МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ПРАКТИЧЕСКИМ РАБОТАМ

«Проектная деятельность»

Направление подготовки 09.03.04 «Программная инженерия»
Направленность (профиль) «Разработка и сопровождение программного обеспечения»
Квалификация выпускника: бакалавр

Ставрополь
2026

Содержание

Практическая работа №1

Введение в проектную деятельность. Специфика IT-проектов

Цель: сформировать понимание базовых понятий проектной деятельности в области разработки программного обеспечения (ПО), различие между проектом, процессом и продуктом, ознакомиться с типовыми ролями в IT-команде и основными моделями жизненного цикла ПО.

Задачи:

1. Изучить определения и признаки понятий «проект», «процесс», «продукт» применительно к IT-сфере.
2. Научиться различать проектную и процессную деятельность.
3. Ознакомиться с ключевыми ролями в команде разработки ПО и их ответственностью.
4. Проанализировать классические модели жизненного цикла ПО (водопад, итеративные модели).
5. Применить полученные знания для анализа учебного кейса IT-проекта.

Теоретическая часть

1. Понятия «проект», «процесс», «продукт»

Проект – это временное предприятие, направленное на создание уникального продукта, услуги или результата.

Ключевые признаки проекта:

– Уникальность – результат не является повторяющимся.

– Ограниченность во времени – имеет начало и конец.

– Ограниченность ресурсов (люди, бюджет, оборудование).

– Целенаправленность – достижение конкретной цели.

Пример IT-проекта: разработка мобильного приложения для интернет-банка.

Процесс – это повторяющаяся деятельность, преобразующая входы в выходы. Процессы выполняются постоянно, поддерживая операционную деятельность организации.

Пример IT-процесса: обработка инцидентов в службе поддержки, ежедневное резервное копирование данных.

Продукт – результат проекта или процесса, который обладает ценностью для потребителя. В IT – это программное обеспечение, сервис, документация, API и т.д.

Важно: Проект создаёт продукт, а процесс – поддерживает его или обеспечивает регулярную деятельность.

2. Роли в команде IT-проекта

В зависимости от масштаба и методологии набор ролей может меняться. Базовые роли (согласно классическим и agile-подходам):

Роль	Основные обязанности
Менеджер проекта (Project Manager)	Планирование, контроль сроков и бюджета, управление рисками, коммуникация с заказчиком.
Аналитик	Сбор и формализация требований, создание спецификаций, моделирование бизнес-процессов.
Разработчик (Dev, Backend/Frontend)	Написание кода, реализация функциональности, техническое документирование.
Тестировщик (QA-engineer)	Проверка качества ПО, написание тест-кейсов, поиск дефектов.
DevOps-инженер	Настройка инфраструктуры, CI/CD, развёртывание и мониторинг.
Дизайнер (UI/UX)	Проектирование пользовательского интерфейса и опыта взаимодействия.
Product Owner (в Scrum)	Управление бэклогом, определение приоритетов, принятие решений по функционалу.

Важно понимать, что каждая роль вносит вклад в успех проекта.

3. Жизненный цикл программного обеспечения (ЖЦ ПО)

Жизненный цикл – это последовательность фаз, которые проходит ПО от идеи до вывода из эксплуатации.

Водопадная модель (Waterfall)

Классическая последовательная модель, где каждая фаза завершается до перехода к следующей.

Фазы:

Требования → Проектирование → Реализация → Тестирование → Внедрение → Эксплуатация и сопровождение.

Достоинства:

– Простота и чёткость.

– Фиксированная документация на каждом этапе.

Недостатки:

– Невозможно вернуться назад без больших затрат.

– Результат виден только в конце.

– Высокий риск несоответствия ожиданиям заказчика.

Применение: проекты с жёсткими требованиями (безопасность, авионика, госзаказ).

Итеративные модели (Agile, Scrum, итеративная разработка)

Разработка ведётся циклами (итерациями) длительностью 1–4 недели. Каждая итерация включает анализ, проектирование, кодирование и тестирование небольшой части функционала. После каждой итерации заказчик видит работающий инкремент продукта.

Достоинства:

– Быстрая обратная связь.

– Возможность менять требования.

– Снижение рисков.

Недостатки:

– Требует высокой дисциплины команды и вовлечения заказчика.

– Меньшая предсказуемость сроков и бюджета.

– Применение: стартапы, проекты с меняющимися требованиями, веб-сервисы, мобильные приложения.

Практическая часть

Задание 1. Анализ понятий

Заполните таблицу, определив, что из приведённого списка относится к проекту, процессу или продукту. Поставьте «+» в соответствующем столбце.

Пример	Проект	Процесс	Продукт
Разработка новой CRM-системы для компании			
Ежедневное проведение Scrum-митингов			
Готовое приложение для учёта задач			
Релиз версии 2.0 существующей программы			
Регламент обработки запросов в техподдержке			
Исходный код библиотеки авторизации			

Задание 2. Определение жизненного цикла

Прочитайте описание двух IT-проектов. Определите, какая модель ЖЦ (водопад или итеративная) более подходит для каждого, и обоснуйте выбор.

Кейс А. Разработка системы управления космическим спутником. Требования строго определены контрактом, ошибки критичны, заказчик не может участвовать в процессе еженедельно.

Кейс Б. Создание интернет-магазина для стартапа. Заказчик хочет запустить минимально работоспособный продукт за 2 месяца, а затем дорабатывать функционал на основе отзывов пользователей.

Задание 3. Распределение ролей

Представьте, что команда из 5 человек начинает проект «Разработка мобильного приложения для записи к врачу». Назначьте роли каждому участнику, опишите одну ключевую задачу для каждой роли в рамках первой недели работы.

Участник	Назначенная роль	Ключевая задача на первую неделю
1. Анна		
2. Иван		
3. Ольга		
4. Дмитрий		
5. Екатерина		

Дополнительное задание: предложите, какую модель ЖЦ вы выберете для этого проекта и почему.

Задание 4. Мини-кейс

«Компания заказала разработку системы управления складом. В ходе работы выяснилось, что требования к отчётам были неверно поняты аналитиком. При водопадной модели это обнаружилось на этапе тестирования. При итеративной – после второй итерации».

1. Вопросы для обсуждения (ответы кратко зафиксируйте в тетради):
2. Какие последствия будут в каждом случае?
3. Сколько времени и ресурсов может потребоваться на исправление?
4. Какая модель лучше защищает заказчика от таких ошибок?

Требования к отчёту

Структура отчёта

1. Титульный лист
2. Цель работы (переписывается из методички).
3. Задачи работы (переписываются из методички).

4. Теоретическая часть (краткий конспект основных определений: проект, процесс, продукт, роли, жизненные циклы – не более 1-2 страниц).

5. Практическая часть – выполнение заданий 1–4 в том порядке, как они даны.

6. Ответы на контрольные вопросы (переписать вопрос и дать чёткий ответ).

7. Вывод (2–4 предложения: что узнал, чему научился, какой кейс показался наиболее полезным).

Оформление практической части (обязательные требования)

Задание 1 – таблица из методички должна быть полностью заполнена символами «+» или «–» (можно использовать слова «да/нет»).

Задание 2 – написать название модели (водопад / итеративная) и обоснование (2-3 предложения на каждый кейс).

Задание 3 – заполнить таблицу: роль + конкретная задача на первую неделю (не общими фразами вроде «разработка», а именно: «написать план тестирования экрана регистрации» и т.п.).

Задание 4 – дать развёрнутые ответы на три вопроса по мини-кейсу (каждый ответ – минимум 1-2 предложения).

Технические требования

1. Шрифт: Times New Roman, 14 пт (для таблиц допускается 12 пт).

2. Межстрочный интервал: 1,5.

3. Поля: левое – 30 мм, правое – 15 мм, верхнее/нижнее – 20 мм.

4. Абзацный отступ (отступ первой строки) – 1,25 см.

5. Заголовки разделов – полужирные, выравнивание по ширине, абзацный отступ, нумерация (при необходимости).

6. Таблицы и рисунки (если есть) нумеруются и подписываются. Рисунок и подпись к рисунку выравниваются по центру. Подпись к рисунку располагается под ним, к таблице – над.

Контрольные вопросы

1. Перечислите три основных отличия проекта от процесса.
2. Приведите пример продукта, который может быть результатом как проекта, так и процесса (например, обновление документации).
3. Назовите не менее четырёх ролей в IT-команде и их основные функции.
4. В чём главный недостаток водопадной модели жизненного цикла ПО?
5. Что такое итерация в разработке ПО?
6. Почему для стартапов чаще выбирают итеративные подходы, а не водопад?
7. Кто в проекте несёт ответственность за приоритеты задач и коммуникацию с заказчиком?
8. Может ли один человек совмещать несколько ролей? Приведите пример (например, аналитик + тестировщик в малом проекте).

Практическая работа №2

Инициация проекта и командообразование

Цель работы: сформировать малые проектные команды, выбрать и обосновать идею IT-проекта, освоить базовые приёмы генерации идей (мозговой штурм) и распределения ролей в команде.

Задачи:

1. Освоить метод мозгового штурма для генерации идей IT-проектов.
2. Сформировать малые группы (3–4 человека) для совместной работы над проектом.
3. Выбрать и зафиксировать идею проекта, определить его основные цели и целевую аудиторию.
4. Распределить роли в команде согласно выбранному проекту.
5. Научиться создавать проектную доску в Yougile, добавлять задачи и колонки.
6. Подготовить стартовый бэклог задач на первую итерацию.

Теоретическая часть

1. Инициация проекта

Инициация – это первый этап жизненного цикла проекта, на котором:

- формулируется бизнес-идея или проблема,
- определяются предварительные цели и ограничения,
- оценивается целесообразность,
- назначается команда и менеджер.

Результатом инициации является **Устав проекта** (Project Charter) – документ, фиксирующий: название, цель, заказчика, ключевых участников, бюджет (если есть), сроки, критерии успеха.

Для IT-проектов на этапе инициации важно ответить на вопросы:

1. Какую проблему решает продукт?

2. Кто целевые пользователи?

3. Какие основные функции будут в MVP (Minimum Viable Product)?

2. Генерация идей: мозговой штурм

Мозговой штурм (brainstorming) – метод группового поиска решений, основанный на свободном высказывании любых идей без критики.

Правила классического мозгового штурма:

1. Запрещена любая критика (даже позитивная оценка – позже).

2. Поощряются любые, даже фантастические идеи.

3. Количество идей важнее качества.

4. Разрешено развивать и комбинировать чужие идеи.

Этапы:

– **Генерация** (10–15 мин): каждый участник по кругу называет идеи; ведущий фиксирует все на доске/флипчарте.

– **Анализ и отбор** (15–20 мин): группа обсуждает, отбрасывает заведомо нереализуемые, выбирает 1–3 лучших по критериям (актуальность, техническая сложность, интерес для команды).

Альтернативные техники (для справки): метод SCAMPER, ментальные карты, техника «6 шляп».

3. Формирование команд и распределение ролей

При формировании малых групп (3–4 человека) рекомендуется учитывать:

– разнообразие интересов и навыков (один силён в логике, другой – в дизайне, третий – в коммуникациях);

– отсутствие жёстких конфликтов между участниками;

– желание работать над общей идеей.

Роли в рамках учебного проекта могут быть совмещены (например, разработчик + тестировщик), но каждый член команды должен иметь зону ответственности. Базовый набор для небольшого проекта:

Роль	Ответственность в учебном проекте
Капитан (Project Manager)	Организация встреч, контроль сроков, связь с преподавателем, ведение доски в Yougile.
Аналитик/документатор	Описание требований, создание пользовательских историй, ведение документации.
Разработчик(и)	Реализация функциональности (код, база данных, интерфейс – в зависимости от компетенций).
Тестировщик	Составление чек-листов, проверка работоспособности, оформление багов.

В команде из 3 человек разработчиков может быть двое, а роли аналитика и тестировщика выполняет один человек. В команде из 4 человек можно выделить отдельного дизайнера/UI/UX.

4. Знакомство с Yougile

YouGile – это российская система управления проектами и задачами, которая объединяет в себе таск-трекер с корпоративным мессенджером и простой CRM (Customer Relationship Management, Система управления взаимоотношениями с клиентами), делая общение и управление задачами прозрачными и удобными, с акцентом на вовлечение всей команды в рабочий процесс.

К основным особенностям сервиса относятся:

– Коммуникации по средствам встроенного мессенджера с личными, групповыми чатами и чатами в каждой задаче;

– Более 50 инструментов для удобной работы на доске: дедлайн, исполнитель, ресурсы, свободные поля и др.;

– Бесконечное дерево подзадач для больших растущих процессов;

– Гибкая система прав – 108 опций для точной настройки ролей;

– Инструменты для связи разных отделов и прозрачной совместной работы – гибкие отчеты, настройка прав, возможность совместной работы с подрядчиками;

– Инструменты планирования – календарь, диаграмма Ганта, личный планировщик «Мои задачи»;

– CRM: Простая система для ведения сделок и работы с клиентами;

– Интеграция искусственного интеллекта для управления проектами (AI-инструменты).

– Развертывание: может работать как облачное решение, так и устанавливаться на собственный сервер (on-premise) для полной локальной работы.

YouGile работает на всех платформах, на любых устройствах.

Существуют различные версии системы:

1. Веб-версия, доступная по ссылке https://ru.yougile.com
2. Десктопная версия для Windows, macOS и Linux

Запуск десктопной версии вместе с операционной системой подталкивает каждого участника знакомиться с актуальными задачами и видеть общую картину по проекту.

Скачать последнюю версию десктоп-приложения можно по ссылке https://ru.yougile.com/download

3. Мобильное приложение YouGile Mobile – для работы с iOS и Android
4. Коробочная версия для Windows и Linux

Коробочным решением YouGile обеспечивает и максимальную конфиденциальность, и надёжность. YouGile устанавливается на сервер компании и полностью независим от облачной версии. У компании находится полный контроль над всеми данными, включая переписку, а также над политикой безопасности и системой резервного копирования. Также она становится независимой от сбоев, происходящих не по её вине.

