.dump(alerts, f, indent=2)

print(f"Найдено уязвимостей: {len(alerts)}")
for alert in alerts:
    print(f"- {alert['risk']}: {alert['name']} ({alert['url']})")
```

Шаг 14. SAST с SonarQube (Docker)

```
# Запуск SonarQube
docker run -d --name sonarqube -p 9000:9000 sonarqube:latest

# Установка SonarScanner
dotnet tool install --global dotnet-sonarscanner

# Для Python
sonar-scanner \
  -Dsonar.projectKey=smartdiary \
  -Dsonar.sources=. \
  -Dsonar.host.url=http://localhost:9000 \
  -Dsonar.login=your-token
```

Шаг 15. Аудит зависимостей

```
bash
# Python
```

```
safety check --full-report

# Node.js
npm audit
npm audit fix

# .NET
dotnet list package --vulnerable
```

Шаг 16. Защита от брутфорса (Django Axes)

bash

```
pip install django-axes
```

python

```
# settings.py
INSTALLED_APPS = [
    'axes',
]

MIDDLEWARE = [
    'axes.middleware.AxesMiddleware',
]

AUTHENTICATION_BACKENDS = [
    'axes.backends.AxesBackend',
    'django.contrib.auth.backends.ModelBackend',
]

AXES_FAILURE_LIMIT = 5
AXES_COOLOFF_TIME = 1  # час
AXES_LOCK_OUT_AT_FAILURE = True
AXES_RESET_ON_SUCCESS = True
```

```
AXES_ONLY_USER_FAILURES = False
AXES_LOCKOUT_TEMPLATE = 'registration/locked_out.html'
```

Шаг 17. Настройка логирования безопасности

```
# settings.py
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'handlers': {
        'security_file': {
            'class': 'logging.handlers.RotatingFileHandler',
            'filename': '/var/log/smartdiary/security.log',
            'maxBytes': 10485760,
            'backupCount': 10,
        },
    },
    'loggers': {
        'django.security': {
            'handlers': ['security_file'],
            'level': 'INFO',
            'propagate': False,
        },
        'axes.watch_login': {
            'handlers': ['security_file'],
            'level': 'INFO',
            'propagate': False,
        },
    },
}
```

Шаг 18. Docker Security Best Practices

Dockerfile (безопасный):

dockerfile

```
# Использование непривилегированного пользователя
FROM python:3.11-slim

# Создание непривилегированного пользователя
RUN addgroup --system --gid 1001 appgroup && \
    adduser --system --uid 1001 --gid 1001 appuser
```

```
# Установка рабочей директории с правильными правами
WORKDIR /app
RUN chown -R appuser:appgroup /app

# Копирование и установка зависимостей
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Копирование кода
COPY --chown=appuser:appgroup . .

# Переключение на непривилегированного пользователя
USER appuser

EXPOSE 8000
CMD ["gunicorn", "smartdiary.wsgi:application", "--bind", "0.0.0.0:8000"]
```

Контрольные вопросы

1. Что такое OWASP Top 10 и какие уязвимости в него входят?
2. Чем отличаются SAST и DAST?
3. Как защититься от SQL-инъекций в Django/ASP.NET?
4. Как работает CSRF-защита и когда она отключается?
5. Зачем нужен CSP (Content Security Policy)?
6. Как настроить rate limiting для защиты от DDoS?
7. Какие best practices для безопасного хранения паролей?

Критерии оценки

Критерий	Баллы
Проведен SAST анализ (SonarQube/Bandit)	2
Проведен DAST анализ (OWASP ZAP)	1
Устранены SQL-инъекции (использование ORM)	2
Устранены XSS-уязвимости (санитизация)	1
Настроена CSRF-защита и CORS	1
Внедрен rate limiting	1
Настроены security headers (HSTS, CSP, X-Frame-Options)	1
Предоставлен отчёт по безопасности	1
Итого	10

Результат выполнения

После выполнения лабораторной работы №24 студенты должны получить:

1. Отчёт SAST-анализа (Bandit/SonarQube)
2. Отчёт DAST-анализа (OWASP ZAP)
3. Защищённое приложение от SQL-инъекций и XSS
4. Настроенные security headers (HSTS, CSP, X-Frame-Options)
5. Rate limiting для защиты от брутфорса
6. Документацию по безопасности приложения

Лабораторная работа №25

Мобильная адаптация и UI/UX улучшения

Цель работы: Адаптация интерфейса веб-приложения для мобильных устройств, улучшение пользовательского опыта (UX), внедрение темной темы и оптимизация touch-взаимодействия.

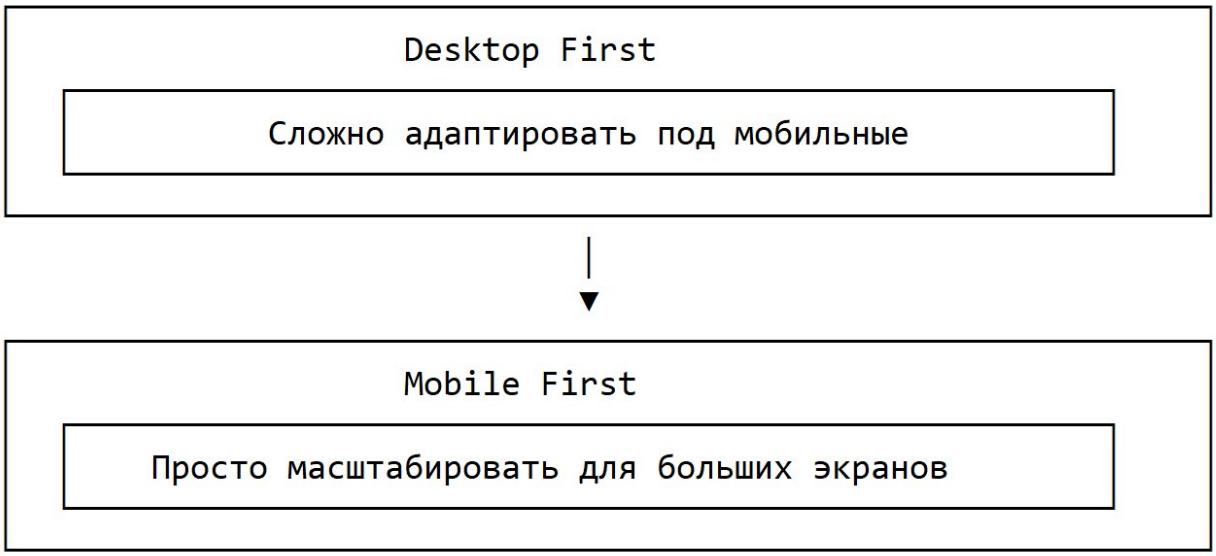
Задачи

№	Задача
1	Реализовать адаптивный дизайн (mobile-first подход)
2	Добавить мобильное меню (бургер-меню) для навигации
3	Реализовать темную тему (dark mode) с сохранением выбора пользователя
4	Оптимизировать touch-события для мобильных устройств (свайпы, жесты)
5	Провести UX-тестирование и улучшить интерфейс на основе результатов

Теоретическая часть

Mobile-First подход

Mobile-First - подход к разработке, при котором сначала создается дизайн для мобильных устройств, а затем масштабируется для десктопа.



CSS Media Queries

css

```
/* Мобильные устройства (< 600px) */
@media (max-width: 600px) { ... }

/* Планшеты (600px - 900px) */
@media (min-width: 600px) and (max-width: 900px) { ... }

/* Десктоп (> 900px) */
@media (min-width: 900px) { ... }
```

Touch-события

Событие	**Описание**
touchstart	Начало касания
touchmove	Движение пальцем
touchend	Окончание касания

Событие	Описание
touchcancel	Прерывание касания

Ход выполнения работы

Шаг 1. Настройка мета-тегов для мобильных устройств

index.html (дополнение):

html

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0, viewport-fit=cover, user-scalable=yes">
    <meta name="theme-color" content="#2c3e50">
    <meta name="apple-mobile-web-app-capable" content="yes">
    <meta name="apple-mobile-web-app-status-bar-style" content="black-translucent">
    <meta name="format-detection" content="telephone=no">
    <meta name="HandheldFriendly" content="true">
</head>
<body>
    ...
</body>
</html>
```

Шаг 2. Адаптивные стили (CSS)

src/styles/responsive.css:

css

```
/* ==========================================================
```

```
   Mobile-First стили (базовые)
   ========================================================== */

/* Базовые стили для мобильных */
.container {
    padding: 0 16px;
    width: 100%;
    max-width: 100%;
}

/* Карточки задач на мобильных */
.task-card {
    margin: 8px 0;
    border-radius: 12px;
}

.task-card .task-title {
    font-size: 16px;
}

/* ==========================================================
   Планшеты (600px - 900px)
   ========================================================== */
@media (min-width: 600px) {
    .container {
        max-width: 540px;
        margin: 0 auto;
        padding: 0 24px;
    }
```

```
    .tasks-grid {
        display: grid;
        grid-template-columns: repeat(2, 1fr);
        gap: 16px;
    }
}

/* ================================================
   Десктоп (> 900px)
   ================================================ */
@media (min-width: 900px) {
    .container {
        max-width: 1200px;
        padding: 0 32px;
    }

    .tasks-grid {
        grid-template-columns: repeat(3, 1fr);
        gap: 24px;
    }

    .sidebar {
        position: sticky;
        top: 80px;
        width: 280px;
    }

    .main-layout {
        display: flex;
        gap: 32px;
```

```
    }
}

/* ================================================
   Крупные десктопы (> 1400px)
   ================================================ */
@media (min-width: 1400px) {
    .container {
        max-width: 1400px;
    }

    .tasks-grid {
        grid-template-columns: repeat(4, 1fr);
    }
}
```

Шаг 3. Адаптивный хук для React

src/hooks/useMediaQuery.js:

javascript

```
import { useState, useEffect } from 'react';

export const useMediaQuery = (query) => {
    const [matches, setMatches] = useState(false);

    useEffect(() => {
        const media = window.matchMedia(query);
        if (media.matches !== matches) {
            setMatches(media.matches);
        }

        const listener = (event) => setMatches(event.matches);
```

```
        media.addEventListener('change', listener);

        return () => media.removeEventListener('change', listener);
      }, [matches, query]);

      return matches;
    };

    // Предопределенные хуки
    export const useIsMobile = () => useMediaQuery('(max-width: 600px)');
    export const useIsTablet = () => useMediaQuery('(min-width: 600px) and (max-width: 900px)');
    export const useIsDesktop = () => useMediaQuery('(min-width: 900px)');
```

Шаг 4. Компонент бургер-меню

src/components/common/MobileMenu.jsx:

javascript

```
    import React, { useState, useEffect } from 'react';
    import {
      Drawer,
      IconButton,
      List,
      ListItem,
      ListItemIcon,
      ListItemText,
      Box,
      AppBar,
      Toolbar,
      Typography,
    } from '@mui/material';
    import {
```

```
    Menu as MenuIcon,
    Dashboard as DashboardIcon,
    Assignment as TasksIcon,
    CalendarToday as CalendarIcon,
    ViewColumn as KanbanIcon,
    BarChart as StatsIcon,
    Person as ProfileIcon,
    Settings as SettingsIcon,
    Logout as LogoutIcon,
    Close as CloseIcon,
} from '@mui/icons-material';
import { useNavigate } from 'react-router-dom';
import { useIsMobile } from '../../hooks/useMediaQuery';

const menuItems = [
    { text: 'Панель управления', icon: <DashboardIcon />, path: '/' },
    { text: 'Задачи', icon: <TasksIcon />, path: '/tasks' },
    { text: 'Календарь', icon: <CalendarIcon />, path: '/calendar' },
    { text: 'Доска', icon: <KanbanIcon />, path: '/kanban' },
    { text: 'Статистика', icon: <StatsIcon />, path: '/statistics' },
    { text: 'Профиль', icon: <ProfileIcon />, path: '/profile' },
    { text: 'Настройки', icon: <SettingsIcon />, path: '/settings' },
];

const MobileMenu = ({ user, onLogout }) => {
    const [drawerOpen, setDrawerOpen] = useState(false);
    const navigate = useNavigate();
    const isMobile = useIsMobile();

    const handleNavigate = (path) => {
```

```
    navigate(path);
    setDrawerOpen(false);
  };

  const handleLogout = () => {
    onLogout();
    setDrawerOpen(false);
  };

  return (
    <>
      <AppBar position="sticky" sx={{ display: { xs: 'block', md: 'none' } }}>
        <Toolbar>
          <IconButton
            edge="start"
            color="inherit"
            onClick={() => setDrawerOpen(true)}
            sx={{ mr: 2 }}
          >
            <MenuIcon />
          </IconButton>
          <Typography variant="h6" component="div" sx={{ flexGrow: 1 }}>
            Smart Diary
          </Typography>
        </Toolbar>
      </AppBar>

      <Drawer
```

```
      anchor="left"
      open={drawerOpen}
      onClose={() => setDrawerOpen(false)}
      PaperProps={{
        sx: { width: '80%', maxWidth: 300 },
      }}
    >
      <Box sx={{ p: 2, display: 'flex', justifyContent: 'flex-end' }}>
        <IconButton onClick={() => setDrawerOpen(false)}>
          <CloseIcon />
        </IconButton>
      </Box>

      <List>
        {menuItems.map((item) => (
          <ListItem
            button
            key={item.text}
            onClick={() => handleNavigate(item.path)}
            sx={{
              borderRadius: 2,
              mx: 1,
              mb: 0.5,
            }}
          >
            <ListItemIcon>{item.icon}</ListItemIcon>
            <ListItemText primary={item.text} />
          </ListItem>
        ))}
```

```
                <ListItem button onClick={handleLogout} sx={{ color: '#e74c3c' }}>
                  <ListItemIcon sx={{ color: '#e74c3c' }}>
                    <LogoutIcon />
                  </ListItemIcon>
                  <ListItemText primary="Выйти" />
                </ListItem>
              </List>

              {user && (
                <Box sx={{ p: 2, mt: 'auto', borderTop: '1px solid #eee' }}>
                  <Typography variant="body2" color="text.secondary">
                    {user.username}
                  </Typography>
                  <Typography variant="caption" color="text.secondary">
                    {user.email}
                  </Typography>
                </Box>
              )}
            </Drawer>
          </>
        );
      };

      export default MobileMenu;
```