Ещё один аспект безопасности – коробочная версия может работать в локальной сети без доступа в интернет.

Можно работать в коробочной версии, используя мобильное приложение.

При необходимости, можно сделать любые кастомные доработки, используя встроенный редактор кода (конфигуратор) или REST API.

Практическая часть

Задание 1. Мозговой штурм

Формат: вся академическая группа делится на микрогруппы по 4–5 человек (временные).

Тема: «IT-продукты, которые облегчают жизнь студентов или преподавателей».

Процесс:

Каждая временная группа генерирует не менее 10 идей (записывает на листе или в общем документе).

Выбирает одну самую интересную и реалистичную идею для дальнейшей разработки (учитывая объём работы на семестр).

Результат: каждая временная группа представляет выбранную идею (название, 1–2 предложения описания) вслух для всей группы.

Задание 2. Формирование постоянных команд

На основе результатов мозгового штурма и пожеланий студентов формируются постоянные команды по 3–4 человека. Каждая команда окончательно выбирает одну идею проекта (можно взять любую из предложенных в задании 1, включая не свою).

Действия команды:

1. Придумать название команды.

2. Записать список участников (ФИО).

Задание 3. Распределение ролей

Внутри постоянной команды обсудите и назначьте роли. Используйте таблицу:

ФИО	Роль	Ключевые обязанности
...	Капитан	...
...	Аналитик	...
...	Разработчик	...
...	Тестировщик/дизайнер	...

Фиксация: сохранить в общем документе команды (Google Docs или текстовый файл).

Задание 4. Регистрация в YouGile

Шаг 1. Регистрация в системе YouGile

Перед началов работы в системе управления проектами YouGile необходимо пройти процедуру авторизации. Для этого:

1. Открыть в браузере официальный сайт системы управления проектами YouGile: https://yougile.com

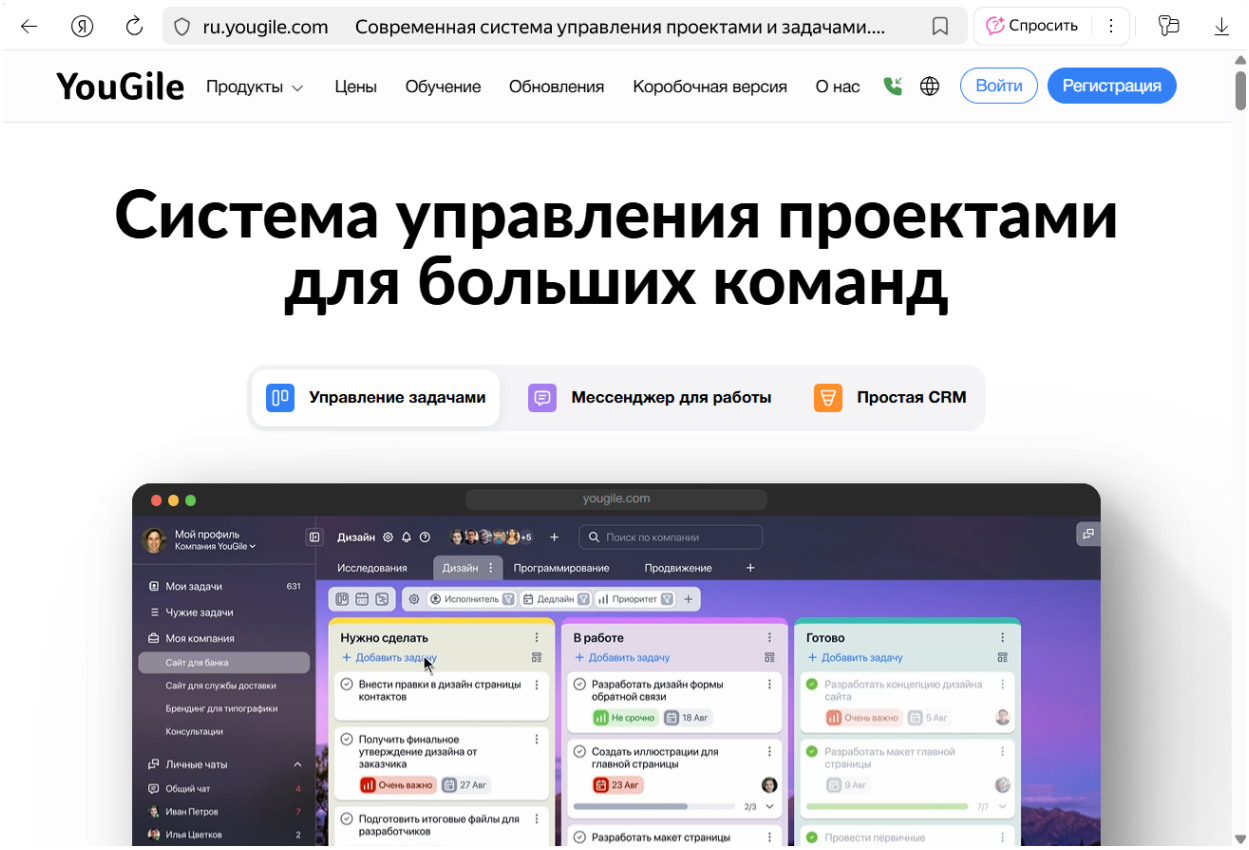


Рисунок 1 – Приветственная страница системы YouGile

2. В правом верхнем углу страницы нажмите кнопку **«Войти»**

На открывшейся странице будет предложено 2 сопособа входа в систему:

1) Войти через аккаунт Google, Яндекс ID или VK ID либо ввести логин и пароль при наличии:

Нажать кнопку «Войти с помощью Google», «Войти с помощью Яндекс ID» или «Войти с помощью VK ID» (рисунок 2).

В появившемся окне выбрать свой аккаунт и разрешить доступ приложению YouGile.

Система автоматически создаст аккаунт, используя данные из выбранного профиля Google, Яндекс или VK.

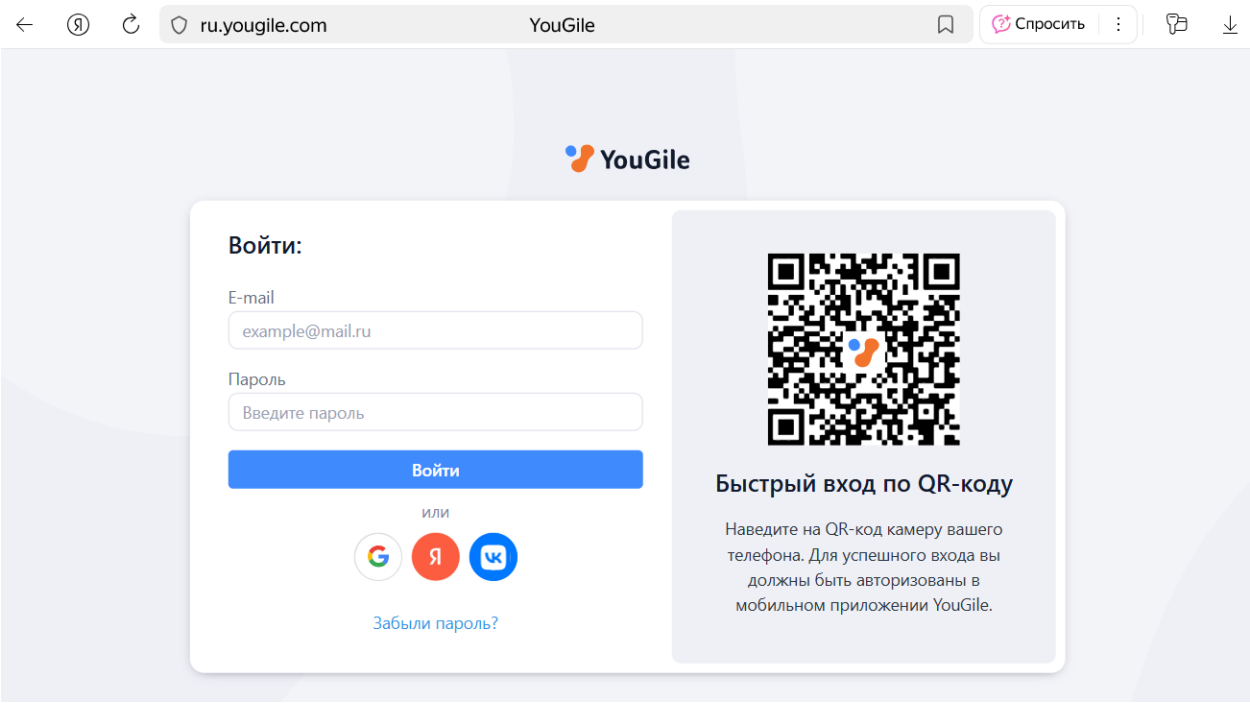


Рисунок 2 – Страница входа в систему

2) Пройти процесс регистрации с помощью email-адреса:

В поле «Электронная почта» ввести адрес рабочей электронной почты, нажать кнопку «Регистрация» (рисунок 3).

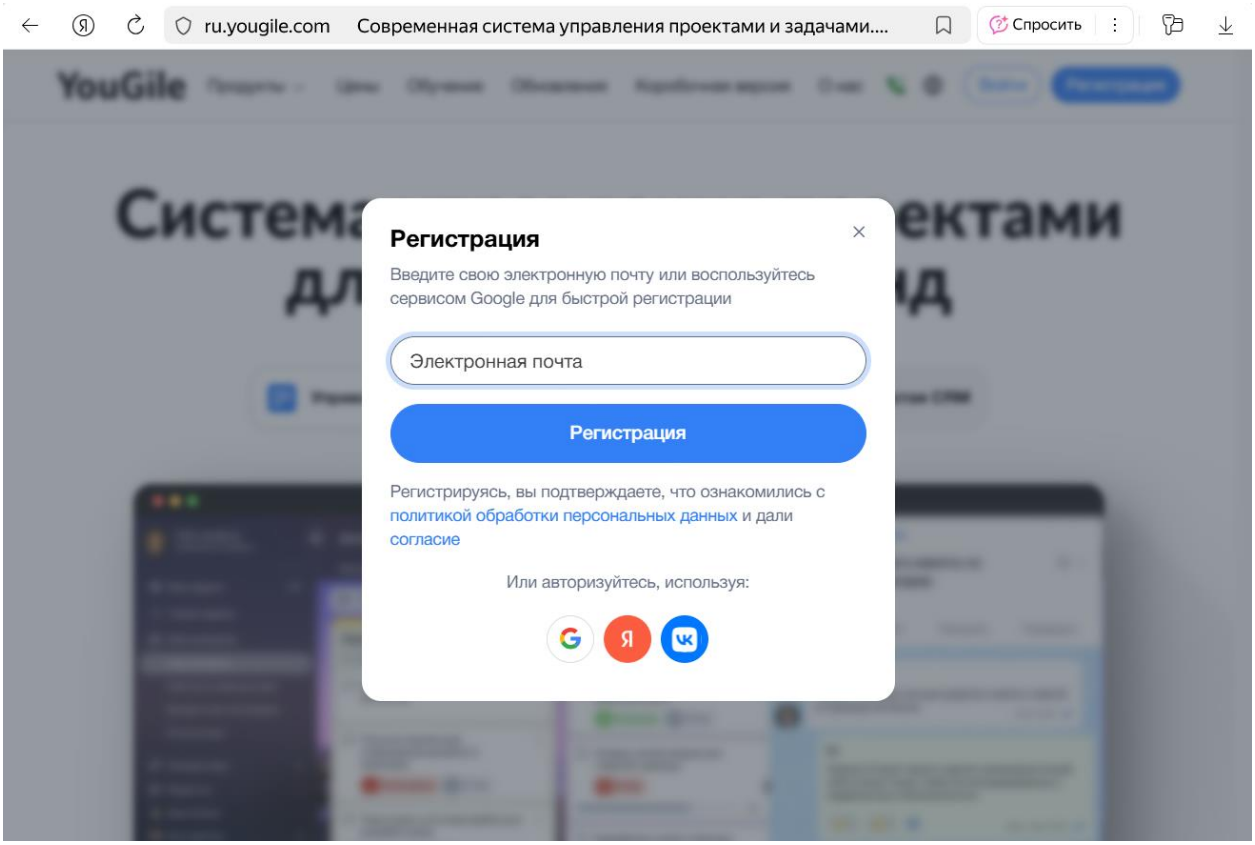


Рисунок 3 – Форма регистрации в систему

После этого действия на указанный адрес электронной почты будет направлено письмо для подтверждения регистрации пользователя.

Для прохождения процедуры подтверждения необходимо перейти в свой почтовый клиент (Gmail, Mail.ru, Яндекс.Почта и т.д.), найдите письмо от YouGile и перейдите по ссылке, после чего произойдёт переадресация на страницу для заполнения данных о пользователе.

В открывшемся окне вводится имя, название компании, номер телефона пользователя и придумывается пароль для будущих входов в систему (рисунок 4).

После чего нажимается конпка «Войти в систему» и происходит переход в созданное рабочее пространство.