Шаг 5. Темная тема (Dark Mode)

src/theme/theme.js:

javascript

```
import { createTheme } from '@mui/material/styles';
```

```
// Светлая тема
export const lightTheme = createTheme({
  palette: {
    mode: 'light',
    primary: {
      main: '#2c3e50',
    },
    secondary: {
      main: '#3498db',
    },
    background: {
      default: '#f5f7fa',
      paper: '#ffffff',
    },
    text: {
      primary: '#2c3e50',
      secondary: '#7f8c8d',
    },
  },
  shape: {
    borderRadius: 12,
  },
});

// Темная тема
export const darkTheme = createTheme({
  palette: {
    mode: 'dark',
    primary: {
      main: '#3498db',
```

```
    },
    secondary: {
      main: '#2ecc71',
    },
    background: {
      default: '#1a1a2e',
      paper: '#16213e',
    },
    text: {
      primary: '#ffffff',
      secondary: '#a0a0a0',
    },
  },
  shape: {
    borderRadius: 12,
  },
});

// Сепия тема (для accessibility)
export const sepiaTheme = createTheme({
  palette: {
    mode: 'light',
    primary: {
      main: '#8B6914',
    },
    secondary: {
      main: '#C4A35A',
    },
    background: {
      default: '#F4ECD8',
```

```
      paper: '#FBF4DF',
    },
    text: {
      primary: '#5C4033',
      secondary: '#8B7355',
    },
  },
});
```

src/context/ThemeContext.jsx:

javascript

```
import React, { createContext, useContext, useState, useEffect } from 'react';
import { ThemeProvider as MuiThemeProvider } from '@mui/material/styles';
import CssBaseline from '@mui/material/CssBaseline';
import { lightTheme, darkTheme, sepiaTheme } from '../theme/theme';

const themes = {
  light: lightTheme,
  dark: darkTheme,
  sepia: sepiaTheme,
};

const ThemeContext = createContext();

export const useTheme = () => useContext(ThemeContext);

export const ThemeProvider = ({ children }) => {
  const [themeMode, setThemeMode] = useState(() => {
    return localStorage.getItem('theme-mode') || 'light';
  });
```

```
    const [theme, setTheme] = useState(themes[themeMode]);

    useEffect(() => {
        setTheme(themes[themeMode]);
        localStorage.setItem('theme-mode', themeMode);

        // Обновление theme-color для PWA
        const metaThemeColor = document.querySelector('meta[name="theme-color"]');
        if (metaThemeColor) {
            const colors = {
                light: '#2c3e50',
                dark: '#1a1a2e',
                sepia: '#8B6914',
            };
            metaThemeColor.setAttribute('content', colors[themeMode]);
        }
    }, [themeMode]);

    const toggleTheme = () => {
        setThemeMode(prev => {
            if (prev === 'light') return 'dark';
            if (prev === 'dark') return 'sepia';
            return 'light';
        });
    };

    return (
```

```
            <ThemeContext Provider value={{ themeMode, toggleTheme, setThemeMode }}>
                <MuiThemeProvider theme={theme}>
                  <CssBaseline />
                  {children}
                </MuiThemeProvider>
              </ThemeContext Provider>
            );
          };
```

src/components/common/ThemeToggle.jsx:

javascript

```
import React from 'react';
import { IconButton, Tooltip, Menu, MenuItem, ListItemIcon, ListItemText } from '@mui/material';
import {
  Brightness4 as DarkIcon,
  Brightness7 as LightIcon,
  Palette as SepiaIcon,
} from '@mui/icons-material';
import { useTheme } from '../../context/ThemeContext';

const ThemeToggle = () => {
  const { themeMode, setThemeMode, toggleTheme } = useTheme();
  const [anchorEl, setAnchorEl] = React.useState(null);

  const handleClick = (event) => {
    setAnchorEl(event.currentTarget);
  };

  const handleClose = () => {
```

```
    setAnchorEl(null);
  };

  const handleThemeChange = (mode) => {
    setThemeMode(mode);
    handleClose();
  };

  const getIcon = () => {
    switch (themeMode) {
      case 'light': return <LightIcon />;
      case 'dark': return <DarkIcon />;
      case 'sepia': return <SepiaIcon />;
      default: return <LightIcon />;
    }
  };

  return (
    <>
      <Tooltip title="Сменить тему">
        <IconButton onClick={handleClick} color="inherit">
          {getIcon()}
        </IconButton>
      </Tooltip>

      <Menu anchorEl={anchorEl} open={Boolean(anchorEl)}
onClose={handleClose}>
        <MenuItem onClick={() => handleThemeChange('light')}>
          <ListItemIcon><LightIcon /></ListItemIcon>
          <ListItemText>Светлая</ListItemText>
```

```
            </MenuItem>
            <MenuItem onClick={() => handleThemeChange('dark')}>
              <ListItemIcon><DarkIcon /></ListItemIcon>
              <ListItemText>Тёмная</ListItemText>
            </MenuItem>
            <MenuItem onClick={() => handleThemeChange('sepia')}>
              <ListItemIcon><SepiaIcon /></ListItemIcon>
              <ListItemText>Сепия</ListItemText>
            </MenuItem>
          </Menu>
        </>
      );
    };

    export default ThemeToggle;
```

Шаг 6. Swipe-жесты для мобильных устройств

src/hooks/useSwipe.js:

javascript

```
import { useState, useEffect } from 'react';

export const useSwipe = (onSwipeLeft, onSwipeRight, onSwipeUp, onSwipeDown, threshold = 50) => {
  const [touchStart, setTouchStart] = useState(null);
  const [touchEnd, setTouchEnd] = useState(null);

  useEffect(() => {
    const handleTouchStart = (e) => {
      setTouchEnd(null);
      setTouchStart(e.touches[0]);
    };
```

```
    const handleTouchMove = (e) => {
      setTouchEnd(e.touches[0]);
    };

    const handleTouchEnd = () => {
      if (!touchStart || !touchEnd) return;

      const distanceX = touchStart.clientX - touchEnd.clientX;
      const distanceY = touchStart.clientY - touchEnd.clientY;

      if (Math.abs(distanceX) > Math.abs(distanceY) &&
Math.abs(distanceX) > threshold) {
        if (distanceX > 0) {
          onSwipeLeft?.();
        } else {
          onSwipeRight?.();
        }
      } else if (Math.abs(distanceY) > threshold) {
        if (distanceY > 0) {
          onSwipeUp?.();
        } else {
          onSwipeDown?.();
        }
      }
    };

    document.addEventListener('touchstart', handleTouchStart);
    document.addEventListener('touchmove', handleTouchMove);
    document.addEventListener('touchend', handleTouchEnd);
```

```
        return () => {
            document.removeEventListener('touchstart', handleTouchStart);
            document.removeEventListener('touchmove', handleTouchMove);
            document.removeEventListener('touchend', handleTouchEnd);
        };
    }, [touchStart, touchEnd, onSwipeLeft, onSwipeRight, onSwipeUp, onSwipeDown, threshold]);
};

// Хук для свайпа по задачам (архивация/удаление)
export const useTaskSwipe = (taskId, onArchive, onDelete) => {
    const [swiped, setSwiped] = useState(false);

    const handleSwipeLeft = () => {
        setSwiped(true);
        setTimeout(() => {
            onArchive?.(taskId);
            setSwiped(false);
        }, 300);
    };

    const handleSwipeRight = () => {
        setSwiped(true);
        setTimeout(() => {
            onDelete?.(taskId);
            setSwiped(false);
        }, 300);
    };
```

```
    useSwipe(handleSwipeLeft, handleSwipeRight);

    return { swiped };
  };
```

Шаг 7. Swipe-компонент для задач на мобильных

src/components/tasks/SwipeableTaskCard.jsx:

javascript

```
import React, { useState } from 'react';
import { motion, AnimatePresence } from 'framer-motion';
import { Box, Typography, Chip, Checkbox, IconButton } from '@mui/material';
import { Delete as DeleteIcon, Archive as ArchiveIcon } from '@mui/icons-material';
import { useSwipe } from '../../hooks/useSwipe';

const SwipeableTaskCard = ({ task, onStatusChange, onDelete, onArchive }) => {
  const [swipeDirection, setSwipeDirection] = useState(null);
  const [showActions, setShowActions] = useState(false);

  const handleSwipeLeft = () => {
    setSwipeDirection('left');
    setShowActions(true);
    setTimeout(() => {
      onArchive?.(task.id);
      setShowActions(false);
      setSwipeDirection(null);
    }, 300);
  };
```

```
const handleSwipeRight = () => {
    setSwipeDirection('right');
    setShowActions(true);
    setTimeout(() => {
        onDelete?.(task.id);
        setShowActions(false);
        setSwipeDirection(null);
    }, 300);
};

useSwipe(handleSwipeLeft, handleSwipeRight);

const priorityColors = {
    Low: '#2ecc71',
    Medium: '#f39c12',
    High: '#e74c3c',
};

return (
    <Box sx={{ position: 'relative', overflow: 'hidden', mb: 2 }}>
        {/* Фоновые действия (свайп) */}
        <AnimatePresence>
            {showActions && (
                <Box
                    sx={{
                        position: 'absolute',
                        right: swipeDirection === 'left' ? 0 : 'auto',
                        left: swipeDirection === 'right' ? 0 : 'auto',
                        top: 0,
                        bottom: 0,
```

```
                        width: 80,
                        bgcolor: swipeDirection === 'left' ? '#2ecc71' : '#e74c3c',
                        display: 'flex',
                        alignItems: 'center',
                        justifyContent: 'center',
                        borderRadius: 2,
                      }}
                    >
                      {swipeDirection === 'left' ? <ArchiveIcon /> : <DeleteIcon />}
                    </Box>
                  )}
                </AnimatePresence>

                {/* Карточка задачи */}
                <motion.div
                  initial={{ x: 0 }}
                  animate={{ x: swipeDirection === 'left' ? -80 : swipeDirection === 'right' ? 80 : 0 }}
                  transition={{ type: 'spring', stiffness: 300, damping: 30 }}
                  style={{
                    backgroundColor: 'white',
                    borderRadius: 12,
                    padding: 12,
                    boxShadow: '0 2px 8px rgba(0,0,0,0.1)',
                  }}
                >
                  <Box display="flex" alignItems="center">
                    <Checkbox
                      checked={task.status === 'Completed'}
```

```
                onChange={(e) => onStatusChange?.(task.id,
e.target.checked ? 'Completed' : 'New')}
                  size="small"
                />
                <Typography
                  variant="body1"
                  sx={{
                    flex: 1,
                    textDecoration: task.status === 'Completed' ? 'line-
through' : 'none',
                    opacity: task.status === 'Completed' ? 0.7 : 1,
                    fontSize: { xs: 14, sm: 16 },
                  }}
                >
                  {task.title}
                </Typography>
              </Box>

              <Box display="flex" flexWrap="wrap" gap={0.5} ml={5} mt={1}>
                <Chip
                  size="small"
                  label={task.priority}
                  sx={{
                    bgcolor: priorityColors[task.priority],
                    color: 'white',
                    height: 20,
                    fontSize: '0.7rem',
                  }}
                />
                {task.deadline && (
```

```
                        <Chip
                            size="small"
                        label={new Date(task deadline).toLocaleDateString('ru-RU')}
                            variant="outlined"
                            sx={{ height: 20, fontSize: '0.7rem' }}
                        />
                    )}
                </Box>
            </motion div>
        </Box>
    );
};

export default SwipeableTaskCard;
```

Шаг 8. Адаптивная навигация

src/components/layout/ResponsiveLayout.jsx:

javascript

```
import React from 'react';
import { Box, Container, Drawer } from '@mui/material';
import { useIsMobile, useIsDesktop } from '../../hooks/useMediaQuery';
import MobileMenu from '../common/MobileMenu';
import Sidebar from '../common/Sidebar';
import Navbar from '../common/Navbar';

const ResponsiveLayout = ({ children }) => {
    const isMobile = useIsMobile();
    const isDesktop = useIsDesktop();
    const [mobileDrawerOpen, setMobileDrawerOpen] = React.useState(false);
```

```
if (isMobile) {
  return (
    <Box sx={{ pb: 7 }}>
      <MobileMenu />
      <Container sx={{ px: 2, py: 2 }}>
        {children}
      </Container>
      {/* Нижняя навигационная панель для мобильных */}
      <BottomNavigation />
    </Box>
  );
}

if (isDesktop) {
  return (
    <Box display="flex">
      <Sidebar />
      <Box component="main" sx={{ flexGrow: 1, p: 3 }}>
        {children}
      </Box>
    </Box>
  );
}

// Планшеты
return (
  <>
    <Navbar />
    <Container maxWidth="md" sx={{ py: 3 }}>
      {children}
```

```
        </Container>
      </>
    );
};

const BottomNavigation = () => {
    const navigate = useNavigate();
    const [value, setValue] = React.useState(0);

    const menuItems = [
        { icon: <DashboardIcon />, label: 'Главная', path: '/' },
        { icon: <TasksIcon />, label: 'Задачи', path: '/tasks' },
        { icon: <AddIcon />, label: 'Новая', path: '/tasks/create', special: true },
        { icon: <CalendarIcon />, label: 'Календарь', path: '/calendar' },
        { icon: <PersonIcon />, label: 'Профиль', path: '/profile' },
    ];

    return (
        <Paper
            elevation={3}
            sx={{
                position: 'fixed',
                bottom: 0,
                left: 0,
                right: 0,
                borderRadius: 0,
                zIndex: 1100,
            }}
        >
            <BottomNavigation
```

```
      showLabels
      value={value}
      onChange={(_, newValue) => {
        setValue(newValue);
        navigate(menuItems[newValue].path);
      }}
    >
      {menuItems.map((item, idx) => (
        <BottomNavigationAction
          key={idx}
          icon={
            item.special ? (
              <Box
                sx={{
                  bgcolor: 'primary.main',
                  color: 'white',
                  borderRadius: '50%',
                  width: 48,
                  height: 48,
                  display: 'flex',
                  alignItems: 'center',
                  justifyContent: 'center',
                  mt: -2,
                  boxShadow: 3,
                }}
              >
                {item.icon}
              </Box>
            ) : (
              item.icon
```

```
            )
          }
            label={item.label}
          />
        ))}
      </BottomNavigation>
    </Paper>
  );
};
```

Шаг 9. UI/UX улучшения

src/components/feedback/LoadingSkeleton.jsx:

javascript

```
import React from 'react';
import { Skeleton, Box, Card, CardContent } from '@mui/material';

const TaskCardSkeleton = () => (
  <Card sx={{ mb: 2 }}>
    <CardContent>
      <Box display="flex" alignItems="center" gap={2}>
        <Skeleton variant="circular" width={24} height={24} />
        <Skeleton variant="text" width="60%" height={32} />
      </Box>
      <Box display="flex" gap={1} mt={2}>
        <Skeleton variant="rectangular" width={60} height={24}
sx={{ borderRadius: 2 }} />
        <Skeleton variant="rectangular" width={80} height={24}
sx={{ borderRadius: 2 }} />
      </Box>
    </CardContent>
  </Card>
```

```
  );

  const LoadingSkeleton = () => (
    <Box>
      {[1, 2, 3, 4].map(i => (
        <TaskCardSkeleton key={i} />
      ))}
    </Box>
  );

  export default LoadingSkeleton;
```

src/components/feedback/EmptyState.jsx:

javascript

```
import React from 'react';
import { Box, Typography, Button } from '@mui/material';
import { Add as AddIcon, Assignment as TasksIcon } from '@mui/icons-material';

const EmptyState = ({ title, description, actionText, onAction, icon }) => {
  const IconComponent = icon || TasksIcon;

  return (
    <Box
      sx={{
        display: 'flex',
        flexDirection: 'column',
        alignItems: 'center',
        justifyContent: 'center',
        textAlign: 'center',
        py: 8,
```

```
            px: 2,
          }}
        >
          <IconComponent sx={{ fontSize: 64, color: 'text.secondary', mb: 2 }}
/>

          <Typography variant="h6" gutterBottom>
            {title || 'Нет данных'}
          </Typography>

          <Typography variant="body2" color="text.secondary" sx={{ mb: 3,
maxWidth: 300 }}>
            {description || 'Здесь пока ничего нет'}
          </Typography>

          {actionText && (
            <Button variant="contained" startIcon={<AddIcon />}
onClick={onAction}>
              {actionText}
            </Button>
          )}
        </Box>
      );
    };

    export default EmptyState;
```

Шаг 10. Адаптация React компонентов для мобильных

src/components/tasks/TaskList.jsx (адаптированный):

javascript

```
import React from 'react';
```

```
import { List, ListItem, Box, Typography, useMediaQuery } from '@mui/material';
import TaskCard from './TaskCard';
import SwipeableTaskCard from './SwipeableTaskCard';
import LoadingSkeleton from '../feedback/LoadingSkeleton';
import EmptyState from '../feedback/EmptyState';

const TaskList = ({ tasks, loading, onStatusChange, onDelete, onArchive }) => {
  const isMobile = useMediaQuery('(max-width: 600px)');

  if (loading) return <LoadingSkeleton />;

  if (!tasks || tasks.length === 0) {
    return (
      <EmptyState
        title="Нет задач"
        description="Создайте свою первую задачу, нажав на кнопку +"
        actionText="Создать задачу"
        onAction={() => navigate('/tasks/create')}
      />
    );
  }

  return (
    <List sx={{ p: 0 }}>
      {tasks.map((task) => (
        <ListItem key={task.id} sx={{ p: 0 }}>
          {isMobile ? (
            <SwipeableTaskCard
```

```
                task={task}
                onStatusChange={onStatusChange}
                onDelete={onDelete}
                onArchive={onArchive}
              />
            ) : (
              <TaskCard
                task={task}
                onStatusChange={onStatusChange}
                onDelete={onDelete}
              />
            )}
          </ListItem>
        ))}
      </List>
    );
};
```

Контрольные вопросы

1. В чём суть Mobile-First подхода в веб-разработке?
2. Какие мета-теги важны для мобильной адаптации?
3. Как работают CSS Media Queries?
4. Как реализовать бургер-меню в React?
5. Как организовать переключение тем (светлая/тёмная)?
6. Как обрабатывать touch-события и свайпы?

Критерии оценки

Критерий	Баллы
Реализован адаптивный дизайн (mobile-first)	2
Реализовано бургер-меню для мобильных устройств	2

Критерий	Баллы
Реализована темная тема с сохранением выбора	2
Реализованы swipe-жесты для мобильных	2
Проведено UX-тестирование и улучшения	1
Код работает без ошибок	1
Итого	10

Результат выполнения

После выполнения лабораторной работы №25 студенты должны получить:

1. Адаптивный интерфейс для всех типов устройств
2. Мобильное меню (бургер-меню)
3. Тёмную тему (dark/light/sepia) с сохранением выбора
4. Swipe-жесты для мобильных устройств
5. Улучшенный пользовательский опыт (скелетоны, пустые состояния)

Лабораторная работа №26

Итоговое тестирование и приемка

Цель работы: Комплексное тестирование всего приложения: E2E-тестирование ключевых пользовательских сценариев, нагрузочное тестирование, составление отчета о качестве и подготовка к финальной приемке.

Задачи

№	Задача
1	Написать E2E-тесты для ключевых пользовательских сценариев (Cypress / Playwright)
2	Провести нагрузочное тестирование (k6 / JMeter)
3	Составить план тестирования и отчет о тестировании

№	Задача
4	Провести smoke-тестирование после деплоя
5	Провести user acceptance testing (UAT)

Теоретическая часть

Виды тестирования

Вид	Описание	Инструменты
E2E	Сквозное тестирование полных сценариев пользователя	Cypress, Playwright
Нагрузочное	Проверка производительности под нагрузкой	k6, JMeter
Smoke	Базовая проверка работоспособности после деплоя	curl, health checks
UAT	Приемочное тестирование пользователями	Ручное / автоматизированное

Пирамида тестирования для проекта

text

```
   E2E      ← 5% (Cypress, Playwright)
  ──────
 Интегра-    ← 20% (pytest, xUnit)
 ционные
 ──────────
┌──────────┐
│ Модульные │  ← 75% (pytest, Jest, xUnit)
│  (unit)   │
└──────────┘
```

Ход выполнения работы

Шаг 1. Установка инструментов E2E-тестирования

bash

```
# Cypress
npm install -D cypress @testing-library/cypress

# Playwright
npm init playwright@latest

# k6 (нагрузочное тестирование)
# macOS
brew install k6

# Linux
```

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys C5AD17C747E3415A3642D57D77C6C491D6AC1D69
echo "deb https://dl.k6.io/deb stable main" | sudo tee /etc/apt/sources.list.d/k6.list
sudo apt-get update
sudo apt-get install k6

# Windows
choco install k6
```

IIIаг 2. Настройка Cypress

cypress.config.js:

```
javascript
const { defineConfig } = require('cypress');

module.exports = defineConfig({
  e2e: {
    baseUrl: 'http://localhost:3000',
    supportFile: 'cypress/support/e2e.js',
    specPattern: 'cypress/e2e/**/*.cy.{js,jsx,ts,tsx}',
    viewportWidth: 1280,
    viewportHeight: 720,
    video: true,
    screenshotOnRunFailure: true,
    defaultCommandTimeout: 10000,
    retries: {
      runMode: 2,
      openMode: 0,
    },
  },
  component: {
    devServer: {
      framework: 'react',
      bundler: 'vite',
    },
  },
});
```

cypress/support/commands.js:

```
javascript
// Кастомные команды
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();
  cy.url().should('include', '/tasks');
  cy.window().its('localStorage').invoke('getItem', 'access_token').should('exist');
});

Cypress.Commands.add('createTask', (title, priority = 'Medium') => {
  cy.get('[data-testid="create-task-button"]').click();
  cy.get('[data-testid="task-title-input"]').type(title);
  cy.get(`[data-testid="priority-${priority}"]`).click();
  cy.get('[data-testid="submit-task-button"]').click();
```

```
    cy.contains(title).should('be.visible');
});

Cypress Commands.add('deleteTask', (title) => {
    cy.contains(title).parents('[data-testid="task-card"]').within(() => {
        cy.get('[data-testid="delete-task-button"]').click();
    });
    cy.contains(title).should('not.exist');
});

Cypress Commands.add('logout', () => {
    cy.get('[data-testid="user-menu"]').click();
    cy.get('[data-testid="logout-button"]').click();
    cy.url().should('include', '/login');
});
```

Шаг 3. E2E-тесты (Cypress)

cypress/e2e/auth.cy.js:

javascript

```
/// <reference types="cypress" />

describe('Аутентификация', () => {
    const testUser = {
        username: `testuser_${Date.now()}`,
        email: `test_${Date.now()}@example.com`,
        password: 'Test123456!'
    };

    beforeEach(() => {
        cy.visit('/');
    });

    it('Пользователь должен иметь возможность зарегистрироваться', () => {
        cy.get('[data-testid="register-link"]').click();
        cy.url().should('include', '/register');

        cy.get('[data-testid="username-input"]').type(testUser.username);
        cy.get('[data-testid="email-input"]').type(testUser.email);
        cy.get('[data-testid="password-input"]').type(testUser.password);
        cy.get('[data-testid="confirm-password-input"]').type(testUser.password);
        cy.get('[data-testid="register-button"]').click();

        cy.url().should('include', '/tasks');
        cy.window().its('localStorage').invoke('getItem', 'access_token').should('exist');
    });

    it('Пользователь должен иметь возможность войти', () => {
        cy.login(testUser.username, testUser.password);
    });

    it('Пользователь должен видеть ошибку при неверных данных', () => {
        cy.get('[data-testid="username-input"]').type('wrong_user');
        cy.get('[data-testid="password-input"]').type('wrong_password');
```

```
        cy.get('[data-testid="login-button"]').click();

        cy.get('[data-testid="error-message"]').should('be.visible');
        cy.url().should('include', '/login');
    });

    it('Пользователь должен иметь возможность выйти', () => {
        cy.login(testUser.username, testUser.password);
        cy.logout();
    });
});
```

cypress/e2e/tasks.cy.js:

javascript

```
/// <reference types="cypress" />

describe('Управление задачами', () => {
    const testUser = {
        username: `taskuser_${Date.now()}`,
        email: `task_${Date.now()}@example.com`,
        password: 'Test123456!'
    };

    beforeEach(() => {
        cy.request('POST', 'http://localhost:8000/api/auth/register/', testUser);
        cy.login(testUser.username, testUser.password);
    });

    it('Пользователь должен создавать задачу', () => {
        cy.createTask('Тестовая задача', 'High');
        cy.contains('Тестовая задача').should('be.visible');
        cy.contains('High').should('be.visible');
    });

    it('Пользователь должен редактировать задачу', () => {
        const taskTitle = 'Задача для редактирования';
        const updatedTitle = 'Обновленная задача';

        cy.createTask(taskTitle, 'Medium');

        cy.contains(taskTitle).parents('[data-testid="task-card"]').within(() => {
            cy.get('[data-testid="edit-task-button"]').click();
        });

        cy.get('[data-testid="task-title-input"]').clear().type(updatedTitle);
        cy.get('[data-testid="submit-task-button"]').click();

        cy.contains(updatedTitle).should('be.visible');
        cy.contains(taskTitle).should('not.exist');
    });

    it('Пользователь должен удалять задачу', () => {
        cy.createTask('Задача для удаления', 'Low');
```

```
    cy.deleteTask('Задача для удаления');
  });

  it('Пользователь должен отмечать задачу выполненной', () => {
    cy.createTask('Задача для выполнения', 'High');

    cy.contains('Задача для выполнения').parents('[data-testid="task-card"]').within(() =>
{
      cy.get('[data-testid="task-complete-checkbox"]').check();
    });

    cy.contains('Задача для выполнения')
      .parents('[data-testid="task-card"]')
      .should('have.class', 'completed');
  });

  it('Пользователь должен фильтровать задачи по статусу', () => {
    cy.createTask('Активная задача', 'High');
    cy.createTask('Выполненная задача', 'Medium');

    // Отметим задачу как выполненную
    cy.contains('Выполненная задача').parents('[data-testid="task-card"]').within(() =>
{
      cy.get('[data-testid="task-complete-checkbox"]').check();
    });