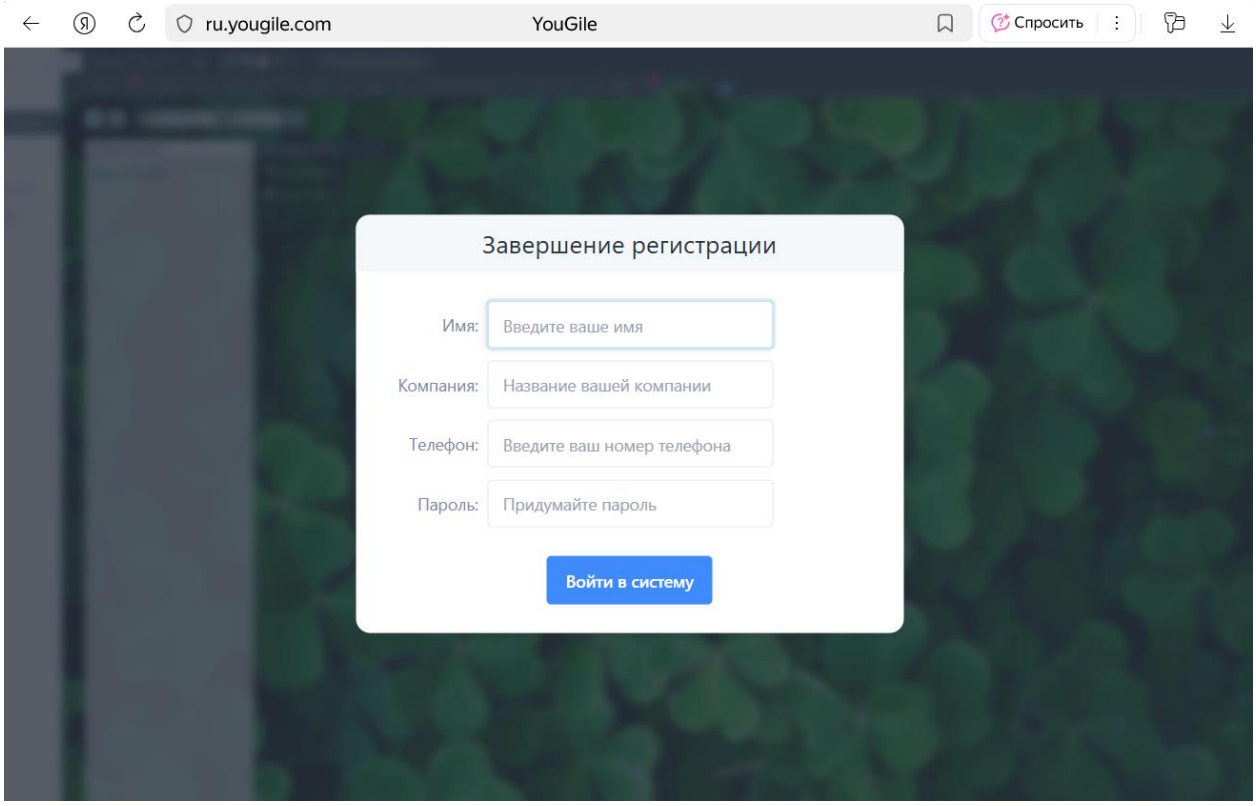


Рисунок 4 – Окно завершения регистрации в систему YouGile

Шаг 2. Изучение интерфейса системы

В YouGile реализована иерархия объектов, позволяющая строить работу компаний любой сложности. В компании может быть любое количество

проектов, в проекте – любое количество досок. На каждой доске можно создать множество колонок с неограниченным количеством задач.

После входа в систему вы попадаете на главную страницу созданной компании. Ознакомимся с её структурой:

1. Левая боковая панель (рисунок 5):

– «Мой профиль»;
– «Моя компания»;
– «Проекты»;
– «Мессенджер»;
– «Лента событий»;
– «Отчёты»;
– «Лицензия и оплата»;
– «Поддержка, Новости».

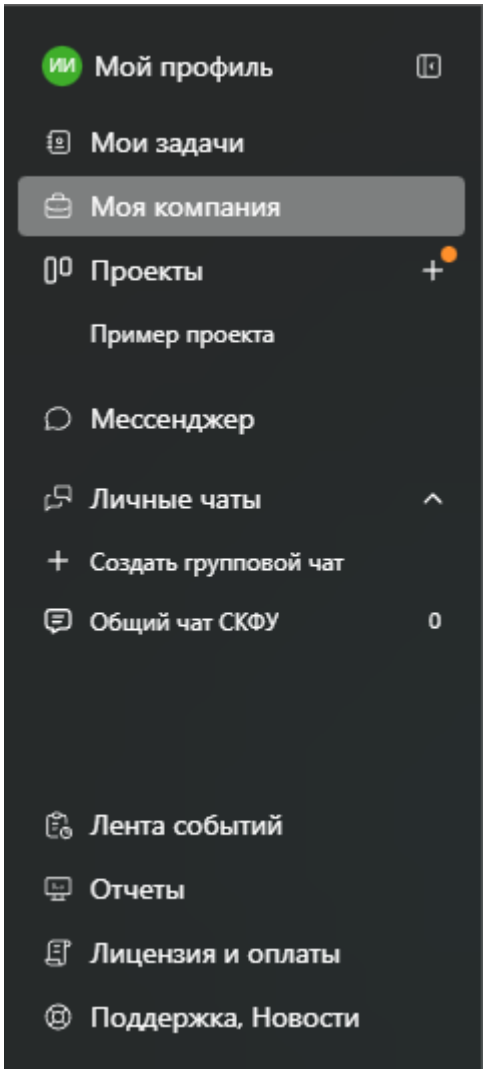


Рисунок 5 – Левоя боковая панель системы YuoGile

2. Основная центральная область:

Раздел «Мой профиль»

Содержит основную информацию о пользователе, ссылки на скачивание приложения YouGile для разных платформ, позволяет провести настройку уведомлений для чатов и внещнего вида системы, ознакомиться со списком компаний (рисунок 6).

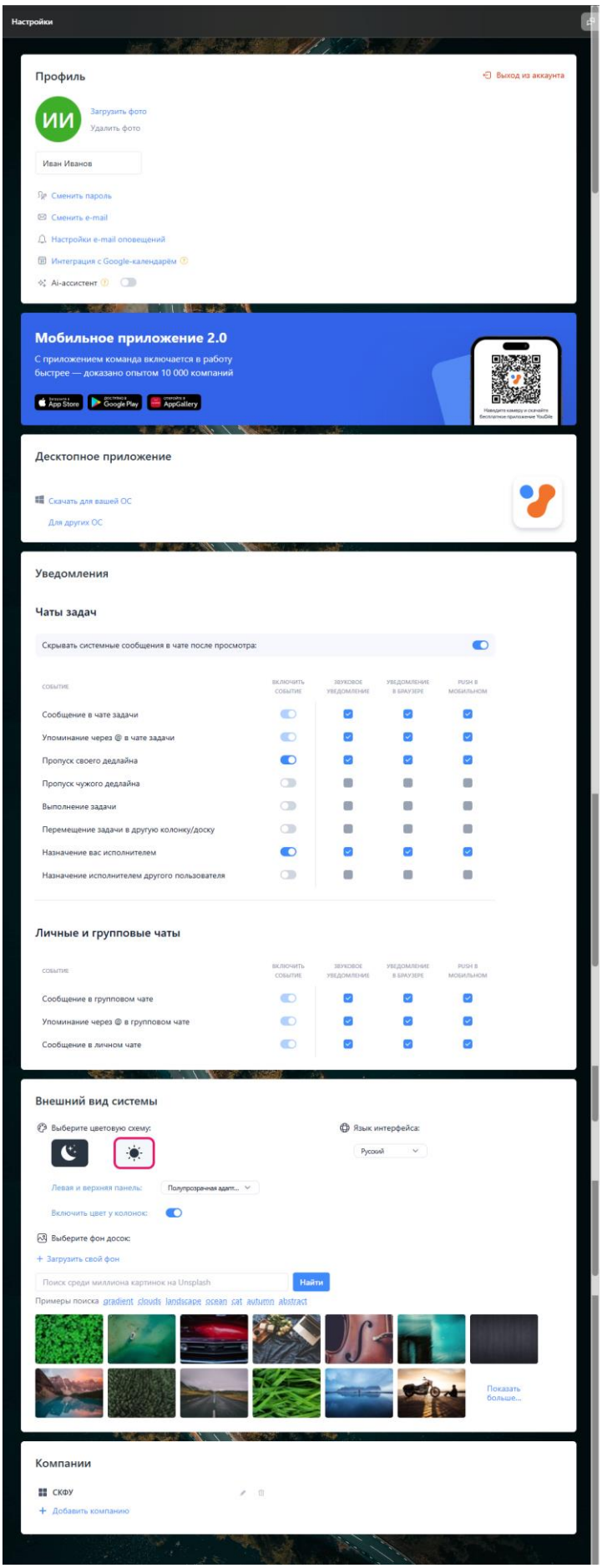


Рисунок 6 – Основная центральная область раздела «Мой профиль»

Блок «Профиль» (рисунок 7) позволяет изменить основную информацию о пользователе, загрузить фото профиля, настроить e-mail оповещения и интеграфию с Google-календарём, включить AI-асситента и выйти из аккаунта.

Интеграция с Google-календарем передает данные только в одном направлении: из YouGile в Google-календарь.

При включении интеграции, события будут создаваться в Google-календаре на основе задач с заполненным стикером **Дедлайн**.

В настройках вы можете выбрать, переносить все задачи или только те, которые назначены на вас.

После включения интеграции, события в Google-календаре будут автоматически обновляться.

Синхронизируются:

– название задачи,

– дата дедлайна,

– путь до задачи в YouGile,

– исполнители указываются в Google-календаре, как участники события.

При включении Ai-ассистента появляется пункт в левой панели, позволяющий открыть окно чата с ai-ассистентом.

Ассистент может выполнять действия в системе на стороне приложения пользователя, можно задавать вопросы по проекту и просить сгенерировать / изменить данные в проектах.

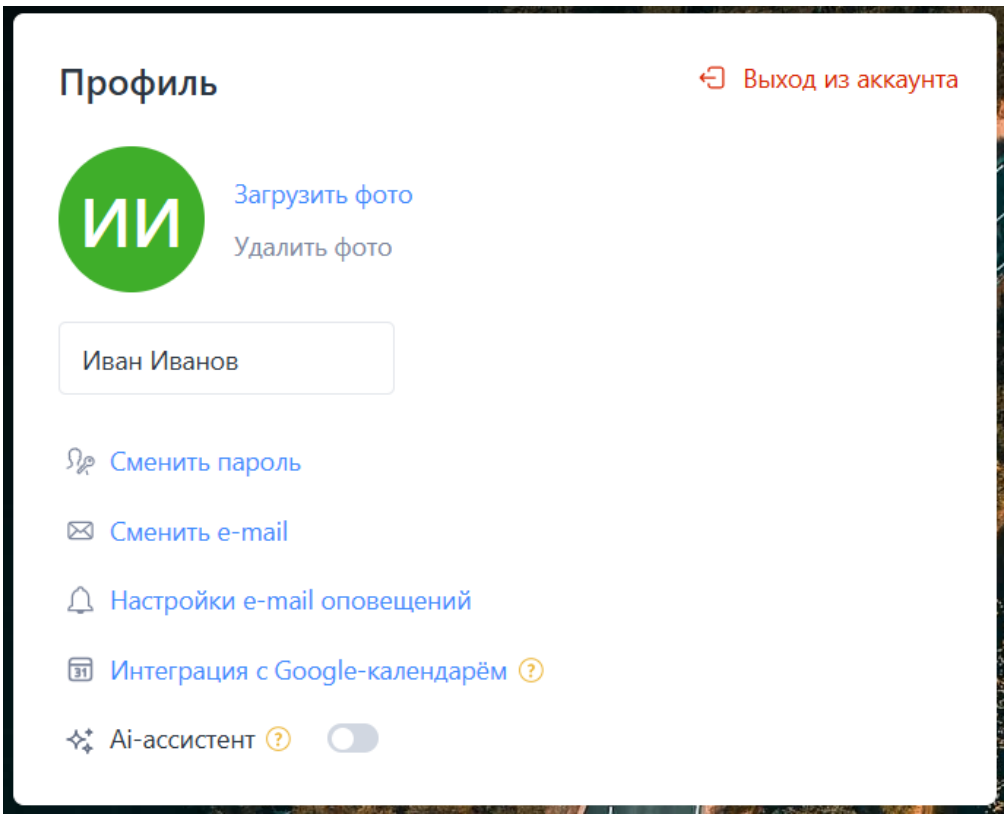


Рисунок 7 – Блок «Профиль» в разделе «Мой профиль»

Блок «Уведомления» даёт возможность детально настроить уведомления для чатов задач и личных и групповых чатов (рисунок 8).