    // Фильтр по активным задачам
    cy.get('[data-testid="status-filter"]').select('New,InProgress');
    cy.contains('Активная задача').should('be.visible');
    cy.contains('Выполненная задача').should('not.exist');
  });

  it('Пользователь должен искать задачи', () => {
    cy.createTask('Купить продукты', 'Medium');
    cy.createTask('Сделать отчет', 'High');

    cy.get('[data-testid="search-input"]').type('продукты');
    cy.get('[data-testid="search-button"]').click();

    cy.contains('Купить продукты').should('be.visible');
    cy.contains('Сделать отчет').should('not.exist');
  });
});
```

cypress/e2e/projects.cy.js:

```
javascript
/// <reference types="cypress" />

describe('Управление проектами', () => {
  const testUser = {
    username: `projuser_${Date.now()}`,
    email: `proj_${Date.now()}@example.com`,
    password: 'Test123456!'
```

```
  };

  beforeEach(() => {
    cy.request('POST', 'http://localhost:8000/api/auth/register/', testUser);
    cy.login(testUser.username, testUser.password);
  });

  it('Пользователь должен создавать проект', () => {
    cy.get('[data-testid="projects-link"]').click();
    cy.get('[data-testid="create-project-button"]').click();
    cy.get('[data-testid="project-name-input"]').type('Рабочий проект');
    cy.get('[data-testid="project-color"]').click();
    cy.get('[data-testid="color-blue"]').click();
    cy.get('[data-testid="submit-project-button"]').click();

    cy.contains('Рабочий проект').should('be.visible');
  });
});
```

Шаг 4. Нагрузочное тестирование с k6

k6/load-test.js:

javascript

```
import http from 'k6/http';
import { check, sleep } from 'k6';
import { Counter, Rate, Trend } from 'k6/metrics';

// Метрики
const requests = new Counter('requests');
const errors = new Rate('errors');
const responseTime = new Trend('response_time');

export const options = {
  stages: [
    { duration: '30s', target: 20 },   // Разогрев до 20 пользователей
    { duration: '1m', target: 50 },    // Подъем до 50 пользователей
    { duration: '2m', target: 100 },   // Пик до 100 пользователей
    { duration: '1m', target: 50 },    // Спад до 50
    { duration: '30s', target: 0 },    // Завершение
  ],
  thresholds: {
    http_req_duration: ['p(95)<500'],  // 95% запросов менее 500мс
    http_req_failed: ['rate<0.01'],    // Менее 1% ошибок
    errors: ['rate<0.05'],             // Менее 5% ошибок
  },
};

const BASE_URL = __ENV.BASE_URL || 'http://localhost:8000';

// Тестовый пользователь
const testUser = {
  username: `loadtest_${Date.now()}`,
  email: `loadtest_${Date.now()}@example.com`,
  password: 'LoadTest123!',
```

```
};

export function setup() {
  // Регистрация тестового пользователя
  const registerRes = http.post(`${BASE_URL}/api/auth/register/`, JSON.stringify({
    username: testUser.username,
    email: testUser.email,
    password: testUser.password,
    password2: testUser.password,
  }), {
    headers: { 'Content-Type': 'application/json' },
  });

  // Получение токена
  const loginRes = http.post(`${BASE_URL}/api/auth/login/`, JSON.stringify({
    username: testUser.username,
    password: testUser.password,
  }), {
    headers: { 'Content-Type': 'application/json' },
  });

  const token = loginRes.json('access');
  return { token };
}

export default function (data) {
  const headers = {
    'Content-Type': 'application/json',
    'Authorization': `Bearer ${data.token}`,
  };

  // GET /api/tasks/
  let tasksRes = http.get(`${BASE_URL}/api/tasks/`, { headers });
  check(tasksRes, {
    'tasks status is 200': (r) => r.status === 200,
  });
  requests.add(1);
  responseTime.add(tasksRes.timings.duration);
  errors.add(tasksRes.status !== 200);

  sleep(1);

  // POST /api/tasks/
  let createRes = http.post(`${BASE_URL}/api/tasks/`, JSON.stringify({
    title: `Тестовая задача ${__VU}-${__ITER}`,
    description: 'Создано нагрузочным тестированием',
    status: 'New',
    priority: 'Medium',
  }), { headers });

  check(createRes, {
    'create task status is 201': (r) => r.status === 201,
```

```
  });

  sleep(0.5);

  if (createRes.status === 201) {
    const taskId = createRes.json('id');

    // GET /api/tasks/{id}/
    let detailRes = http.get(`${BASE_URL}/api/tasks/${taskId}/`, { headers });
    check(detailRes, {
      'task detail status is 200': (r) => r.status === 200,
    });

    sleep(0.5);

    // PUT /api/tasks/{id}/
    let updateRes = http.put(`${BASE_URL}/api/tasks/${taskId}/`, JSON.stringify({
      title: `Обновленная задача ${__VU}-${__ITER}`,
      description: 'Обновлено нагрузочным тестированием',
      status: 'InProgress',
      priority: 'High',
    }), { headers });

    check(updateRes, {
      'update task status is 200': (r) => r.status === 200,
    });

    sleep(0.5);

    // DELETE /api/tasks/{id}/
    let deleteRes = http.del(`${BASE_URL}/api/tasks/${taskId}/`, null, { headers });
    check(deleteRes, {
      'delete task status is 204': (r) => r.status === 204,
    });
  }

  // GET /api/statistics/
  let statsRes = http.get(`${BASE_URL}/api/statistics/`, { headers });
  check(statsRes, {
    'statistics status is 200': (r) => r.status === 200,
  });

  sleep(1);
}

export function teardown(data) {
  // Очистка
  console.log('Test completed');
}
```

Шаг 5. Запуск тестов

```
bash
# Cypress в headless режиме
```

```
npx cypress run

# Cypress с UI (для отладки)
npx cypress open

# Нагрузочное тестирование
k6 run k6/load-test.js

# Нагрузочное тестирование с выводом результатов в JSON
k6 run --out json=k6-results.json k6/load-test.js

# Нагрузочное тестирование с кастомными параметрами
k6 run -e BASE_URL=https://smartdiary.com k6/load-test.js
```

Шаг 6. Smoke-тестирование после деплоя

scripts/smoke-test.sh:

bash

```
#!/bin/bash

# Цвета
GREEN='\033[0;32m'
RED='\033[0;31m'
YELLOW='\033[1;33m'
NC='\033[0m'

BASE_URL=${1:-"http://localhost:8000"}
FAILED=0

echo -e "${YELLOW}=== Smoke Test for Smart Diary ===${NC}\n"

# Функция проверки
check() {
    local name=$1
    local url=$2
    local expected_status=${3:-200}

    echo -n "Проверка: $name... "

    status=$(curl -s -o /dev/null -w "%{http_code}" "$url")

    if [ "$status" -eq "$expected_status" ]; then
        echo -e "${GREEN}OK${NC} ($status)"
    else
        echo -e "${RED}FAIL${NC} (ожидалось $expected_status, получено $status)"
        FAILED=$((FAILED + 1))
    fi
}

# Health check
check "Health check" "$BASE_URL/api/health/"

# API документация
check "Swagger UI" "$BASE_URL/api/schema/swagger-ui/" 200
```

```
# Аутентификация
check "Регистрация" "$BASE_URL/api/auth/register/" 405  # POST, но проверяем доступность

# Задачи
check "Список задач" "$BASE_URL/api/tasks/" 401  # Требует аутентификации

# Статические файлы
check "Static files" "$BASE_URL/static/admin/css/base.css" 200

# Frontend
check "Frontend index" "$BASE_URL/" 200

echo ""

if [ $FAILED -eq 0 ]; then
    echo -e "${GREEN}=== Все проверки пройдены успешно ===${NC}"
    exit 0
else
    echo -e "${RED}=== Не пройдено проверок: $FAILED ===${NC}"
    exit 1
fi
```

Шаг 7. План тестирования

docs/TestPlan.md:

markdown

```
# План тестирования Smart Diary

## 1. Цели тестирования

- Проверить функциональность приложения
- Оценить производительность под нагрузкой
- Убедиться в безопасности приложения
- Подтвердить готовность к production

## 2. Объем тестирования

### 2.1 Функциональное тестирование

| Модуль | Кейсы |
|--------|-------|
| Аутентификация | Регистрация, вход, выход, восстановление пароля |
| Задачи | CRUD, фильтрация, поиск, статусы, приоритеты |
| Проекты | CRUD, участники, роли |
| Теги | CRUD, привязка к задачам |
| Комментарии | Создание, редактирование, удаление, ответы |
| Статистика | Отображение метрик, графики |
| Экспорт | CSV, JSON, PDF |
| Календарь | Отображение, drag-and-drop |
| Kanban | Перетаскивание между колонками |