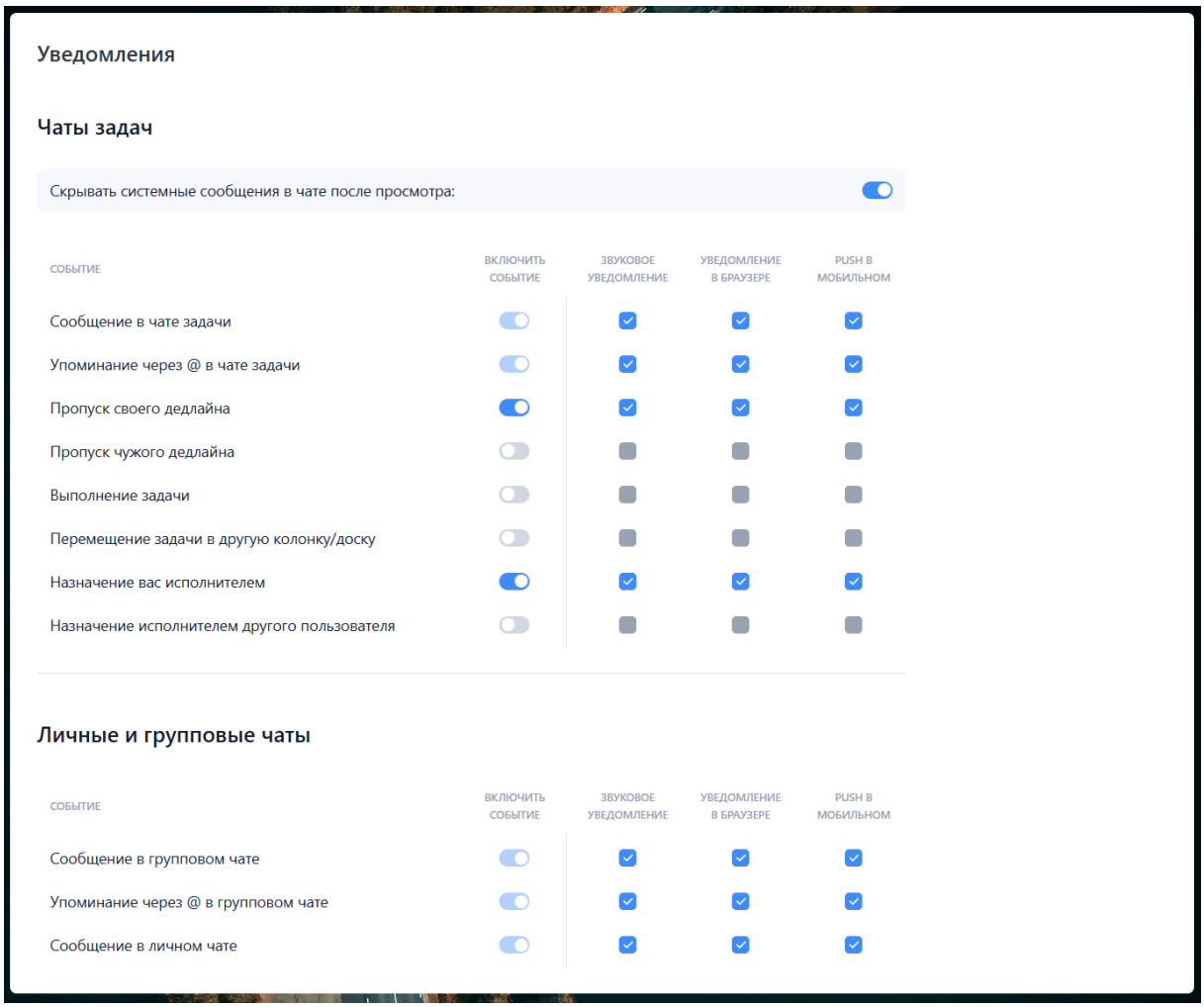


Рисунок 8 – Блок «Уведомления» в разделе «Мой профиль»

Блок «Внешний вид системы» для настройки оформления системы – выбора темы, внешнего вида левой и верхней панелеи, языка интерфейса и фона досок (рисунок 9).

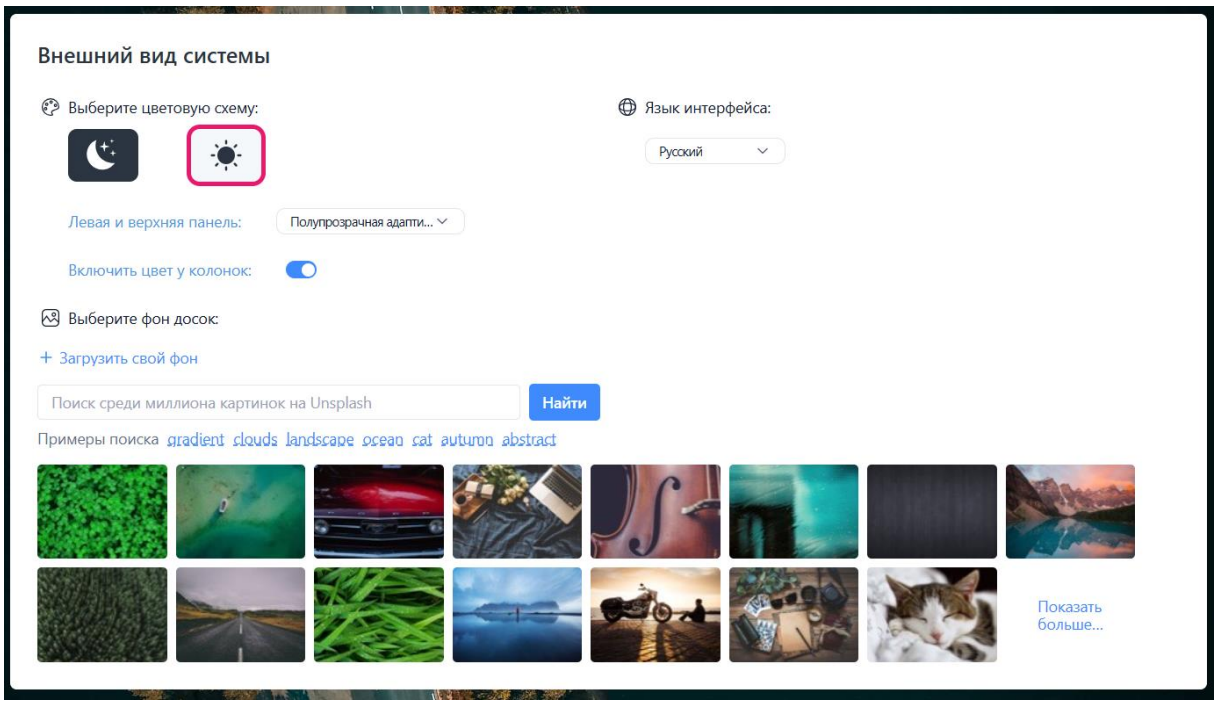


Рисунок 9 – Блок «Внешний вид системы» в разделе «Мой профиль»

Блок «Компании» показывает список компаний пользователя и даёт возможность добавить компанию (рисунок 10).

В каждом аккаунте можно создать несколько компаний.

Для создания новой компании и переключения между компаниями нужно нажать на «Добавить компанию».

В этом же разделе компанию можно переименовать, нажав на иконку с карандашом, а также удалить через значок с корзиной. Изменить порядок отображения компаний можно путем перетаскивания мышью.

Приглашенный в компанию пользователь, не являющийся администратором, не может удалить компанию у себя. Чтобы выйти из компании, ему нужно обратиться к администратору.

Также есть следующее ограничение: администратор не может удалить у себя последнюю, единственную компанию. Это можно сделать только через запрос в поддержку YouGile.

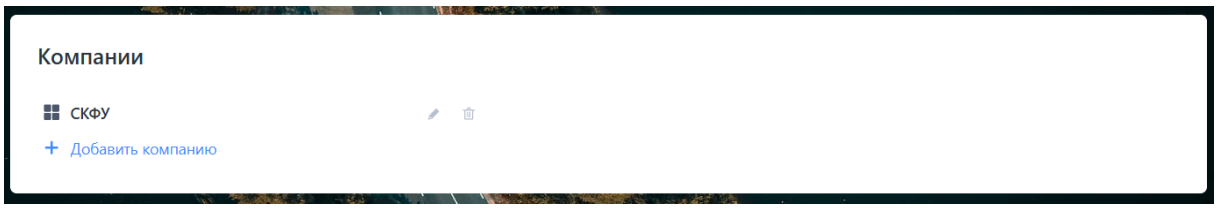


Рисунок 10 – Блок «Компании» в разделе «Мой профиль»

Раздел «Мои задачи»

Здесь отображаются все задачи пользователя – задачи, порученные ему (вкладка «Мои задачи»), задачи, порученные данным пользователем другим (вкалдка «Порученные мной»), а также избранные задачи и чужие задачи в соответствующих вкладках (рисунок 11).

На вкладке «Мои задачи» есть возможность создавать списки, куда будут рассортировываться поступающие задачи. Можно добавить обычный список и приватный список, отличающийся от обычного тем, что это спсиок будет виден только данному пользователю на вкладке «Чужие задачи».

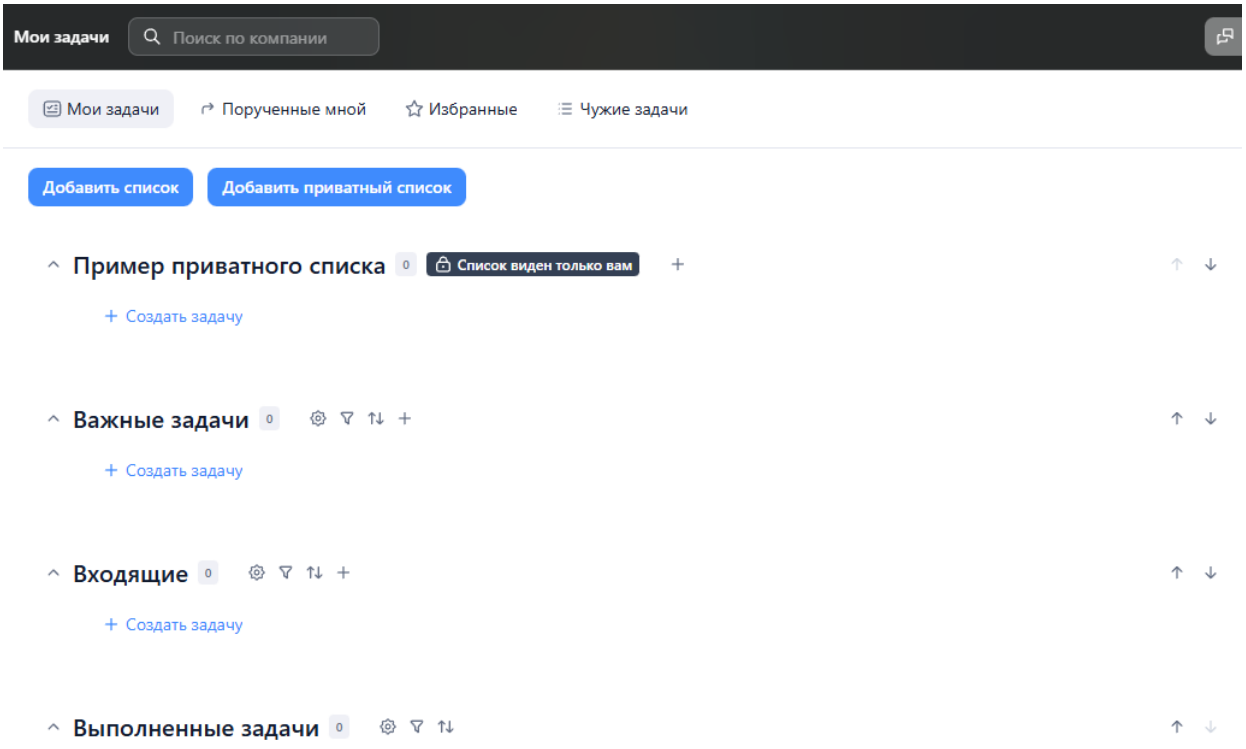


Рисунок 11 – Основная центральная область раздела «Мои задачи»

Раздел «Моя компания»

Данный раздел содержит основную информацио о компании (рисунок 12).

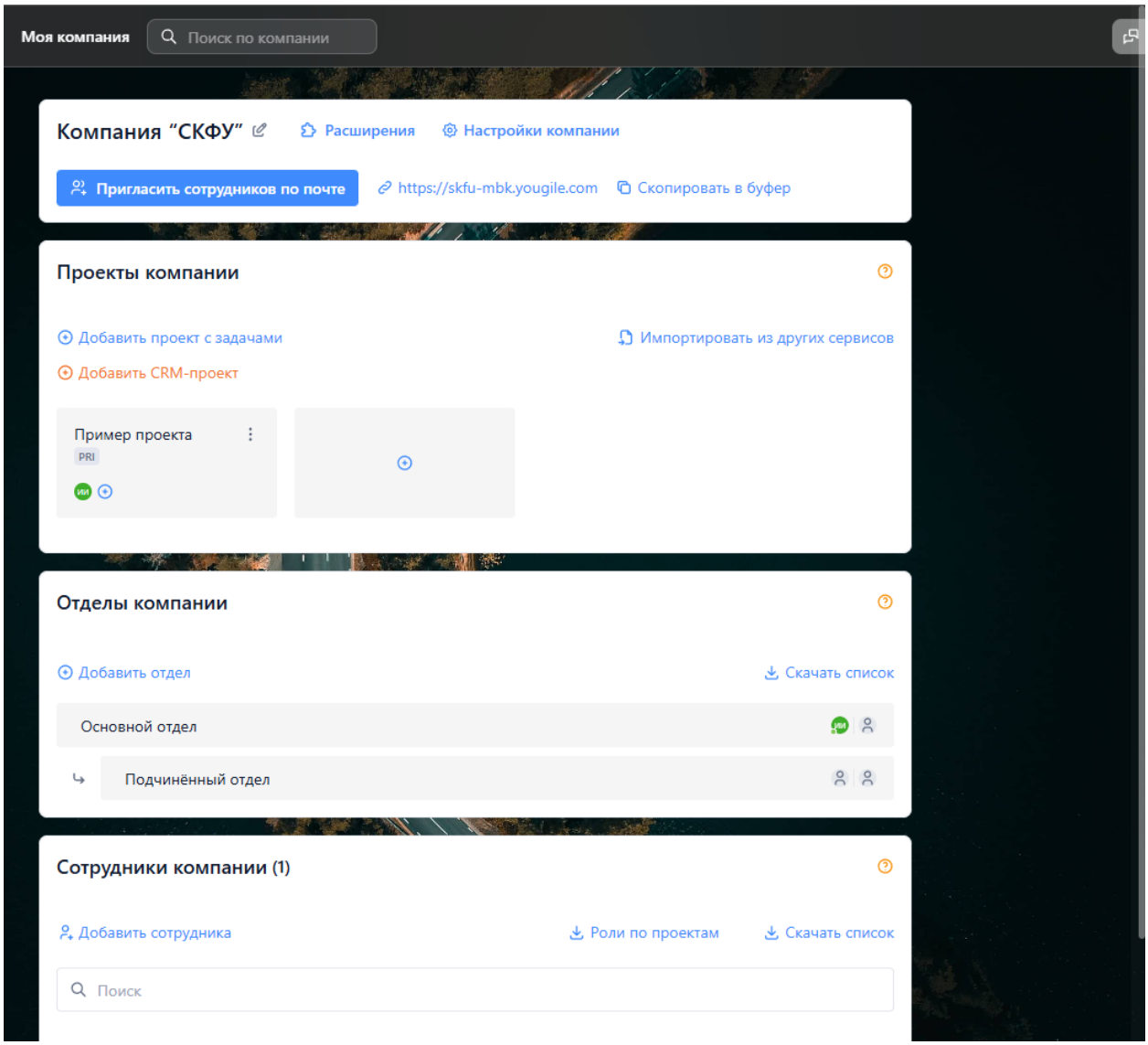


Рисунок 12 – Основная центральная область раздела «Моя компания»

Блок с название компании (рисунок 13) содержит несредственно название кампании, даёт возможность пригласить других пользователей по почте или по средствам ссылки на компанию, которую можно скопировать.