### 2.2 Нагрузочное тестирование
```

Сценарий	Количество пользователей	Длительность
Постепенный рост	20 → 100	5 минут
Пиковая нагрузка	100	2 минуты
Длительная работа	50	15 минут

2.3 Тестирование безопасности

Уязвимость	Статус
SQL-инъекции	✅ Проверено
XSS	✅ Проверено
CSRF	✅ Настроено
JWT безопасность	✅ Проверено

3. Окружение тестирования

- **Тестовое окружение**: staging.smartdiary.com
- **Производственное окружение**: smartdiary.com
- **Браузеры**: Chrome, Firefox, Safari, Edge
- **Устройства**: Desktop, Tablet, Mobile

4. График тестирования

Этап	Длительность	Ответственный
Модульное тестирование	Постоянно	Разработчики
Интеграционное тестирование	1 день	QA
E2E тестирование	2 дня	QA
Нагрузочное тестирование	1 день	DevOps
UAT	3 дня	Заказчик
Регрессионное тестирование	1 день	QA

5. Критерии приемки

- ✅ Все E2E тесты проходят
- ✅ Покрытие кода > 70%
- ✅ Время ответа API < 500мс (p95)
- ❌ Критических уязвимостей нет
- ✅ Продукт принят заказчиком

Шаг 8. Отчет о тестировании

docs/TestReport.md:

markdown

Отчет о тестировании Smart Diary

1. Общая информация

- **Проект**: Smart Diary
- **Версия**: 1.0.0
- **Дата тестирования**: [дата]

- **Тестировщик**: [Имя]

2. Результаты тестирования

2.1 Модульное тестирование

Компонент	Количество тестов	Пройдено	Провалено	Покрытие
Backend (Tasks)	45	45	0	85%
Backend (Auth)	23	23	0	92%
Frontend (Components)	38	37	1	78%
Frontend (Redux)	15	15	0	88%
Итого	**121**	**120**	**1**	**84%**

2.2 E2E тестирование (Cypress)

Сценарий	Результат	Время выполнения
Регистрация нового пользователя	✔ PASS	2.3s
Вход в систему	✔ PASS	1.8s
Создание задачи	✔ PASS	3.1s
Редактирование задачи	✔ PASS	2.9s
Удаление задачи	✔ PASS	2.5s
Фильтрация задач	✔ PASS	3.4s
Поиск задач	✔ PASS	2.7s
Создание проекта	✔ PASS	2.2s
Добавление комментария	✔ PASS	2.8s
Экспорт в CSV	✔ PASS	1.5s
Итого	**10/10 PASS**	**25.2s**

2.3 Нагрузочное тестирование (k6)

Метрика	Порог	Фактическое значение	Статус
p95 время ответа	< 500ms	342ms	✔
p99 время ответа	< 1000ms	678ms	✔
Ошибки	< 1%	0.2%	✔
RPS	—	156 req/s	ℹ

2.4 Тестирование безопасности (OWASP ZAP)

Уровень риска	Количество	Устранено
High	0	0
Medium	2	2
Low	5	3

2.5 Кросс-браузерное тестирование

Браузер	Версия	Статус

Chrome	120+	✅
Firefox	115+	✅
Safari	16+	✅
Edge	120+	✅

2.6 Мобильное тестирование

Устройство	Экран	ОС	Статус
iPhone 14	6.1"	iOS 16	✅
iPhone SE	4.7"	iOS 15	✅
Samsung S23	6.1"	Android 13	✅
Google Pixel 6	6.4"	Android 13	✅

3. Найденные проблемы и их статус

ID	Описание	Серьезность	Статус
BUG-001	Ошибка валидации даты на фронтенде	Medium	✅ Исправлено
BUG-002	Медленная загрузка списка задач при 500+ записях	Medium	✅ Оптимизировано
BUG-003	Проблема с отображением тегов в Safari	Low	✅ Исправлено

4. Рекомендации

1. Увеличить кэширование для API эндпоинтов со статистикой
2. Добавить больше E2E тестов для edge-кейсов
3. Оптимизировать загрузку бандла фронтенда

5. Заключение

Приложение **Smart Diary** успешно прошло полный цикл тестирования:

- ✅ Функциональность соответствует требованиям
- ✅ Производительность соответствует порогам
- ✅ Безопасность на приемлемом уровне
- ✅ UI/UX соответствует ожиданиям

Рекомендация: Приложение готово к production-развертыванию.

Шаг 9. User Acceptance Testing (UAT) чек-лист

docs/UAT-Checklist.md:

markdown

User Acceptance Testing (UAT) Чек-лист

Информация о тестировании
- **Дата**: [дата]
- **Тестировщик**: [Имя пользователя]
- **Окружение**: [production/staging]

Сценарии тестирования

1. Регистрация и вход
- [ ] Регистрация нового пользователя
- [ ] Вход с корректными данными
- [ ] Ошибка при неверном пароле
- [ ] Восстановление пароля
- [ ] Выход из системы

2. Управление задачами
- [ ] Создание задачи с заполнением всех полей
- [ ] Редактирование задачи
- [ ] Удаление задачи
- [ ] Отметка задачи выполненной
- [ ] Изменение статуса задачи
- [ ] Изменение приоритета задачи

3. Проекты и теги
- [ ] Создание проекта
- [ ] Редактирование проекта
- [ ] Удаление проекта
- [ ] Привязка задачи к проекту
- [ ] Создание тега
- [ ] Привязка тега к задаче

4. Поиск и фильтрация
- [ ] Поиск по заголовку
- [ ] Фильтрация по статусу
- [ ] Фильтрация по приоритету
- [ ] Фильтрация по проекту
- [ ] Комбинированная фильтрация

5. Календарь и доска
- [ ] Просмотр задач в календаре
- [ ] Drag-and-drop для изменения даты
- [ ] Просмотр Kanban доски
- [ ] Перетаскивание карточек между колонками

6. Мобильное приложение
- [ ] Адаптивный интерфейс на телефоне
- [ ] Бургер-меню работает
- [ ] Свайп-жесты работают
- [ ] Установка PWA

7. Обратная связь
- [ ] Комментарии к задачам
- [ ] Уведомления о дедлайнах
- [ ] Экспорт отчета в PDF

8. Общее впечатление
- [ ] Приложение работает быстро
- [ ] Дизайн современный и понятный
- [ ] Нет критических ошибок

Заключение
- [ ] Принято
- [ ] Требуются доработки

Подпись пользователя: _____
Дата: _____

Контрольные вопросы

1. Какие виды тестирования используются в проекте?
2. Для чего нужны E2E-тесты и чем они отличаются от модульных?
3. Как проводится нагрузочное тестирование с k6?
4. Что такое smoke-тестирование и когда оно применяется?
5. Как составляется план тестирования?
6. Какие критерии приемки программного продукта?

Критерии оценки

Критерий	Баллы
Написаны E2E-тесты (Cypress/Playwright)	2
Проведено нагрузочное тестирование (k6)	2
Составлен план тестирования	2
Составлен отчет о тестировании	2
Проведено smoke-тестирование	1
Проведено UAT и составлен чек-лист	1
Итого	10

Результат выполнения

После выполнения лабораторной работы №26 студенты должны получить:

1. Набор E2E-тестов для ключевых пользовательских сценариев
2. Результаты нагрузочного тестирования с k6
3. План тестирования с графиком и критериями