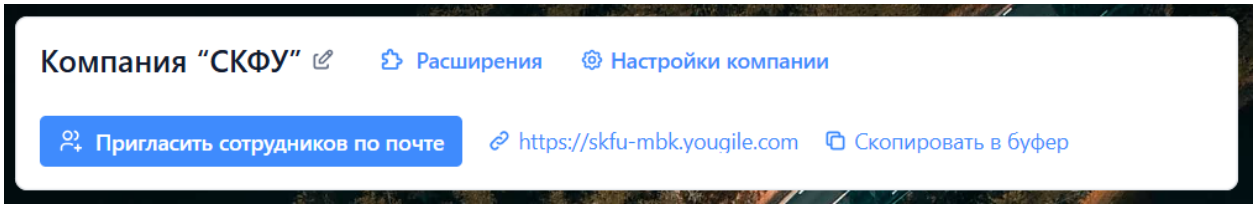


Рисунок 13 – Блок с названием компании в разделе «Моя компания»

Также есть возможность установить различные расширения из галерии расширений (рисунок 14), которую можно открыть нажатием кнопки «Расширения».

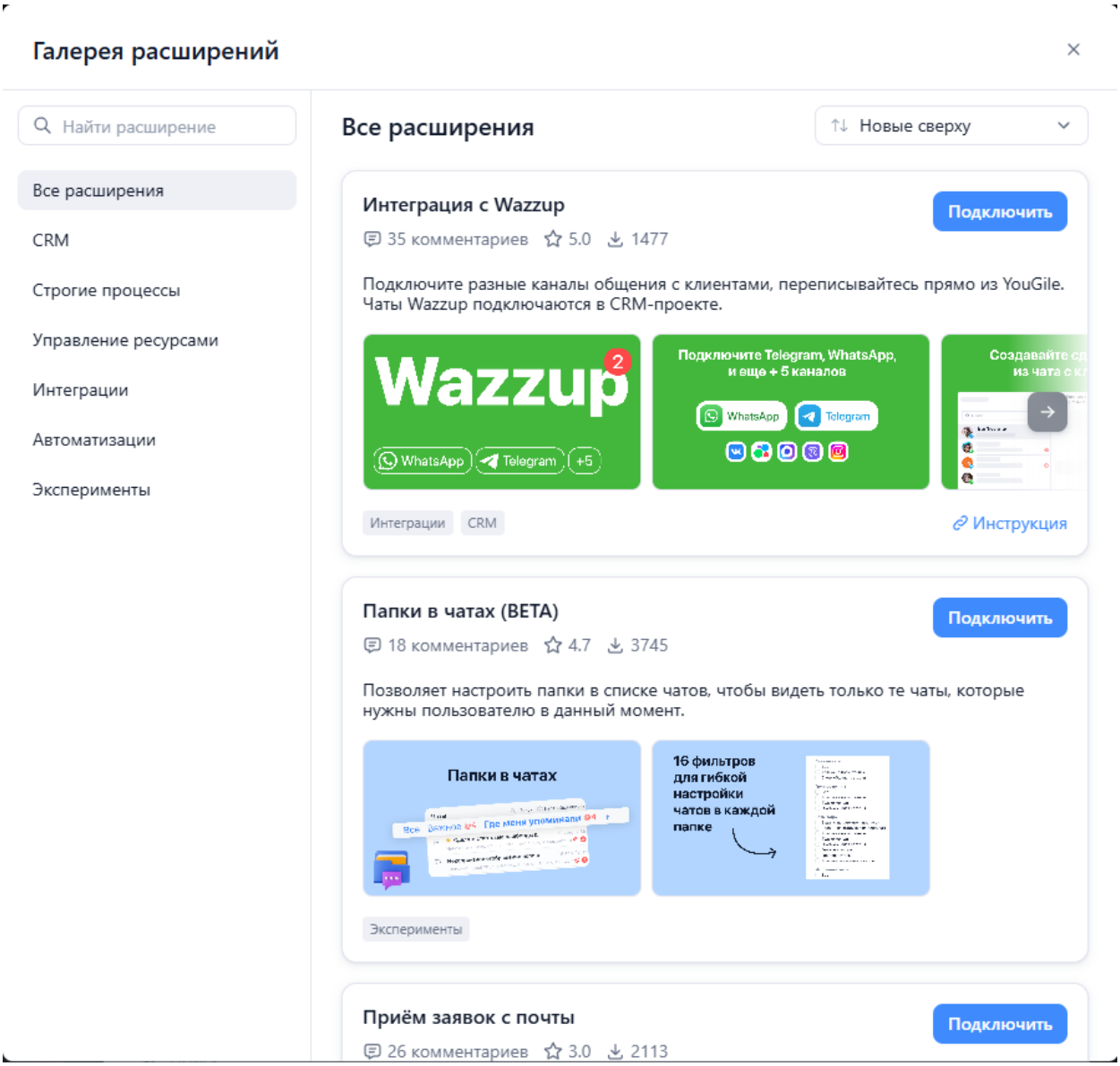


Рисунок 14 – Галерея расширений

Кнопка «Настройки компании» открывает окно для изменения настроек компании (рисунок 15).

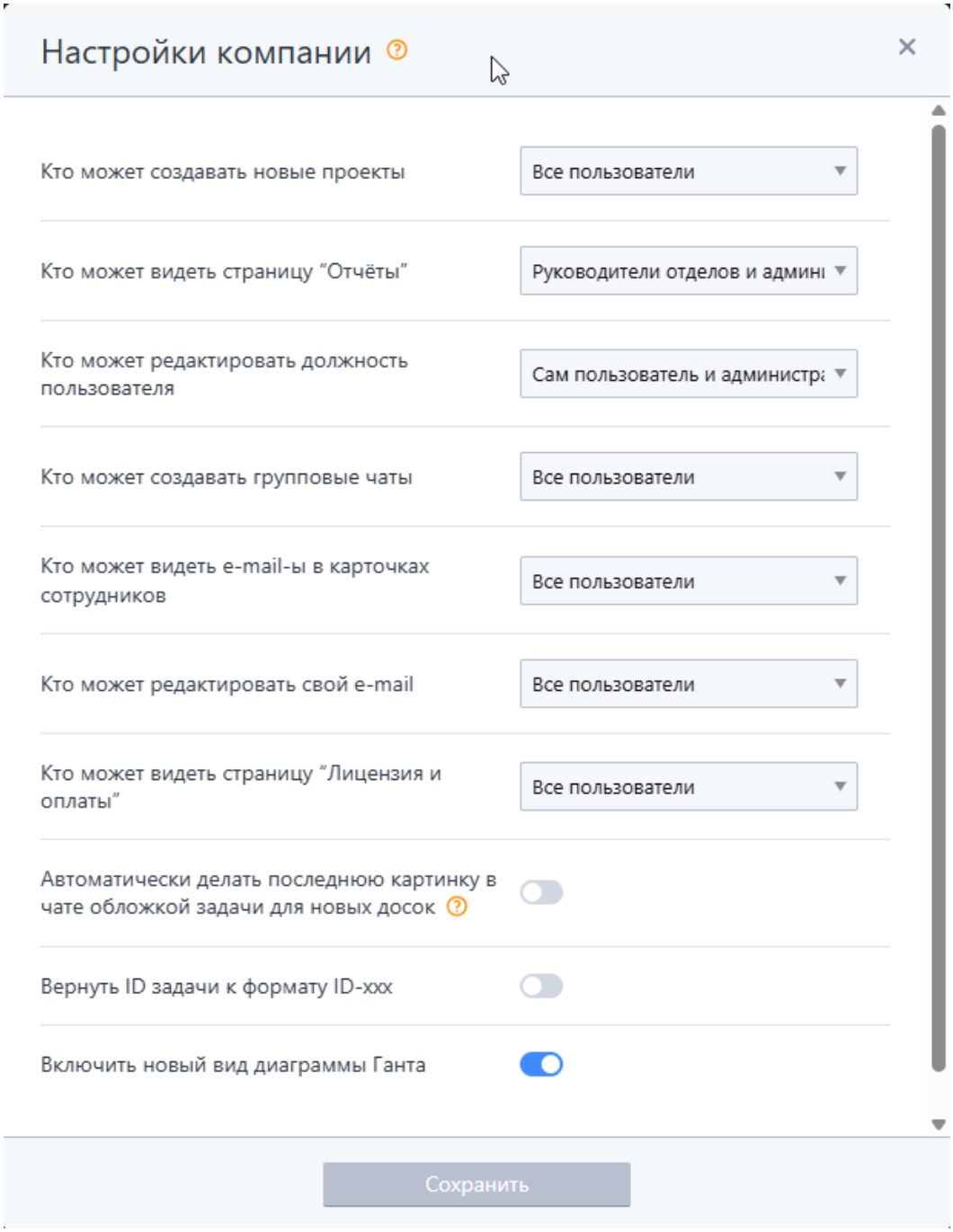


Рисунок 15 – Окно настроек компании

Блок «Проекты компании» отображает набор всех проектов компании (рисунок 16).

Есть возможность добавить два вида проектов: проект с задачами и CRM-проект.

Также есть возможность импортировать проекты из других источников – Trello, CSV-файлы с проектами или CSV-файлы со сделками в CRM.

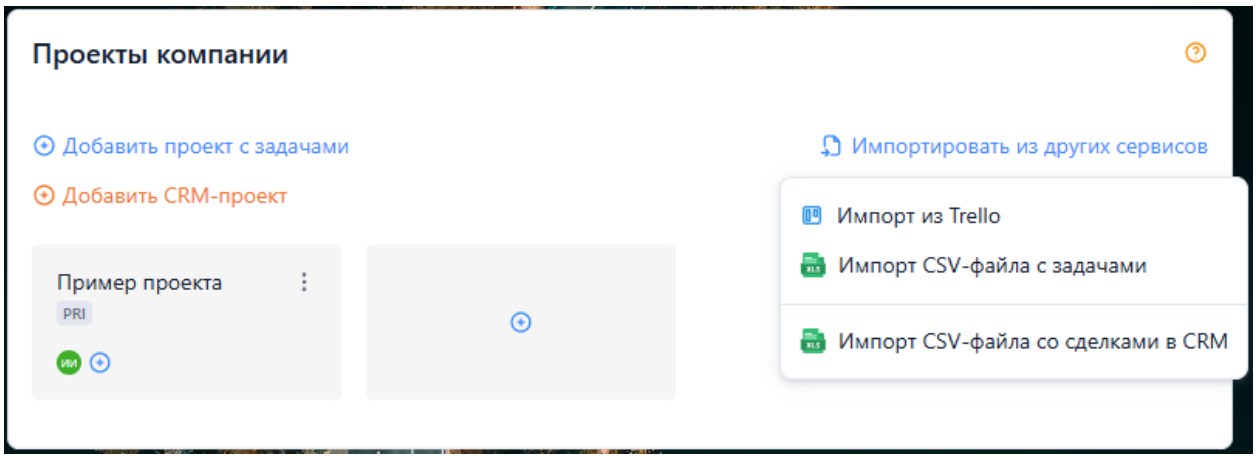


Рисунок 16 – Блок «Проекты компании» в разделе «Моя компания»

Блок «Отделы компании» информирует обо всех основных и подчинённых отделах (рисунок 17). Даёт возможность добавить отделы и скачать общий список отделов.

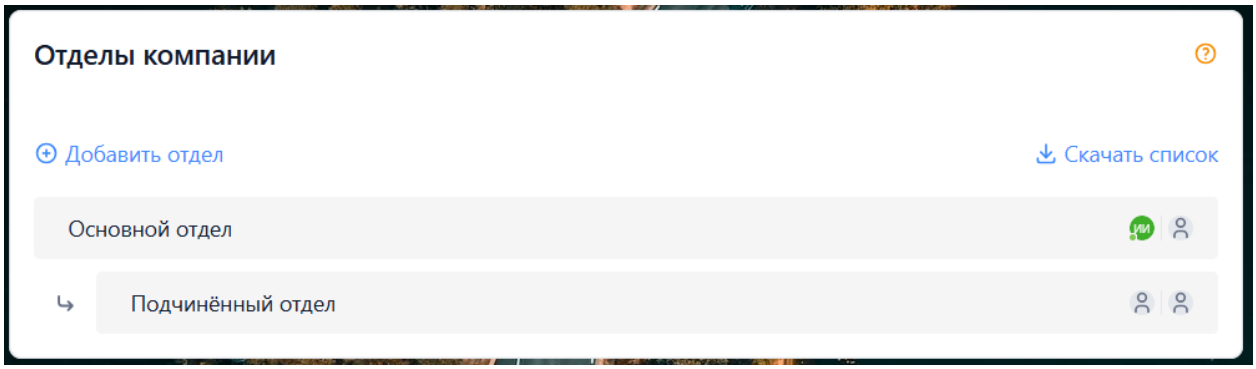


Рисунок 17 – Блок «Отделы компании» в разделе «Моя компания»

Блок «Сотрудники компании» отображает спсиок всех сотрудников компании (рисунок 18). Даёт возможность добавить сотрудника, скачать список ролей сотрудников по проектам и общий список сотрудников с указанием.