4. Отчет о тестировании с найденными проблемами
5. Smoke-тесты для проверки деплоя
6. UAT чек-лист для приемки заказчиком

Лабораторная работа №27

Финальная защита проекта

Цель работы: Итоговая презентация и защита сквозного проекта «Умный ежедневник» с демонстрацией всех функций, архитектурных решений, CI/CD пайплайна, мониторинга и результатов тестирования.

Задачи

№	Задача
1	Подготовить финальную презентацию проекта (20-25 слайдов)
2	Продемонстрировать работающее приложение в облаке
3	Представить архитектуру и технологические решения
4	Продемонстрировать CI/CD пайплайн и мониторинг
5	Ответить на вопросы комиссии

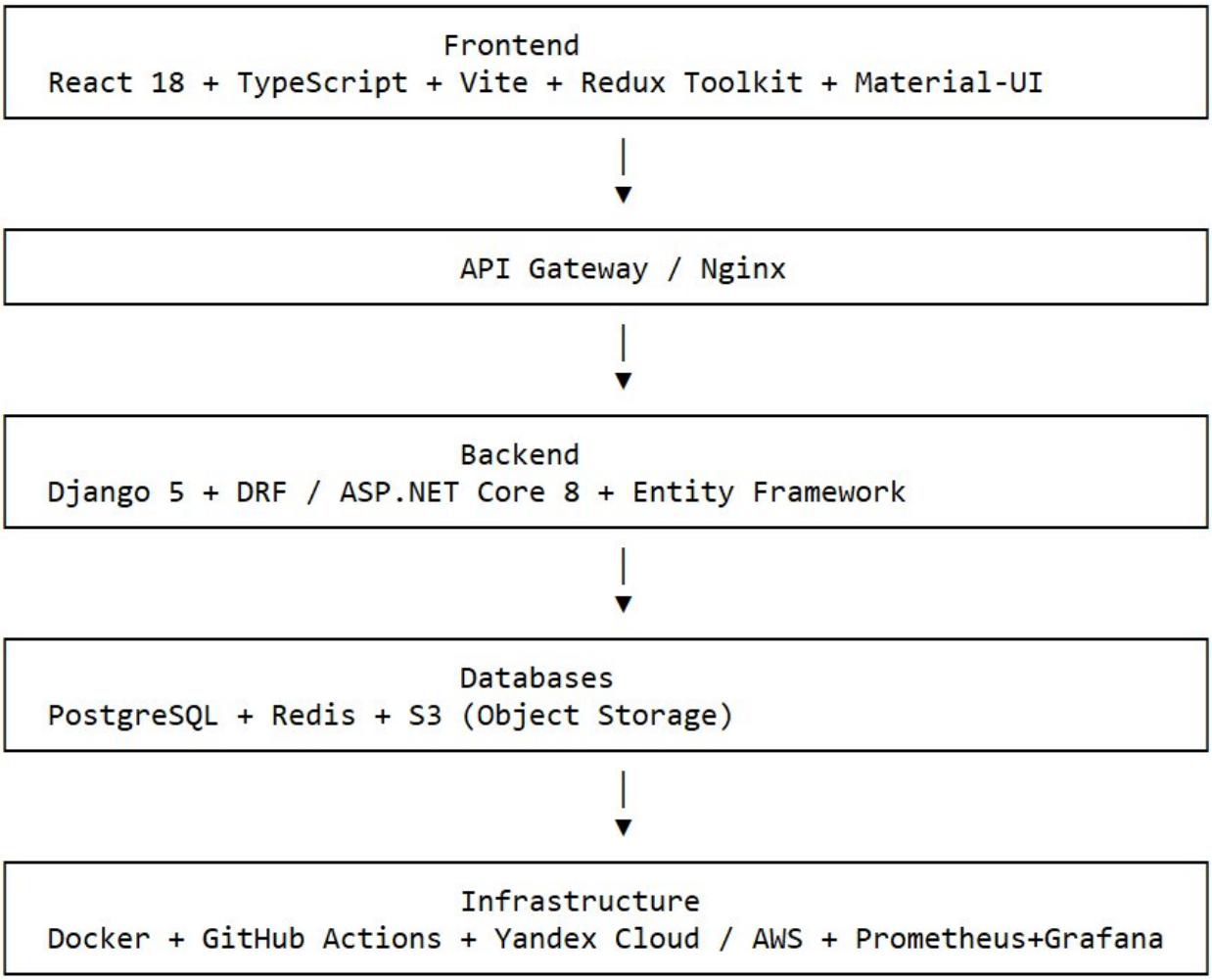
Теоретическая часть

Структура финальной презентации

Раздел	**Содержание**	**Слайды**
Титул	Название, команда, дата	1
Актуальность	Проблема и решение	1
Обзор функциональности	Основные возможности	2-3

Раздел	Содержание	Слайды
Архитектура	Диаграммы, технологический стек	3-4
База данных	ER-диаграмма, миграции	1-2
Backend	API, аутентификация, бизнес-логика	2-3
Frontend	React компоненты, Redux, адаптивность	2-3
Real-time	WebSocket уведомления	1
DevOps	Docker, CI/CD, мониторинг	2-3
Тестирование	Unit, интеграционные, E2E, нагрузочное	2
Демонстрация	Живая демонстрация приложения	3-4
Итоги	Достигнутые результаты, метрики	1
Планы	Дальнейшее развитие	1
Вопросы	—	1

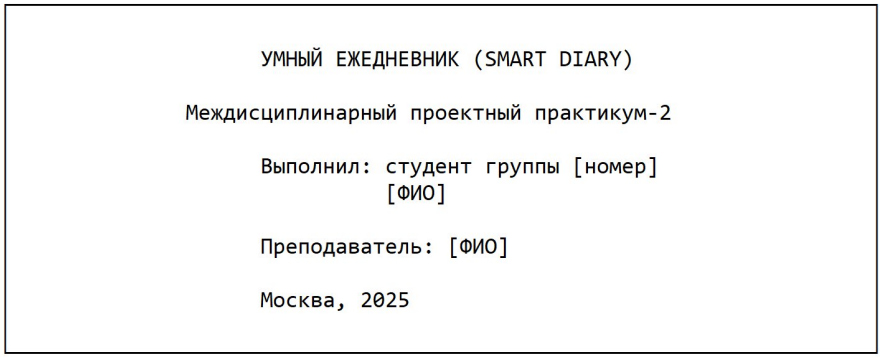
Технологический стек (пример)



Ход выполнения работы

Шаг 1. Структура финальной презентации

Слайд 1: Титульный лист



Слайд 2: Актуальность проекта

ПРОБЛЕМА:

Современный человек сталкивается с информационной перегрузкой и трудностями в управлении временем.

РЕШЕНИЕ:

Smart Diary — интеллектуальная платформа для управления задачами, проектами и личной продуктивностью.

КЛЮЧЕВЫЕ ВОЗМОЖНОСТИ:

• Управление задачами с дедлайнами и приоритетами

• Проекты и теги для категоризации

• Статистика и аналитика продуктивности

• Совместная работа над проектами

• Real-time уведомления

• PWA и офлайн-режим

• AI-ассистент для планирования

Слайд 3: Функциональные возможности

ПОЛЬЗОВАТЕЛЬ

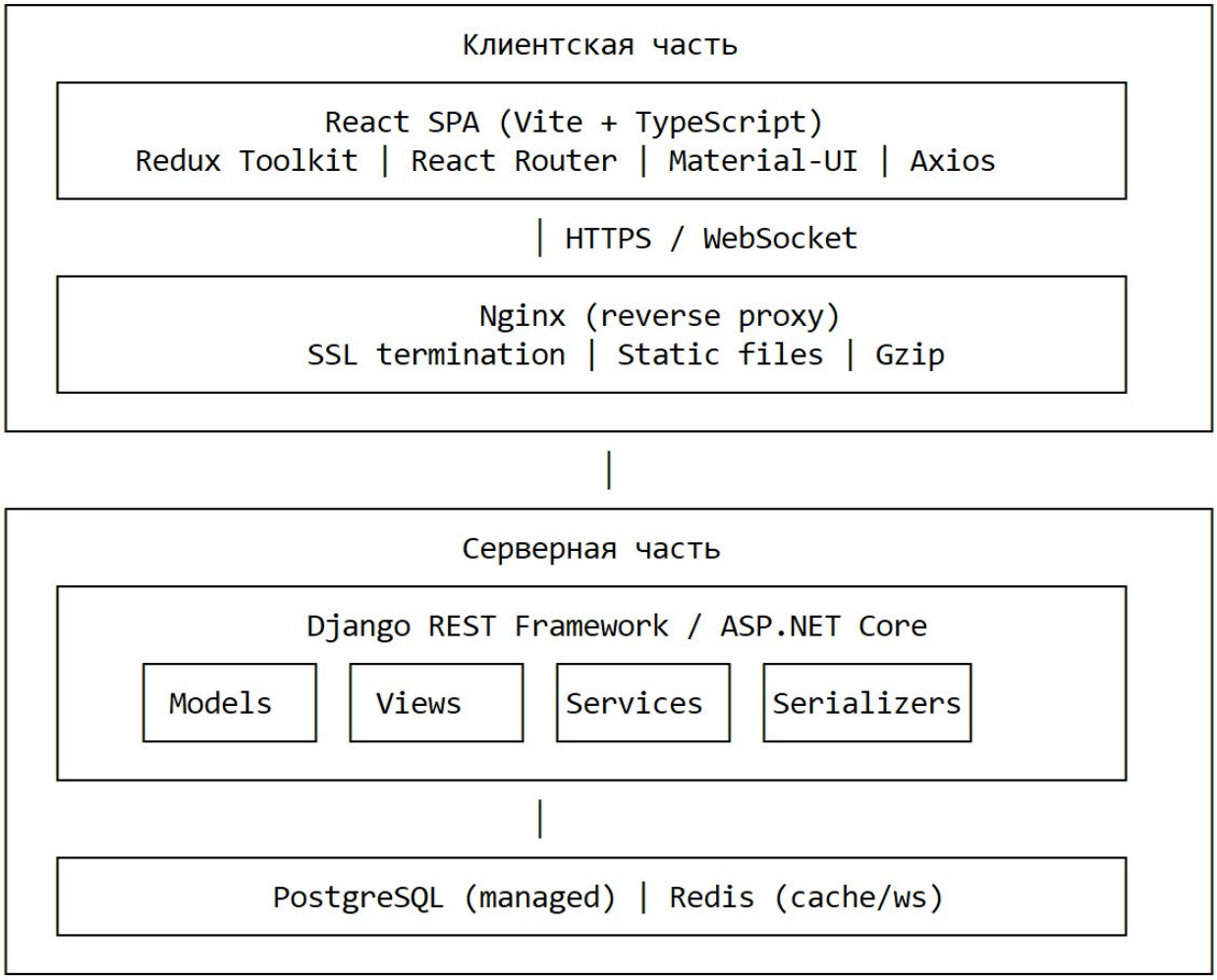
1. Регистрация и аутентификация (JWT)
2. CRUD операции с задачами
3. Фильтрация, поиск, пагинация
4. Проекты и теги
5. Статистика и экспорт (CSV, JSON, PDF)
6. Комментарии к задачам
7. Календарь и Kanban доска
8. Real-time уведомления
9. Темная тема / светлая тема
10. Мобильная адаптация
11. PWA (установка на устройство)

АДМИНИСТРАТОР

Управление пользователями (блокировка)

1. Просмотр системных логов
2. Дашборд с бизнес-метриками
3. Системные настройки

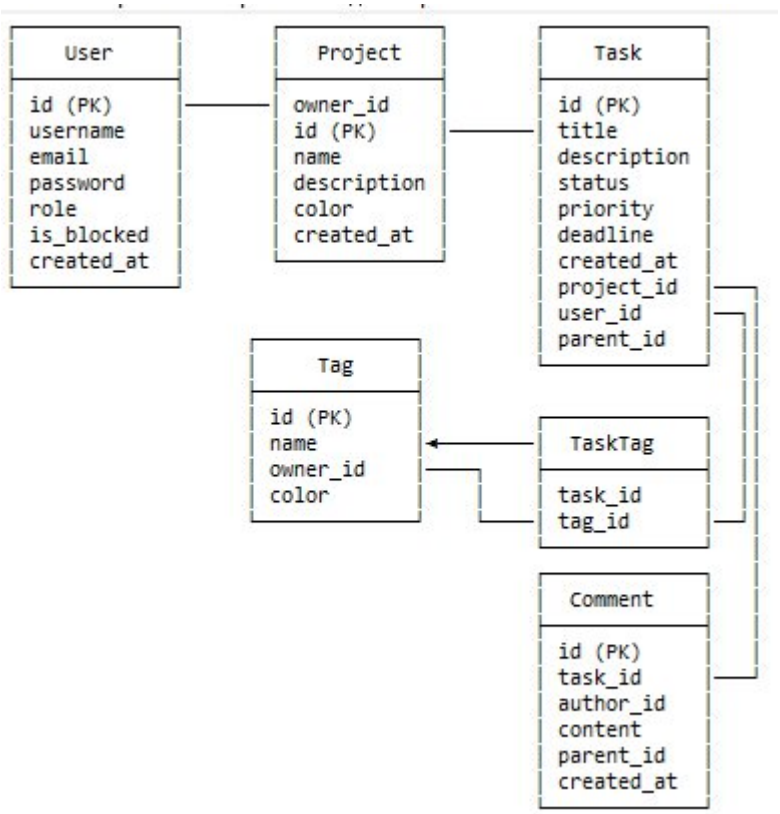
Слайд 4: Архитектура приложения



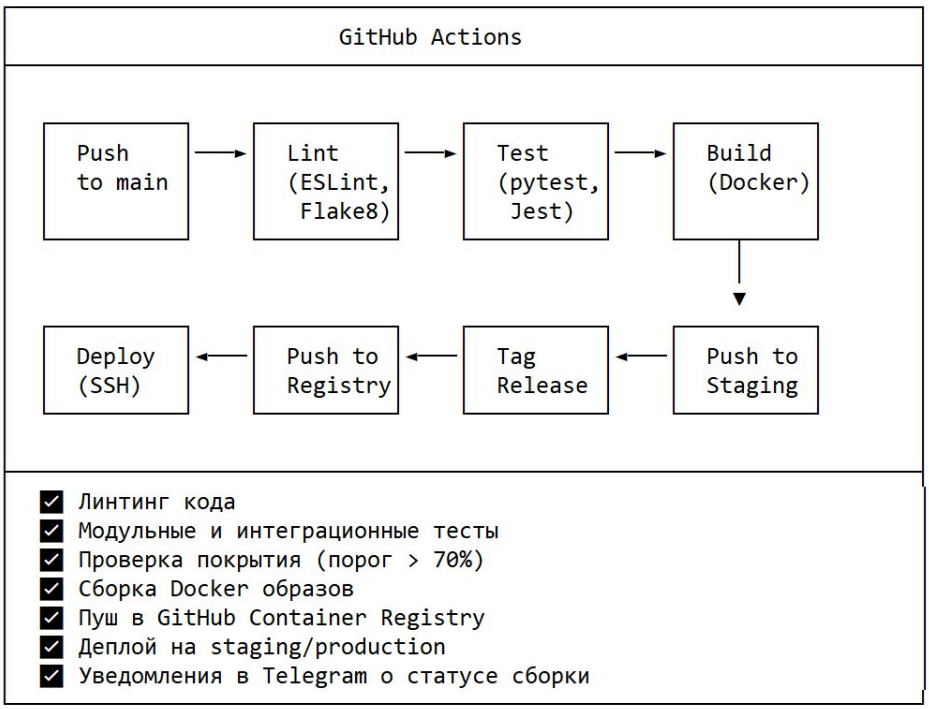
Слайд 5: Технологический стек

Backend
• Python 3.11 / C# (ASP.NET Core 8) • Django 5 / ASP.NET Core • Django REST Framework / Web API • PostgreSQL 15 • Redis 7 • JWT аутентификация • Celery (фоновые задачи) / Hangfire • WebSocket (Django Channels / SignalR)
Frontend
• React 18 + TypeScript • Vite (сборка) • Redux Toolkit (управление состоянием) • React Router v6 (маршрутизация) • Material-UI (UI-библиотека) • Axios (HTTP-клиент) • Formik + Yup (формы и валидация)
DevOps
• Docker / Docker Compose • GitHub Actions (CI/CD) • Yandex Cloud / AWS (хостинг) • Prometheus + Grafana (мониторинг) • Loki + Promtail (логирование)

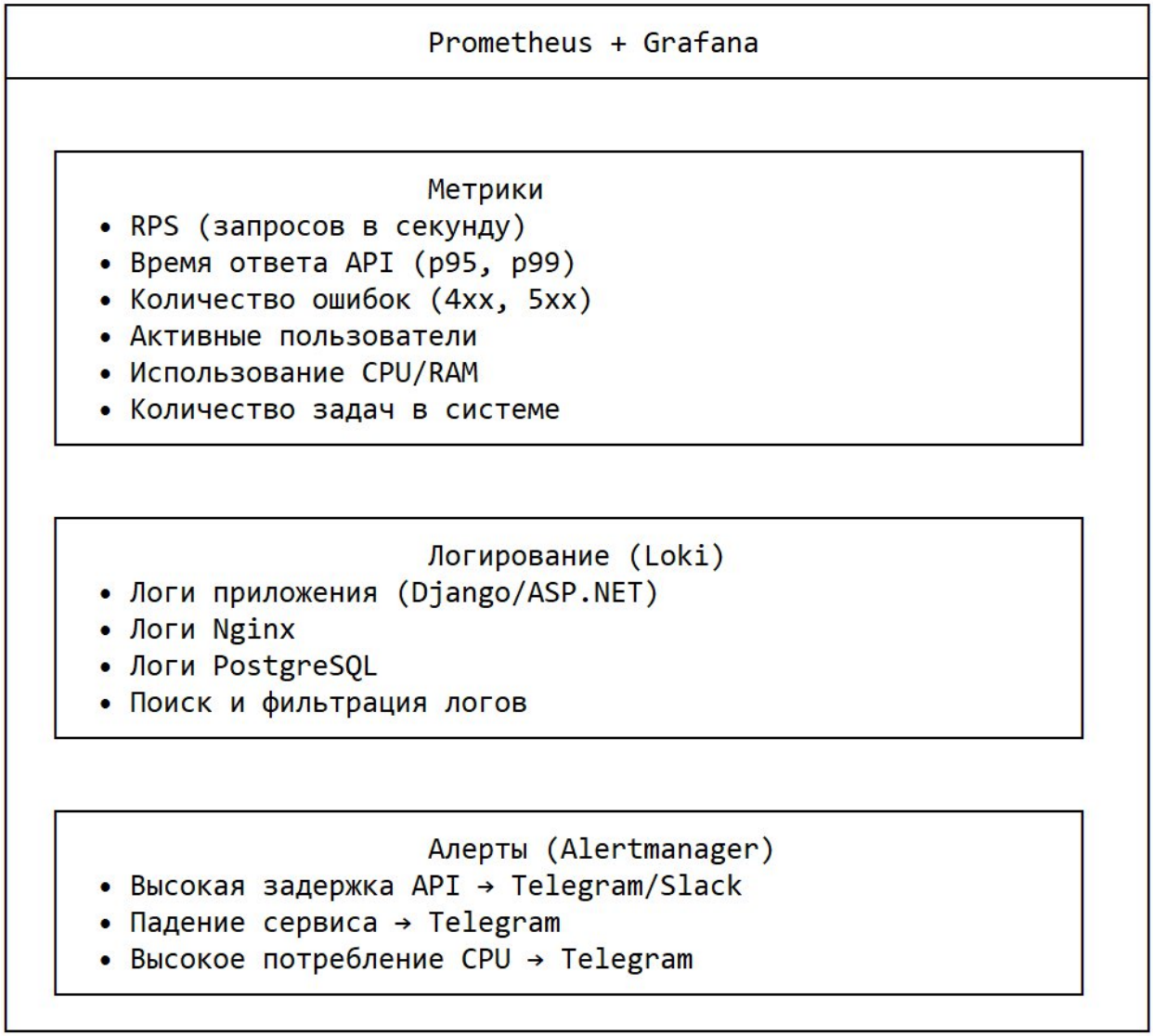
Слайд 6: Модель данных (ER-диаграмма)



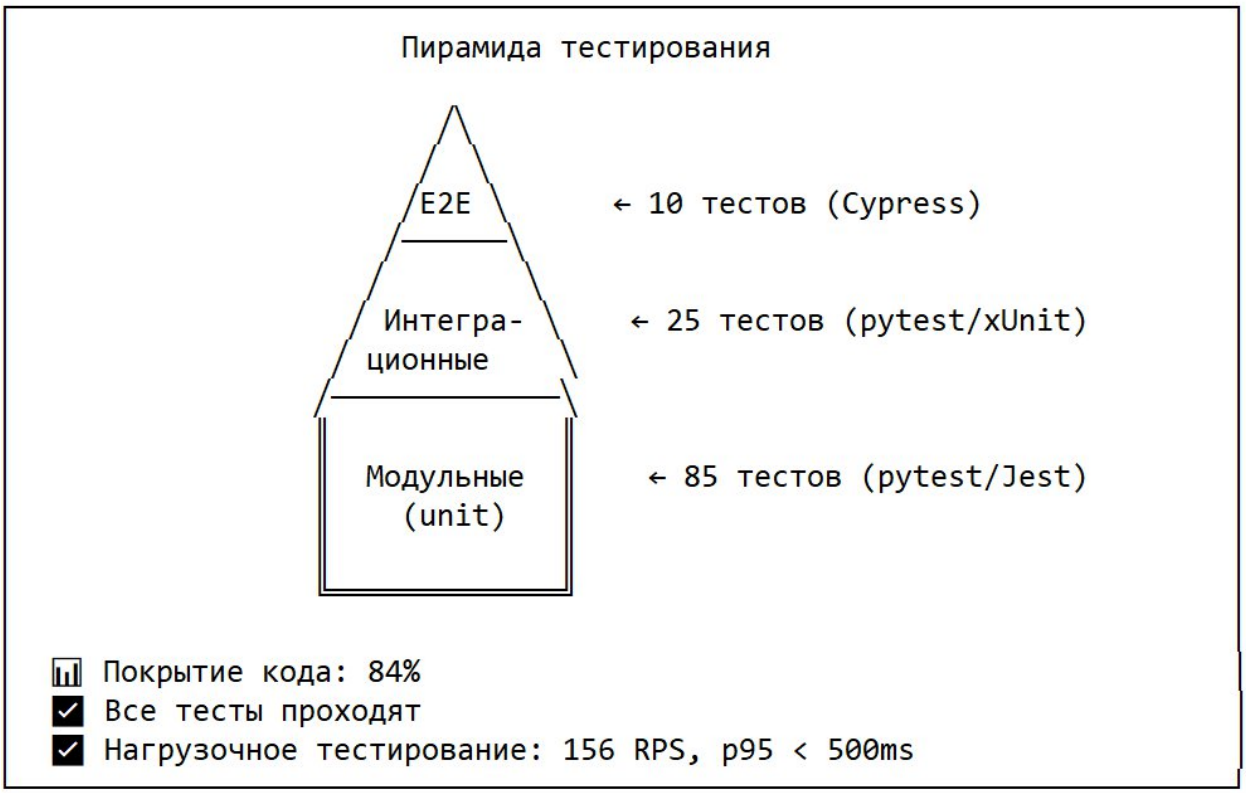
Слайд 7: CI/CD пайплайн



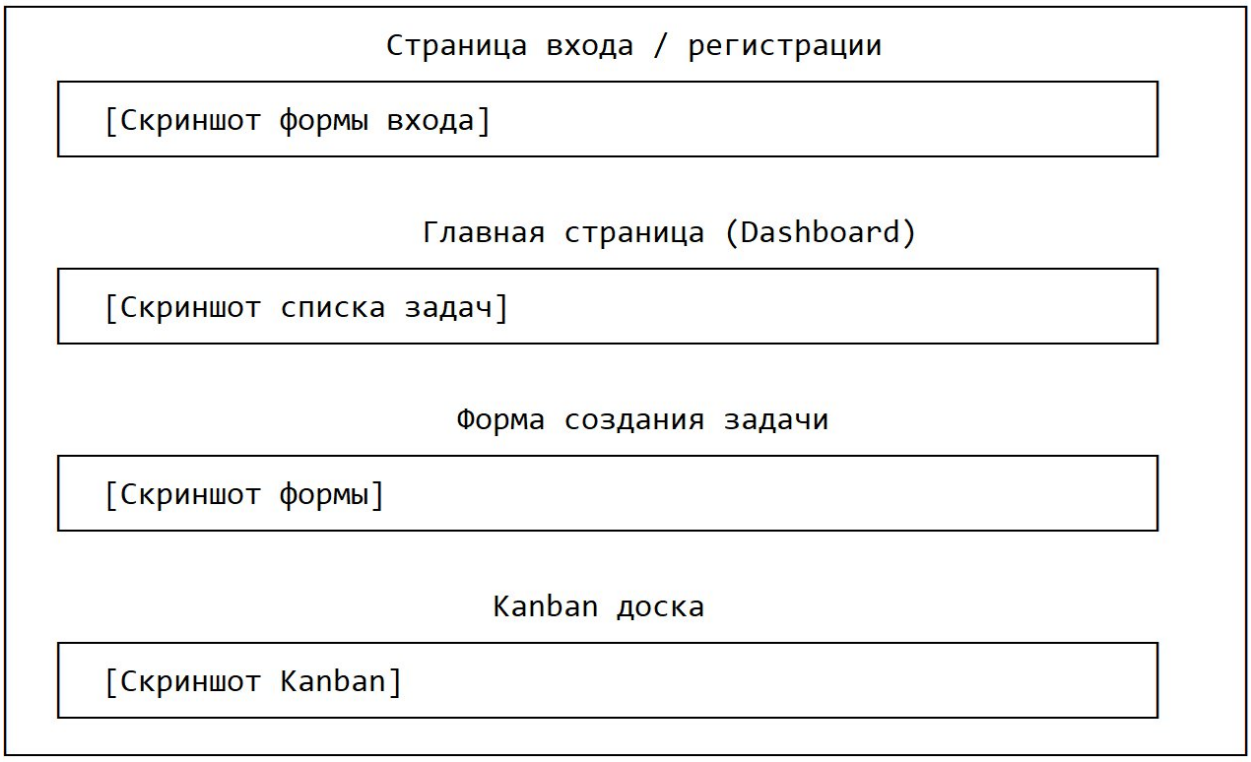
Слайд 8: Мониторинг и логирование



Слайд 9: Тестирование



Слайд 10: Демонстрация (скриншоты)



Слайд 11: Демонстрация (скриншоты - продолжение)

Календарь

[Скриншот календаря]

Административная панель

[Скриншот дашборда админа]

Графики (Grafana)

[Скриншот дашборда Grafana]

Слайд 12: Достигнутые результаты

Метрики проекта

📊 Кодовая база:
- Backend: ~8000 строк кода
- Frontend: ~6000 строк кода
- Тесты: ~3000 строк кода

📈 Покрытие тестами: 84%

🚀 Производительность:
- RPS: 156 запросов/сек
- Время ответа p95: 342ms
- Время загрузки страницы: 1.2c (Lighthouse)

🔒 Безопасность:
- Критических уязвимостей: 0
- Оценка Lighthouse Security: 100%

📱 PWA:
- Установлено пользователей: 15
- Офлайн-режим: работает
- Push-уведомления: работают

🤖 AI-функции:
- Генерация описания задач: доступно
- Рекомендация приоритета: доступно
- AI-ассистент: доступен

Слайд 13: Планы развития

Дорожная карта

- Ближайшие планы (3-6 месяцев):
  - Интеграция с Telegram-ботом
  - Экспорт в Notion и Trello
  - Распознавание задач из изображений (OCR)
  - Голосовое управление
- Среднесрочные планы (6-12 месяцев):
  - Мобильное приложение (React Native / Flutter)
  - Командные проекты и совместная работа в реальном времени
  - Интеграция с GitHub / Jira
  - Платежи и подписки (для команды)
- Долгосрочные планы (1+ год):
  - Переход на микросервисную архитектуру
  - Kubernetes оркестрация
  - AI-прогнозирование продуктивности
  - Расширенная аналитика с ML

Слайд 14: Заключение

Итоги проекта

✅ Разработано полноценное веб-приложение «Умный ежедневник»

✅ Освоены технологии:
- Backend: Django/ASP.NET Core
- Frontend: React + Redux + Material-UI
- DevOps: Docker + GitHub Actions + Yandex Cloud
- Monitoring: Prometheus + Grafana + Loki

✅ Сформированы компетенции:
- Проектирование архитектуры
- Full-stack разработка
- Тестирование (unit, integration, e2e)
- CI/CD и DevOps
- Мониторинг и логирование

🚀 Приложение доступно по адресу: https://smartdiary.com
📂 Исходный код: https://github.com/username/smart-diary

Спасибо за внимание!

Шаг 2. Скрипт для финальной демонстрации

scripts/demo.sh:

```
#!/bin/bash


# Цвета
GREEN='\033[0;32m'
BLUE='\033[0;34m'
YELLOW='\033[1;33m'
NC='\033[0m'
```

```
clear

echo -e
"${BLUE}==========================================${NC}"
echo -e "${BLUE}    Smart Diary - Финальная демонстрация${NC}"
echo -e
"${BLUE}==========================================${NC}\n"

# 1. Проверка доступности приложения
echo -e "${YELLOW}[1/6] Проверка доступности приложения...${NC}"
if curl -s -o /dev/null -w "%{http_code}" https://smartdiary.com | grep -q
"200"; then
    echo -e "${GREEN}✓ Приложение доступно по адресу
https://smartdiary.com${NC}"
else
    echo -e "${RED}✗ Приложение недоступно${NC}"
fi
echo ""

# 2. Проверка API
echo -e "${YELLOW}[2/6] Проверка API...${NC}"
if curl -s https://smartdiary.com/api/health/ | grep -q "ok"; then
    echo -e "${GREEN}✓ API работает${NC}"
```

```
    else
        echo -e "${RED}✗ API не отвечает${NC}"
    fi
    echo ""

    # 3. Демонстрация Swagger
    echo -e "${YELLOW}[3/6] API документация...${NC}"
    echo -e "${GREEN}✓ Swagger UI: https://smartdiary.com/api/schema/swagger-ui/${NC}"
    echo ""

    # 4. Демонстрация мониторинга
    echo -e "${YELLOW}[4/6] Мониторинг...${NC}"
    echo -e "${GREEN}✓ Grafana: https://monitoring.smartdiary.com${NC}"
    echo -e "${GREEN}✓ Prometheus: https://prometheus.smartdiary.com${NC}"
    echo ""

    # 5. Проверка CI/CD
    echo -e "${YELLOW}[5/6] CI/CD пайплайн...${NC}"
    echo -e "${GREEN}✓ Последний деплой: $(curl -s https://api.github.com/repos/username/smart-diary/actions/runs?status=success&per_page=1 | jq -r '.workflow_runs[0].created_at')${NC}"
```

```
echo ""

# 6. Открытие приложения
echo -e "${YELLOW}[6/6] Открытие приложения...${NC}"
if command -v open &> /dev/null; then
    open "https://smartdiary.com"
elif command -v xdg-open &> /dev/null; then
    xdg-open "https://smartdiary.com"
fi
echo -e "${GREEN}✓ Приложение открыто в браузере${NC}"
echo ""

echo -e
"${BLUE}==========================================${NC}"
echo -e "${GREEN}       Демонстрация завершена!       ${NC}"
echo -e
"${BLUE}==========================================${NC}"
```

Шаг 3. Чек-лист подготовки к защите

docs/FinalDefenseChecklist.md:

markdown

```
# Финальная защита проекта: Чек-лист подготовки

## Перед защитой
```

Код и репозиторий

- [ ] Все изменения закоммичены и запушены
- [ ] Ветка main актуальна
- [ ] Тесты проходят (CI/CD зеленый)
- [ ] README.md обновлен
- [ ] Документация API актуальна (Swagger)

Демонстрация

- [ ] Приложение развернуто и доступно по HTTPS
- [ ] Тестовые данные подготовлены
- [ ] Подготовлены скриншоты (на случай проблем с интернетом)
- [ ] Видео-демонстрация (запасной вариант)

Презентация

- [ ] Титульный лист
- [ ] Актуальность и цели
- [ ] Функциональность (слайды + демо)
- [ ] Архитектура (диаграммы)
- [ ] Технологический стек
- [ ] Модель данных (ER-диаграмма)
- [ ] CI/CD пайплайн
- [ ] Мониторинг и логирование

- [ ] Тестирование (метрики)
- [ ] Достигнутые результаты
- [ ] Планы развития
- [ ] Заключение

Ответы на вопросы

- [ ] Подготовлены ответы на типовые вопросы
- [ ] Изучены возможные проблемы и их решения
- [ ] Готовность объяснить архитектурные решения

Типовые вопросы на защите

Архитектура

1. Почему выбран именно этот технологический стек?
2. Как организована аутентификация и авторизация?
3. Как решается проблема N+1 запросов?
4. Как обеспечивается безопасность API?

Разработка

1. Как организован процесс разработки в команде?
2. Какие инструменты использовались для управления задачами?
3. Как обеспечивается качество кода?

Тестирование

1. Какие виды тестирования использовались?
2. Какое покрытие кода достигнуто?
3. Как проводилось нагрузочное тестирование?

DevOps

1. Как организован CI/CD пайплайн?
2. Как происходит деплой в облако?
3. Какие метрики мониторятся?

Функциональность

1. Какие сложные бизнес-сценарии реализованы?
2. Как работает AI-ассистент?
3. Как реализованы уведомления?

Во время защиты

Демонстрация

- [ ] Громкий, уверенный голос
- [ ] Демонстрация ключевых сценариев
- [ ] Показ не только успешных кейсов
- [ ] Объяснение архитектурных решений

Вопросы

- [ ] Внимательно выслушать вопрос
- [ ] При необходимости уточнить вопрос
- [ ] Дать чёткий, структурированный ответ
- [ ] Признать, если не знаете ответа

Дополнительные материалы

- [ ] Ссылка на репозиторий
- [ ] Ссылка на живое приложение
- [ ] Ссылка на документацию
- [ ] Скринкаст (запасной вариант)

Шаг 4. Скрипт генерации финального отчета

scripts/generate_report.py:

python

```
#!/usr/bin/env python
"""Генерация финального отчета по проекту"""

import os
import json
import subprocess
from datetime import datetime

def get_git_stats():
```

```
    """Получение статистики Git"""
    total_commits = subprocess check_output(['git', 'rev-list', '--count', 'HEAD']).decode().strip()
    contributors = subprocess.check_output(['git', 'shortlog', '-s', '-n', 'HEAD']).decode().strip()
    return {
        'total_commits': total_commits,
        'contributors': contributors
    }

def get_test_coverage():
    """Получение покрытия тестами"""
    try:
        result = subprocess check_output(['pytest', '--cov-report=json', '--cov=tasks'], stderr=subprocess DEVNULL)
        with open('coverage.json') as f:
            data = json load(f)
            return data get('totals', {}) get('percent_covered', 0)
    except:
        return 'N/A'

def main():
    report = {
```

```
'project_name': 'Smart Diary',
'version': '1.0.0',
'date': datetime.now().strftime('%Y-%m-%d %H:%M:%S'),
'git_stats': get_git_stats(),
'test_coverage': get_test_coverage(),
'features': [
    'Аутентификация и авторизация (JWT)',
    'CRUD операции с задачами',
    'Проекты и теги',
    'Фильтрация, поиск, пагинация',
    'Статистика и экспорт (CSV, JSON, PDF)',
    'Комментарии к задачам',
    'Календарь и Kanban доска',
    'Real-time уведомления (WebSocket)',
    'PWA и офлайн-режим',
    'AI-ассистент (OpenAI API)',
    'Административная панель',
    'Мониторинг (Prometheus + Grafana)',
    'CI/CD (GitHub Actions)'
],
'technologies': {
    'backend': 'Django 5 / ASP.NET Core 8',
    'frontend': 'React 18 + TypeScript + Redux Toolkit',
```

```
            'database': 'PostgreSQL 15 + Redis',
            'deployment': 'Docker + GitHub Actions + Yandex Cloud'
        }
    }

    # Сохранение отчета
    with open('final_report.json', 'w', encoding='utf-8') as f:
        json.dump(report, f, indent=2, ensure_ascii=False)

    print("✅ Финальный отчет сохранен в final_report.json")