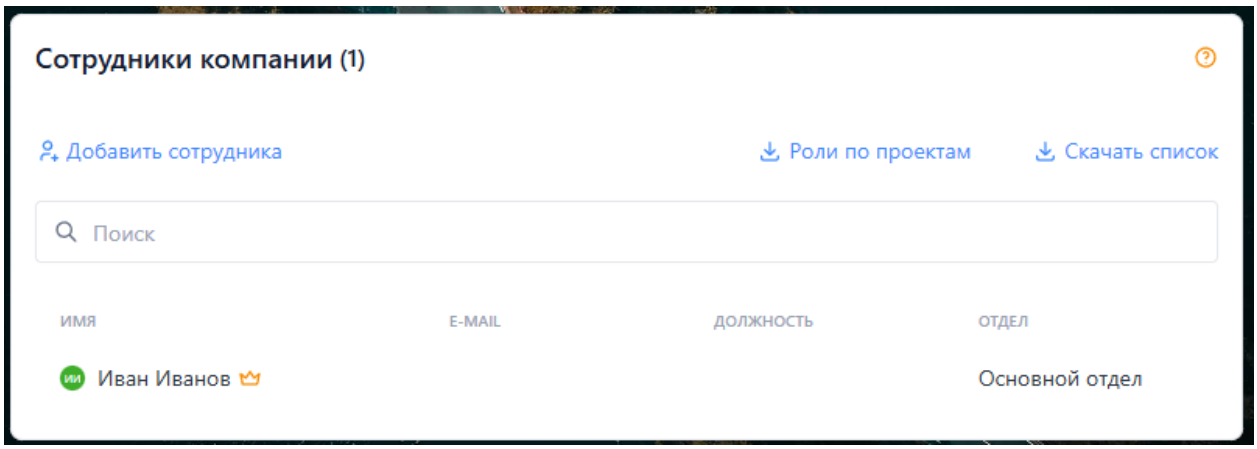


Рисунок 18 – Блок «Сотрудники компании» в разделе «Моя компания»

Раздел «Проекты»

В левой панеле данного раздела пеерчислены те же проекты, что и в блоке «Проекты компании» из раздела «Моя компания».

Создать проект можно двумя способами:

1. Как было показано выше в блоке «Проекты компании» в разделе «Моя компания» (рисунок 16).

2. С помощью пиктограммы + напротив названия раздела «Проекты», при нажатии на которую появляется список доступных для создания объектов, среди которых есть варианты проектов (рисунок 19):

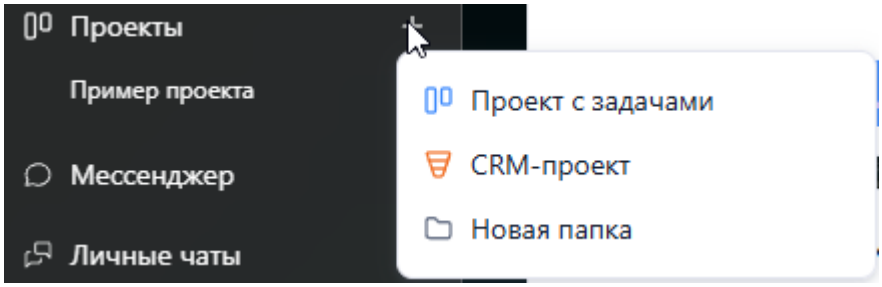


Рисунок 19 – Меню выбора типа проекта

После выбора нужного типа проекта открывается окно создания проекта. В нём можно изменить тип выбранного проекта, необходимо указать название проекта, ID-префикс проекта (создаётся автоматически, можно изменить вручную), поставить отметку о создании группового чата, если

таковой требуется и добавить участников, настроив им роли в данном проекте (рисунок 20).

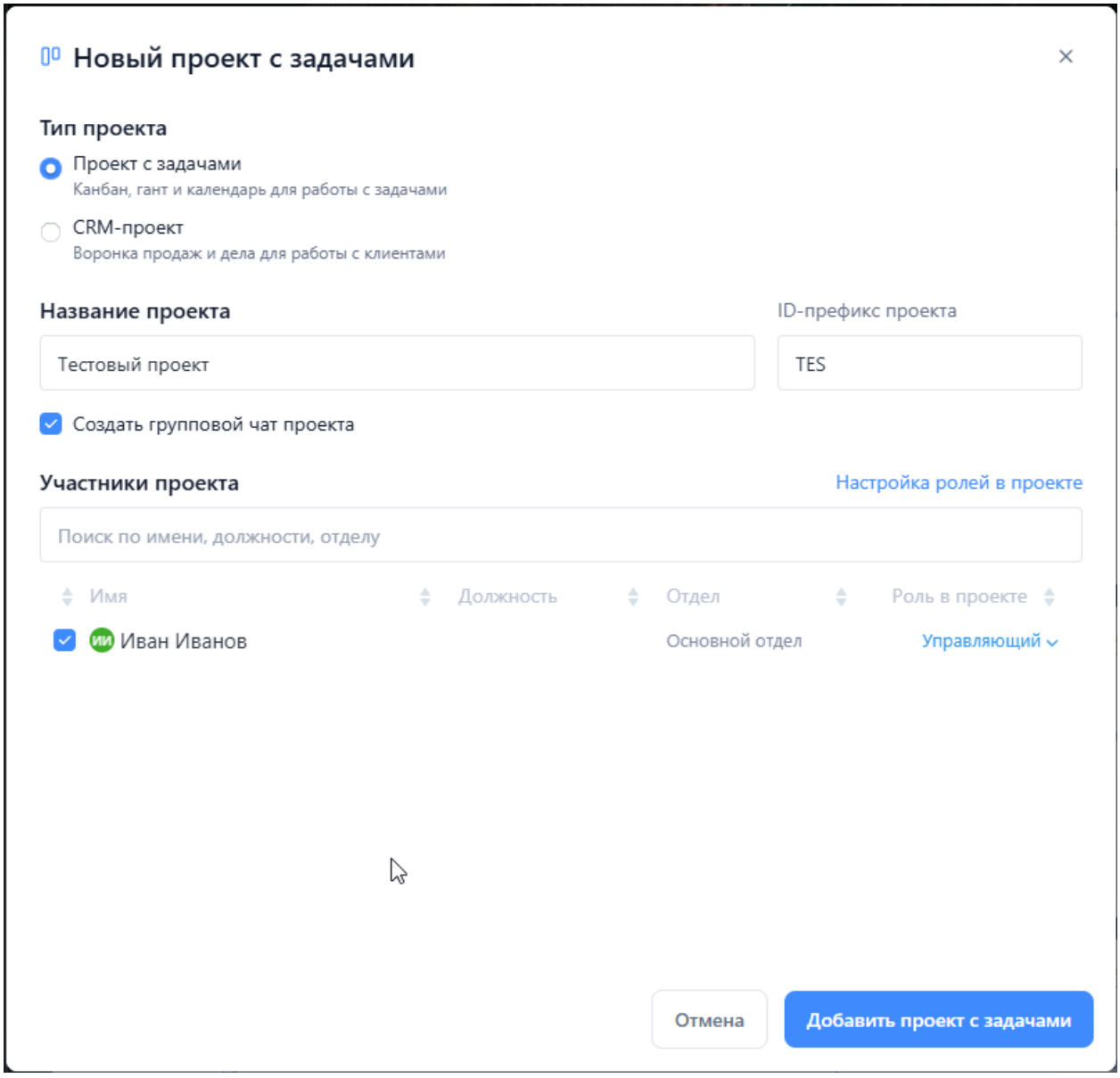


Рисунок 20 – Окно создания нового проекта

На вкладке «Настройка ролей в проекте» можно провести детальную настройку ролей для участников проекта: добавить роль, определив её название, описание и права доступа (рисунок 21), или копировать роли из другого проекта, выбрав из спсика нужный.

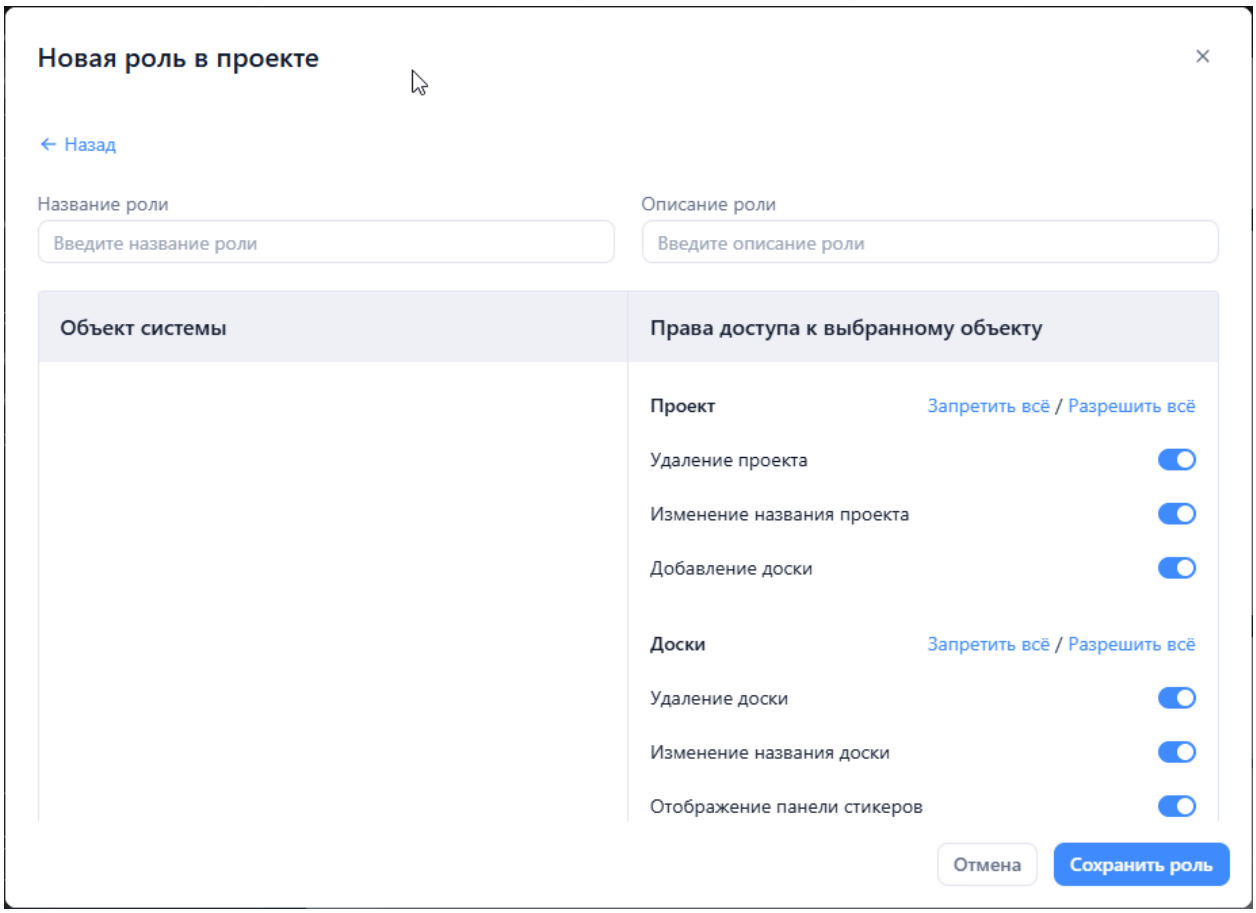


Рисунок 21 – Окно добавления новой роли в проект

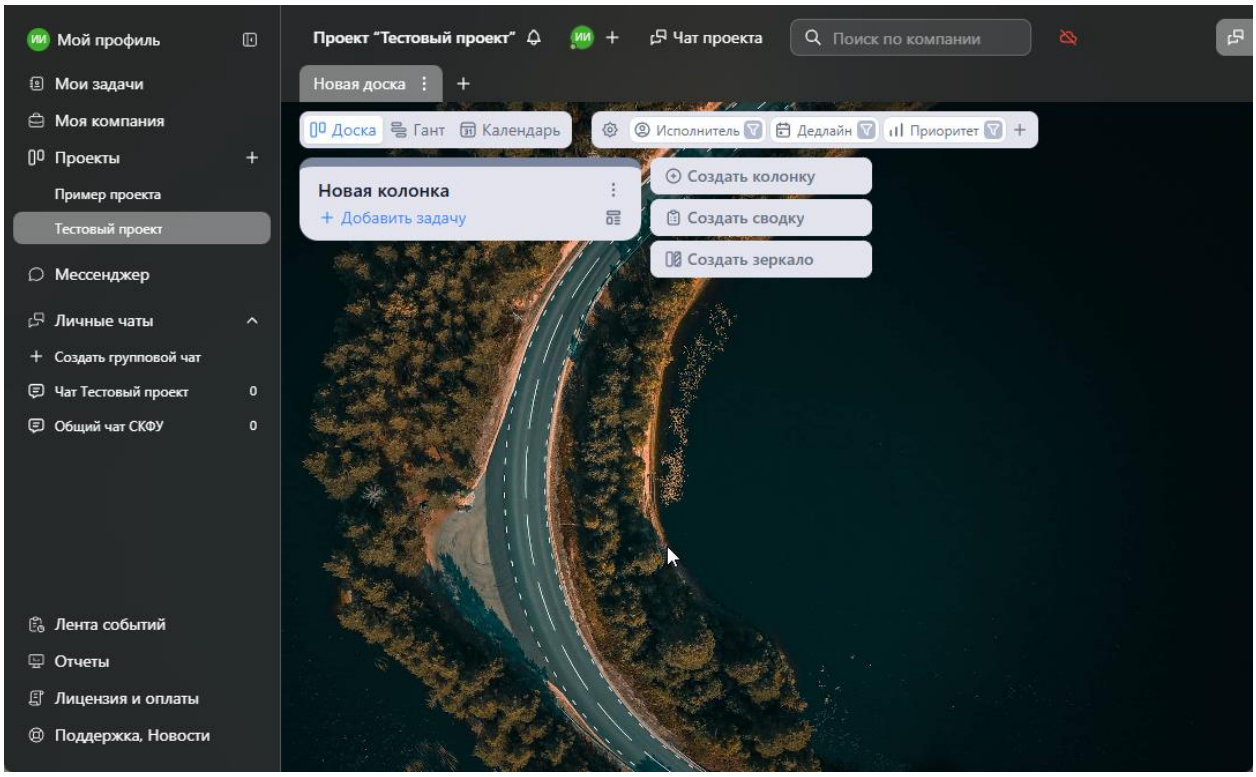


Рисунок 22 – Основная центральная область раздела «Проекты»

Раздел «Мессенджер»

В данной разделе отображается список всех чатов (рисунок 23).

Для быстрого доступа список всех чатов также отображается в левой боковой панеле в разделе «Личные чаты».

Доступ к чатам можно также получить с помощью кнопки  , расположенной в правом верхнем углу страниц (кроме страницы «Мессенджер»).

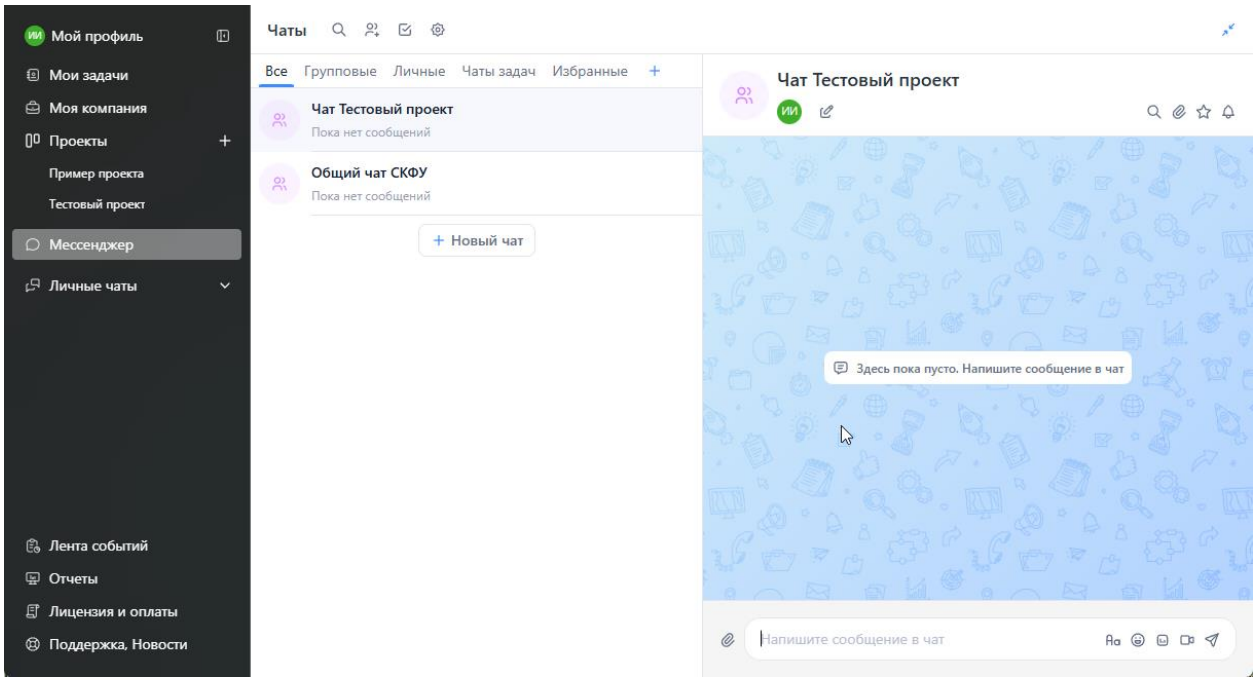


Рисунок 23 – Раздел «Мессенджер»

Раздел «Лента событий»

Отображает все действия, котоые производил пользователь в своём аккаунте (рисунок 24).

Позволяет проводить фильтрацию по событиям, а также скачивать таблицу, отражающую все события.

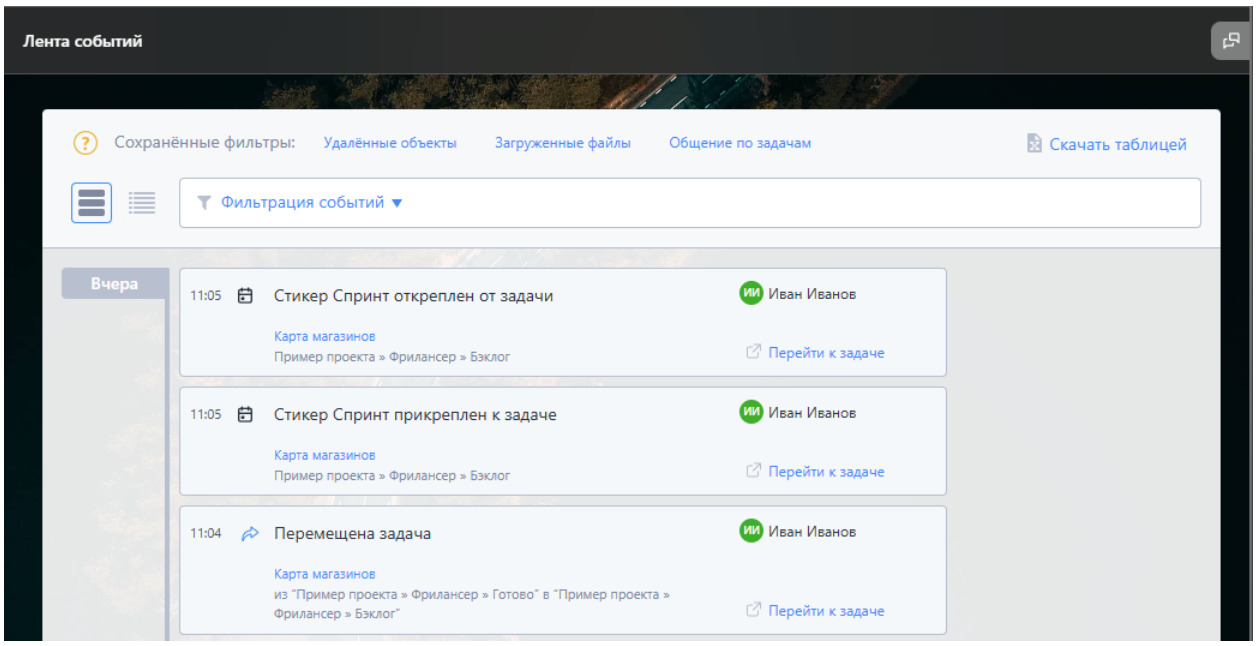


Рисунок 24 – Основная центральная часть раздела «Лента событий»

Раздел «Отчёты»

Данный раздел предоставляет возможность создания нескольких типов простых отчётов по результата работы над проектами (рисунок 25).

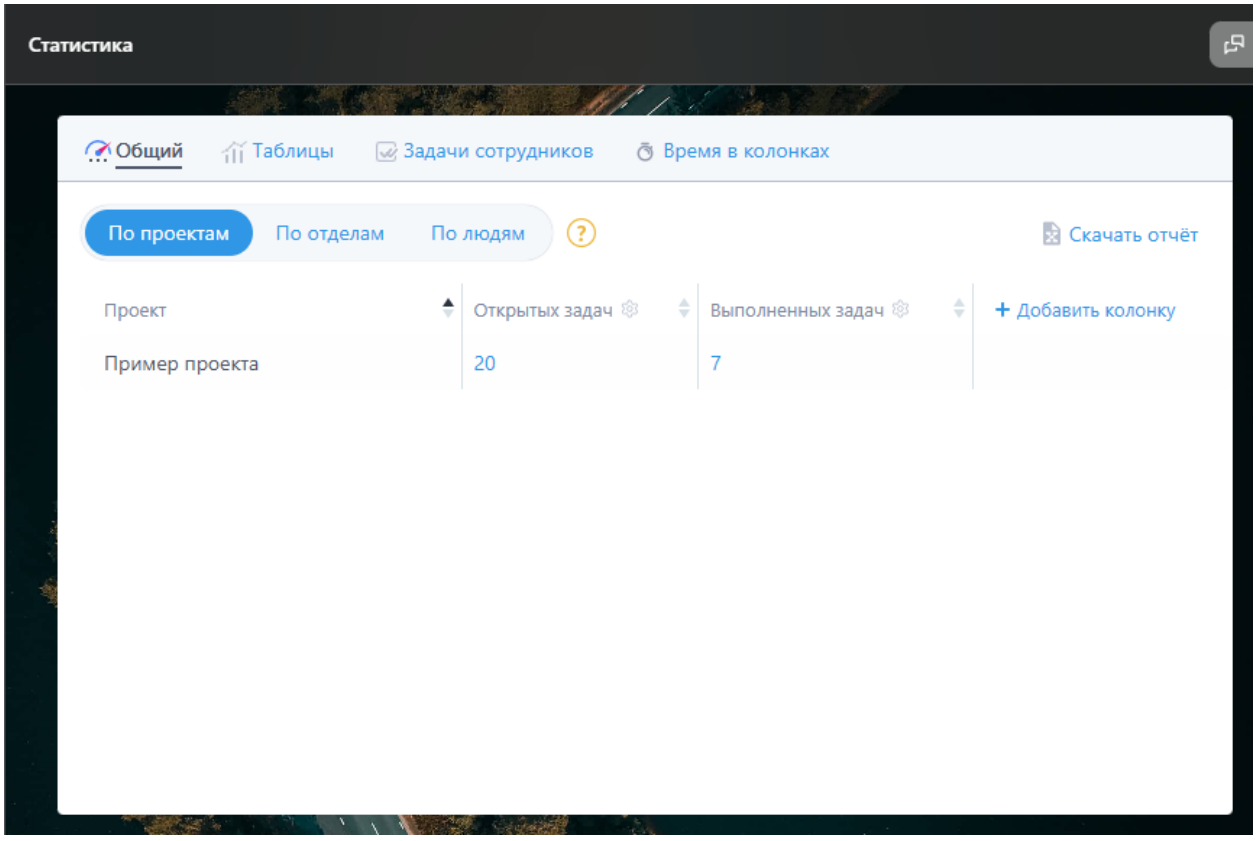


Рисунок 25 – Основная центральная часть раздела «Отчёты»

Задание 4. Создание доски в Yougile

1. Создайте новую доску: назовите её «[Название команды] – [Название проекта]» (например, «Кодеры – Расписание в один клик»).

2. Настройте колонки (статусы):

– Бэклог (Backlog)

– В работе (In Progress)

– На проверке (Review)

– Готово (Done)

3. Добавьте участников команды: пригласите сокомандников по email или отправьте ссылку-приглашение. Каждый должен получить доступ к доске.

4. Создайте первые 5–7 карточек задач на основе выбранной идеи (например, для приложения-расписания):

– «Собрать требования к расписанию» (аналитик)

– «Нарисовать прототип экрана» (дизайнер/разработчик)

– «Настроить доску Yougile» (капитан)

– «Создать репозиторий кода» (разработчик)

– «Написать тест-кейсы для формы входа» (тестировщик)

5. Для каждой карточки укажите:

– исполнителя (роль или конкретного человека),

– примерный дедлайн (через 2–3 дня),

– приоритет (высокий/средний).

Задания для выполнения

1. Зарегестрироваться в системе управления проектами YouGile любым из приведённых способов.

2. При регистрации дать имя копании в формате **«ФИО_Учебные проекты»** (например, **«Иванов А.А._Учебные проекты»**).

3. Изучить интерфейс системы управления проектами YouGile.

4. Создать проект с названием **«Практические работы_ФИО»** (например, **«Практические работы_Иванов А.А.»)**.

5. Продемонстрировать преподавателю процесс выполнения задания.

Требования к отчёту

Отчёт сдаётся каждой командой в виде краткого документа. Индивидуально отчитываться не требуется, но преподаватель может проверить вклад каждого участника.

Структура отчёта

1. Титульный лист
2. Цель работы (переписывается из методички).
3. Задачи работы (переписываются из методички).
4. Краткое описание выбранной идеи проекта (проблема, решение, целевая аудитория, ключевые функции MVP).
5. Скриншот главной страницы Yougile.
6. Вывод (что удалось сделать на этапе инициации, какие трудности возникли при распределении ролей или генерации идей).

Технические требования

1. Шрифт: Times New Roman, 14 пт (для таблиц допускается 12 пт).
2. Межстрочный интервал: 1,5.
3. Поля: левое – 30 мм, правое – 15 мм, верхнее/нижнее – 20 мм.
4. Абзацный отступ (отступ первой строки) – 1,25 см.
5. Заголовки разделов – полужирные, выравнивание по ширине, абзацный отступ, нумерация (при необходимости).
6. Таблицы и рисунки (если есть) нумеруются и подписываются. Рисунок и подпись к рисунку выравниваются по центру. Подпись к рисунку располагается под ним, к таблице – над.

Контрольные вопросы

1. Что такое инициация проекта? Какие документы создаются на этом этапе?

2. Назовите четыре основных правила мозгового штурма.

3. Почему на этапе генерации идей запрещена критика?

4. Какие критерии помогут выбрать лучшую идею проекта из списка?

5. Перечислите не менее трёх ролей в учебной IT-команде и их основные функции.

6. Что такое Yougile? Для каких задач он используется?

7. Какие колонки (статусы) являются минимально необходимыми на канбан-доске?

8. Зачем нужен бэклог (Backlog) в проекте? Чем он отличается от колонки «В работе»?

9. Кто в команде отвечает за своевременное обновление статусов задач в Yougile?

10. Что значит MVP (Minimum Viable Product) и как его определение связано с инициацией проекта?

Практическая работа №3

Основы анализа бизнес-процессов

Цель работы: научиться выделять и описывать бизнес-процессы в предметной области IT-проекта, освоить базовые принципы функционального моделирования с использованием нотации IDEF0, построить контекстную диаграмму для выбранного проекта.