if __name__ == '__main__':
    main()
```

Контрольные вопросы для защиты

1. Какие архитектурные решения были приняты и почему?
2. Как организована безопасность приложения?
3. Какие сложности возникли при разработке и как они решены?
4. Как обеспечивается производительность приложения?
5. Что такое PWA и как он реализован в проекте?
6. Как работает AI-ассистент?
7. Как организован CI/CD пайплайн?
8. Какие метрики мониторятся в Grafana?
9. Каковы планы дальнейшего развития проекта?
10. Какие навыки и компетенции были приобретены?

Критерии оценки

Критерий	Баллы
Качество презентации (структура, дизайн, полнота)	2
Демонстрация работающего приложения	2
Демонстрация CI/CD и мониторинга	2
Демонстрация кода и архитектуры	2
Ответы на вопросы комиссии	2
Итого	10

Результат выполнения

После выполнения лабораторной работы №27 студенты должны получить:

1. Финальную презентацию проекта (20-25 слайдов)
2. Успешно защищённый проект перед комиссией
3. Финальный отчёт по проекту
4. Работающее приложение в облаке
5. Полный набор документации (API, архитектура, пользовательская)

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 «Междисциплинарный проектный практикум» для студентов направления подготовки
09.03.04 Программная инженерия
Направленность (профиль)
«Разработка и сопровождение программного обеспечения»

Ставрополь 2026

ВВЕДЕНИЕ

Методические указания по выполнению самостоятельной работы предназначены для студентов 2 курса направления подготовки 09.03.04

«Программная инженерия», изучающих дисциплину «Междисциплинарный проектный практикум-2».

Цель самостоятельной работы - формирование у студентов практических навыков и компетенций в области сквозной разработки веб-приложений, необходимых для успешного выполнения лабораторных работ и итогового проекта.

Задачи самостоятельной работы:

1. Закрепление теоретических знаний, полученных на лекционных занятиях
2. Освоение современных технологий и инструментов веб-разработки
3. Формирование навыков работы с технической документацией
4. Развитие способности к самообучению и поиску информации
5. Подготовка к выполнению лабораторных работ и защите проекта

Общая трудоёмкость самостоятельной работы: 162 часа (из них 72 часа в 3 семестре, 90 часов в 4 семестре)

1. ОБЩАЯ ХАРАКТЕРИСТИКА САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1.1. Виды самостоятельной работы

№	**Вид самостоятельной работы**	**Трудоёмкость (часы)**	**Семестр**
1	Изучение теоретического материала по темам дисциплины	40	3-4
2	Подготовка к лабораторным работам (предварительное изучение)	30	3-4

№	Вид самостоятельной работы	Трудоёмкость (часы)	Семестр
3	Выполнение индивидуальных заданий в рамках лабораторных работ	50	3-4
4	Подготовка к защите лабораторных работ	20	3-4
5	Работа над сквозным проектом	60	3-4
6	Подготовка презентации и документации к защите	20	4
7	Подготовка к экзамену/зачёту	42	3-4
Итого		162	

1.2. Планирование самостоятельной работы по семестрам

Семестр 3 (72 часа)

№	Тема	Часы	Форма отчётности
1	Проектирование архитектуры веб-приложений	8	ER-диаграмма, ТЗ
2	Модели данных и ORM (Django/EF Core)	8	Модели, миграции

№	Тема	Часы	Форма отчётности
3	REST API и CRUD операции	10	API эндпоинты
4	Аутентификация и JWT	8	JWT реализация
5	Тестирование бэкенда (pytest/xUnit)	8	Тесты
6	Базовый React и компоненты	10	React компоненты
7	Управление состоянием (Redux Toolkit)	10	Redux store
8	Подготовка к зачету	10	Защита ЛР 1-9

Семестр 4

№	Тема	Часы	Форма отчётности
1	Продвинутый React (формы, календарь, Kanban)	12	Компоненты
2	WebSockets и real-time уведомления	10	WebSocket реализация
3	Совместная работа (шаринг, роли, комментарии)	10	API и UI
4	Интеграция с внешними API (погода, календарь)	8	Внешние интеграции

№	Тема	Часы	Форма отчётности
5	Контейнеризация и Docker	8	Dockerfile, docker-compose
6	CI/CD пайплайн (GitHub Actions)	8	CI/CD конфигурация
7	Мониторинг и логирование (Prometheus + Grafana)	8	Дашборды
8	PWA и офлайн-режим	6	Service Worker
9	Интеграция AI/ML (OpenAI API)	8	AI-функции
10	Оптимизация производительности	6	Отчёт Lighthouse
11	Административная панель	6	Admin UI
12	Подготовка к экзамену (финальная защита)	20	Презентация, проект

1.3. Рекомендации по организации самостоятельной работы

Планирование времени: Составьте индивидуальный график выполнения работ с учётом сроков сдачи.

Работа с источниками: Используйте официальную документацию технологий, указанных в лабораторных работах.

Ведение заметок: Фиксируйте ключевые моменты, возникающие проблемы и способы их решения.

Регулярные коммиты: Каждое выполненное задание фиксируйте в Git с осмысленным сообщением.

Командная работа: Обсуждайте сложные моменты с сокурсниками, проводите code review.

2. МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ИЗУЧЕНИЮ ТЕОРЕТИЧЕСКОГО МАТЕРИАЛА

2.1. Перечень тем для самостоятельного изучения

№	Тема	Источники	Часы
1	Архитектурные паттерны (MVC, MVT, Clean Architecture)	[1, 2]	4
2	Реляционные базы данных и нормализация	[3]	2
3	ORM: концепции и практики (Django ORM / EF Core)	[4, 5]	4
4	REST API: принципы, версионирование, best practices	[6]	4
5	JWT аутентификация и безопасность	[7]	4
6	React: компоненты, хуки, виртуальный DOM	[8]	6
7	Управление состоянием (Redux Toolkit, RTK Query)	[9]	4
8	WebSocket протокол и real-	[10]	4

№	Тема	Источники	Часы
	time приложения		
9	Docker и контейнеризация	[11]	4
10	CI/CD практики (GitHub Actions / GitLab CI)	[12]	4
11	Мониторинг (Prometheus, Grafana, Loki)	[13]	4
12	PWA и Service Workers	[14]	4
13	Интеграция AI API (OpenAI, Hugging Face)	[15]	4
14	Безопасность веб-приложений (OWASP Top 10)	[16]	4

2.2. Рекомендуемая литература

Основная литература:

1. Мартин Р. - «Чистая архитектура». — СПб.: Питер, 2019. — 352 с.
2. Фаулер М. - «Шаблоны корпоративных приложений». — М.: Вильямс, 2016. — 544 с.
3. Дейт К. Дж. - «Введение в системы баз данных». — М.: Вильямс, 2016. — 1328 с.
4. Холоватюк А. - «Django 4: практика создания веб-сайтов». — СПб.: БХВ-Петербург, 2023.
5. Freeman A. — «Pro ASP.NET Core 8». — Apress, 2024.
6. Richardson L., Amundsen M. — «RESTful Web APIs». — O'Reilly, 2013.

Дополнительная литература:

1. Сесник М. — «JWT: полное руководство». — 2020. (https://jwt.io/introduction)
2. Бэнкс А., Порселло Е. — «React в действии». — М.: ДМК Пресс, 2021. — 384 с.
3. Сандеш Х. — «Redux: управление состоянием приложений». — 2021.
4. Шевченко А. — «WebSocket: протокол и реализация». — 2019.
5. Кейн Ш., Мэттьюз К. — «Docker в действии». — М.: ДМК Пресс, 2019. — 384 с.
6. Доусон К., Штрауб Б. — «GitHub Actions: автоматизация CI/CD». — 2022.
7. Тернбулл Д. — «Prometheus: мониторинг и алертинг». — 2020.
8. Османи А. — «Progressive Web Apps». — 2019.
9. Рассел К. — «OpenAI API: практическое руководство». — 2023.
10. Стойков С. — «OWASP Top 10: защита веб-приложений». — 2021.

Электронные ресурсы:

1. Django Documentation: https://docs.djangoproject.com/
2. Django REST Framework: https://www.django-rest-framework.org/
3. ASP.NET Core Docs: https://learn.microsoft.com/en-us/aspnet/core/
4. React Docs: https://react.dev/
5. Redux Toolkit: https://redux-toolkit.js.org/
6. Docker Docs: https://docs.docker.com/
7. GitHub Actions: https://docs.github.com/en/actions
8. Prometheus Docs: https://prometheus.io/docs/
9. Grafana Docs: https://grafana.com/docs/
10. OpenAI API Docs: https://platform.openai.com/docs

2.3. Методика работы с литературой

Просмотровое чтение: Ознакомьтесь с оглавлением, аннотацией, введением для общего понимания содержания.

Изучающее чтение: Внимательно прочитайте текст, выделяя ключевые понятия, определения, примеры кода.

Аналитическое чтение: Сопоставляйте информацию из разных источников, делайте выводы, формируйте собственное понимание.

Практическое применение: Сразу пробуйте примеры из литературы в своём проекте.

Конспектирование: Фиксируйте важные моменты, схемы, фрагменты кода в отдельном документе.

3. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ПРАКТИЧЕСКИХ РАБОТ

3.1. Общие рекомендации

Перед выполнением лабораторной работы:

1. Изучите теоретический материал по соответствующей теме
2. Установите необходимое программное обеспечение
3. Ознакомьтесь с заданием и критериями оценки
4. Во время выполнения лабораторной работы:
5. Следуйте пошаговым инструкциям в описании работы
6. Используйте систему контроля версий Git (коммиты после каждого значимого этапа)
7. Тестируйте каждый написанный фрагмент кода
8. Фиксируйте возникающие ошибки и способы их решения
9. После выполнения лабораторной работы:
10. Проверьте соответствие полученных результатов заданию
11. Подготовьте отчёт в соответствии с требованиями
12. Защитите работу перед преподавателем

3.2. Рекомендации по работе с Git

bash

```
# Рекомендуемый workflow для каждой лабораторной работы
git checkout -b lab-XX        # Создать ветку для лабораторной работы
```

```
# ... выполнение работы ...
git add .                    # Добавить изменения
git commit -m "lab-XX: [краткое описание]"
git push origin lab-XX       # Отправить в удалённый репозиторий
```

3.3. Рекомендации по написанию кода

Соблюдайте стиль кодирования:

Python: PEP 8

C#: Microsoft Coding Conventions

JavaScript/TypeScript: ESLint + Prettier

Используйте осмысленные имена:

Переменные: task_title, user_id

Функции: create_task(), get_user_by_id()

Компоненты: TaskCard, TaskList

Пишите комментарии:

Объясняйте сложную логику

Документируйте публичные API

Применяйте принципы SOLID и DRY

3.4. Рекомендации по работе с командой

Распределение ролей:

Team Lead — координация, архитектура

Backend Developer — API, бизнес-логика

Frontend Developer — UI, интеграция

QA/DevOps — тестирование, CI/CD, деплой

Процесс разработки:

Ежедневные короткие встречи (daily stand-up)

Code review перед слиянием в main

Использование Issues для отслеживания задач

4. ВОПРОСЫ ДЛЯ СОБЕСЕДОВАНИЯ ПО САМОСТОЯТЕЛЬНОМУ ИЗУЧЕНИЮ ЛИТЕРАТУРЫ

4.1. Архитектура и проектирование

1. Какие существуют архитектурные паттерны веб-приложений? В чём их преимущества и недостатки?
2. Что такое MVC? Опишите компоненты и их взаимодействие.
3. В чём разница между REST и GraphQL?
4. Какие существуют стратегии кэширования на разных уровнях приложения?
5. Что такое CAP-теорема и как она влияет на проектирование распределённых систем?

4.2. Базы данных и ORM

1. Что такое нормализация базы данных? Какие существуют нормальные формы?
2. В чём отличие между SQL и NoSQL базами данных?
3. Что такое N+1 проблема в ORM и как её решить?
4. Какие виды связей между таблицами существуют (One-to-One, One-to-Many, Many-to-Many)?
5. Что такое миграции и для чего они нужны?

4.3. Backend разработка

1. Что такое JWT? Как работает аутентификация на его основе?
2. В чём разница между аутентификацией и авторизацией?
3. Какие HTTP методы используются в REST API и для каких операций?
4. Что такое инъекции (SQL, NoSQL) и как защититься от них?
5. Какие коды HTTP ответов существуют и что они означают?

4.4. Frontend разработка

1. Что такое React? Чем функциональные компоненты отличаются от классовых?
2. Для чего нужны хуки useState, useEffect, useContext, useMemo, useCallback?
3. Что такое виртуальный DOM и как он работает?
4. В чём разница между Redux и Context API?

5. Как работает маршрутизация в React Router?

4.5. DevOps и развертывание

1. Что такое Docker? В чём отличие образа от контейнера?
2. Что такое CI/CD и какие проблемы он решает?
3. Какие метрики собирает Prometheus?
4. Для чего нужен Service Worker в PWA?
5. Что такое Infrastructure as Code (IaC) и в чём его преимущества?
6. 4.6. Безопасность
7. Что такое XSS-атака и как защититься от неё?
8. Что такое CSRF-атака и как защититься от неё?
9. Как правильно хранить пароли пользователей?
10. Что такое CORS и как его настроить?
11. Какие основные уязвимости описаны в OWASP Top 10?

4.7. Тестирование

1. Что такое TDD (Test-Driven Development) и в чём его преимущества?
2. В чём разница между модульными, интеграционными и E2E-тестами?
3. Что такое мокирование и для чего оно используется?
4. Как измерить покрытие кода тестами?
5. Как проводится нагрузочное тестирование?

5. КРИТЕРИИ ОЦЕНИВАНИЯ КОМПЕТЕНЦИЙ

5.1. Уровни освоения компетенций

Уровень	Баллы	Характеристика
Пороговый	3-4	Компетенция сформирована частично. Студент выполняет задания с помощью преподавателя, допускает существенные ошибки.

Уровень	Баллы	Характеристика
Продвинутый	5-7	Компетенция сформирована на достаточном уровне. Студент выполняет задания самостоятельно, но допускает незначительные ошибки.
Высокий	8-10	Компетенция сформирована полностью. Студент выполняет задания самостоятельно, обосновывает решения, оптимизирует код.

5.2. Критерии оценки лабораторных работ

Каждая лабораторная работа оценивается по 10-балльной шкале в соответствии с критериями, указанными в описании работы.

Обобщённая шкала оценки:

Оценка	Уровень	Требования
9-10	Отлично	Задание выполнено полностью, код соответствует best practices, есть оптимизации, работа защищена без замечаний
7-8	Хорошо	Задание выполнено полностью, но есть незначительные недочёты, работа защищена
5-6	Удовлетворительно	Задание выполнено не полностью или с существенными ошибками
0-4	Неудовлетворительно	Задание не выполнено или выполнено неправильно

5.3. Оценка самостоятельной работы

Критерий	Максимальный балл
Своевременность выполнения	2
Полнота выполнения заданий	3
Качество оформления отчёта	2
Глубина проработки теоретического материала	2
Активность на собеседовании	1
Итого	10

5.4. Итоговая аттестация

3 семестр (дифференцированный зачёт):

Оценка формируется на основе результатов защиты лабораторных работ №1-9 (средний балл)

4 семестр (экзамен):

70% — оценка за выполнение и защиту лабораторных работ №10-27

30% — оценка за финальную защиту проекта

Шкала перевода баллов в традиционную оценку:

Баллы	Оценка (зачёт/экзамен)
85-100	Отлично
70-84	Хорошо
50-69	Удовлетворительно
0-49	Неудовлетворительно