Задачи:

1. Изучить основные понятия бизнес-процесса и функциональной модели.
2. Освоить нотацию IDEF0: блок, интерфейсные дуги (вход, выход, управление, механизм).
3. Научиться формулировать цель и точку зрения модели.
4. Построить контекстную диаграмму (A-0) для своего проекта, определив его границы и основные взаимодействия с внешней средой.
5. Подготовить описание процесса верхнего уровня в текстовом виде.

Теоретическая часть

1. Что такое бизнес-процесс

Бизнес-процесс – это совокупность взаимосвязанных действий или работ, направленных на создание определённого продукта или услуги для потребителя. Процесс имеет:

- **входы** (ресурсы, информация, заявки),
- **выходы** (результат, продукт),
- **управление** (правила, стандарты, законы),
- **механизмы** (исполнители, оборудование, ПО).

Пример для IT-проекта: процесс «Разработка модуля авторизации» – вход: требования, выход: работающий код, управление: гайдлайны безопасности, механизмы: разработчики, IDE.

2. Нотация IDEF0: основные элементы

IDEF0 – методология функционального моделирования, стандарт для описания бизнес-процессов. Модель представляет собой иерархию диаграмм. Базовый элемент – **функциональный блок** (Activity).

Функциональный блок

Изображается прямоугольником, внутри которого – название процесса (глагол + существительное, например: «Разработать мобильное приложение»).

Интерфейсные дуги (стрелки)

Всего четыре типа стрелок, каждая подключается к своей стороне блока:

Сторона блока	Тип стрелки	Что обозначает	Пример для IT-проекта
Левая	Вход (Input)	Что преобразуется процессом	Заявка пользователя, сырые данные
Правая	Выход (Output)	Результат процесса	Готовое приложение, отчёт
Верхняя	Управление (Control)	Правила, ограничения, стандарты	Техническое задание, ГОСТ, бюджет
Нижняя	Механизм (Mechanism)	Исполнители, инструменты, ресурсы	Команда разработки, IDE, Yougile

Важно: стрелки должны быть подписаны существительными.

3. Контекстная диаграмма (A-0)

Контекстная диаграмма – это самая верхняя диаграмма в иерархии IDEF0. Она представляет модель в целом как один блок. Её цель – определить границы модели и её взаимодействие с внешним миром.

Правила построения A-0:

– Только один функциональный блок.

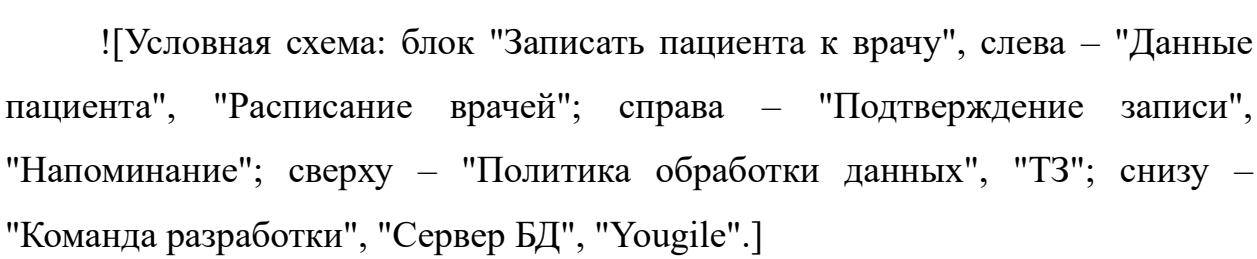
– Показываются все внешние сущности, которые влияют на процесс.

– Входы и выходы отражают, что именно входит в систему и что из неё выходит.

– Управление – это документы, стандарты, требования.

– Механизмы – люди, инструменты, оборудование.

Пример для учебного проекта «Приложение для записи к врачу»:



4. Цель и точка зрения модели

Перед построением необходимо определить:

- **Цель моделирования** – зачем мы строим диаграмму? (например, «понять основные функции системы», «согласовать границы проекта с заказчиком»).
- **Точка зрения** – с чьей позиции рассматривается процесс? (например, «менеджер проекта», «аналитик», «пользователь»). Это определяет, какие детали важны, а какие опускаются.

5. Правила именования

- Блок: «Отглагольное существительное + дополнение» (не просто «Пользователь», а «Авторизовать пользователя»).
- Стрелки: существительные, отражающие объект или данные («Требования», «Отчёт о тестировании», «Команда»).

Практическая часть

Выполняется в командах (сформированных в ПР2) с использованием проектной идеи, выбранной ранее.

Задание 1. Определение цели и точки зрения

Для своего проекта сформулируйте в текстовом виде:

- **Цель построения IDEF0-модели** (например: «Определить основные функции системы и её взаимодействие с внешними сущностями для последующей декомпозиции»).
- **Точка зрения** (например: «Аналитик проекта», «Руководитель разработки»).

Запишите в общий документ команды.

Задание 2. Выделение внешних сущностей и потоков

Обсудите в команде и составьте список того, что находится **вне** вашего проекта, но взаимодействует с ним. Заполните таблицу:

Тип	Сущность / поток	Пример для проекта «Приложение для учёта задач»
Входы (что получает система извне)	Регистрационные данные пользователя, новая задача	Логин, пароль, название задачи
Выходы (что система отдаёт наружу)	Уведомление, список задач, отчёт о выполнении	Push-уведомление, JSON с задачами
Управление (правила, ограничения)	Техническое задание, политика конфиденциальности, требования заказчика	ТЗ, закон о персональных данных
Механизмы (кто/что выполняет работу)	Команда разработки, сервер, Yougile, компилятор	3 разработчика, хостинг, Git

Задание 3. Построение контекстной диаграммы (25 минут)

Используя таблицу из задания 2, нарисуйте контекстную диаграмму (один блок A-0). Можно рисовать:

- **На бумаге** (сфотографировать для отчёта);
- **В онлайн-инструментах** (Draw.io, Lucidchart, Diagrams.net);
- **В текстовом редакторе с использованием фигур** (Word, Google Drawings).

Требования к диаграмме:

- Блок с названием процесса, соответствующим вашему проекту (например: «Разработать мобильное приложение для записи к врачу» или более конкретно «Обработать запись пациента» – выберите уровень, который описывает всю систему).
- Не менее 2 входов, 2 выходов, 1 управления, 2 механизмов.

– Все стрелки подписаны.

– Соблюдены стороны подключения стрелок (лево – вход, право – выход, верх – управление, низ – механизмы).

Совет: не перегружайте диаграмму. Если много стрелок – объедините их в группы (например, «Данные пользователя» вместо «Имя, телефон, email»).

Задание 4. Описание процесса

К диаграмме подготовьте текстовое описание по шаблону:

Название процесса: [название блока]

Цель модели: [из задания 1]

Точка зрения: [из задания 1]

Входы: перечислить и кратко пояснить.

Выходы: перечислить.

Управление: какие документы/правила регулируют процесс.

Механизмы: кто и с помощью чего выполняет процесс.

Краткое описание: 2-3 предложения о том, как процесс преобразует входы в выходы.

Задание 5. Презентация и обсуждение

Каждая команда показывает свою контекстную диаграмму и поясняет:

– почему выбраны именно такие входы/выходы,

– что является главным управляющим документом,

– достаточны ли механизмы для выполнения процесса.

Остальные команды могут задавать вопросы и указывать на пропущенные важные потоки.

Требования к отчёту

Отчёт сдаётся **каждой командой** в электронном виде (PDF, DOCX или ссылка на облачный документ). Индивидуальные отчёты не требуются, но преподаватель может провести опрос для проверки вклада каждого.

Структура отчёта:

1. Титульный лист
2. Цель работы (переписать из методички).
3. Задачи (переписать из методички).
4. Исходные данные – краткое описание выбранного проекта (из ПР2).
5. Цель моделирования и точка зрения (из задания 1).
6. Таблица потоков (задание 2) – в виде оформленной таблицы.
7. Контекстная диаграмма (A-0) – изображение (скриншот или фото) с подписью «Рисунок 1 – Контекстная диаграмма процесса [название]».
8. Текстовое описание процесса (задание 4).
9. Вывод – что удалось понять о проекте благодаря диаграмме, какие сложности возникли при выделении потоков, насколько полезен метод IDEF0 для начала работы над проектом.

Технические требования

1. Формат файла: .pdf (предпочтительно) или .docx.
2. Шрифт: Times New Roman, 14 пт для основного текста, 12 пт для таблиц и подписей.
3. Межстрочный интервал: 1,5.
4. Поля: левое – 30 мм, правое – 15 мм, верх/низ – 20 мм.
5. Абзацный отступ (отступ первой строки) – 1,25 см.
6. Заголовки разделов – полужирные, выравнивание по ширине, абзацный отступ, нумерация (при необходимости).
7. Таблицы и рисунки (если есть) нумеруются и подписываются. Рисунок и подпись к рисунку выравниваются по центру. Подпись к рисунку располагается под ним, к таблице – над.

8. Диаграмма должна быть чёткой, текст на стрелках читаемым. Если использован ручной рисунок – качественное фото с хорошим освещением.

Контрольные вопросы

1. Что такое бизнес-процесс? Приведите пример из IT-сферы.
2. Назовите четыре типа стрелок в нотации IDEF0 и укажите, к какой стороне блока они подключаются.
3. Чем вход отличается от механизма? Приведите примеры для одного процесса.
4. Что такое контекстная диаграмма (A-0) и для чего она нужна?
5. Почему перед построением диаграммы нужно определить цель и точку зрения?
6. Может ли на контекстной диаграмме быть больше одного блока? Почему?
7. Как правильно назвать функциональный блок в IDEF0? Приведите правильное и неправильное название.
8. В какой нотации можно построить контекстную диаграмму, кроме IDEF0? (например, DFD, BPMN – для общего развития).
9. Какую роль играет техническое задание в модели IDEF0 – вход, управление или механизм? Почему?
10. Если на диаграмме не указан механизм, что это означает для процесса?

Практическая работа №4

Сбор требований к ПО

Цель работы: освоить основные методы выявления требований к ПО (анкетирование, интервью, анализ документов), научиться собирать и структурировать информацию о предметной области, составлять глоссарий терминов для IT-проекта.

Задачи:

1. Изучить классификацию требований к ПО и методы их сбора.
2. Освоить технику подготовки вопросов для интервью и анкетирования заказчиков/пользователей.
3. Научиться анализировать существующие документы (аналоги, регламенты) для выявления требований.
4. Составить глоссарий ключевых терминов предметной области своего проекта.
5. Подготовить отчёт по результатам сбора требований.

Теоретическая часть

1. Что такое требования к ПО

Требование – это условие или возможность, необходимая пользователю для решения проблемы или достижения цели. Требования бывают:

Тип	Описание	Пример
Функциональные	Что система должна делать	«Пользователь должен иметь возможность войти по логину и паролю»
Нефункциональные	Как система это делает (качество)	«Время отклика страницы – не более 2 секунд», «Система должна работать под нагрузкой 1000 пользователей»

Ограничения	Технологические, бюджетные, временные рамки	«Разработка ведётся на Python», «Бюджет – 500 тыс. руб.»

2. Методы выявления требований

Анкетирование

– **Что это:** письменный опрос с использованием заранее составленных вопросов (открытых/закрытых).

– **Когда применять:** большая аудитория респондентов, нужно собрать статистику.

– **Плюсы:** охват, экономия времени на опрос.

– **Минусы:** нет возможности уточнить ответ, низкая глубина.

Совет: смешивайте закрытые вопросы (с вариантами ответов) и открытые (свободный комментарий).

Интервью

– **Что это: беседа** с заказчиком, пользователем или экспертом по заранее подготовленному сценарию.

– **Когда применять:** нужно глубже понять потребности, выявить неочевидные требования.

– **Плюсы:** возможность задавать уточняющие вопросы, гибкость.

– **Минусы:** трудоёмко, требует навыков ведения беседы.

Структура интервью:

1. Вступление (представление, цель).
2. Общие вопросы о деятельности респондента.
3. Конкретные вопросы о задачах и проблемах.
4. Вопросы о желаемых функциях системы.
5. Завершение (благодарность, предложение доп. контакта).

Анализ документов

– **Что это:** изучение существующих регламентов, инструкций, описаний процессов, конкурентных решений.

– **Когда применять:** есть бумажные или электронные источники информации.

– **Плюсы:** объективность, возможность выявить скрытые требования.

– **Минусы:** документы могут устареть или не отражать реальную практику.

Какие документы анализировать:

– Техническое задание (если есть на аналогичный продукт).

– Пользовательские руководства конкурентов.

– Нормативные акты (например, законы о персональных данных).

– Протоколы совещаний, письма заказчика.

3. Глоссарий предметной области

Глоссарий – это структурированный словарь терминов, используемых в проекте. Он необходим для:

– единого понимания терминов всеми участниками (заказчик, аналитик, разработчики);

– устранения неоднозначности («клиент», «пользователь», «администратор» – разные сущности);

– документирования предметной области.

Формат записи термина:

Термин	Определение	Синонимы	Источник
...	...	...	...

Пример для проекта «Интернет-магазин»:

Термин	Определение	Синонимы	Источник
Корзина	Хранилище выбранных пользователем товаров до оформления заказа	«Корзина покупок»	Интервью с заказчиком
Сессия	Период активности пользователя после входа в систему	–	Анализ аналогов

4. Процесс сбора требований (общая схема)

1. Планирование – какие методы используем, кого опрашиваем, какие документы изучаем.

2. Проведение – анкетирование, интервью, анализ.

3. Документирование – запись всех выявленных требований (можно в виде списка или пользовательских историй).

4. Валидация – проверка требований с заказчиком (не противоречат ли они, все ли важные учтены).

Практическая часть

Выполняется в командах (сформированных в ПР2)

Задание 1. Определение источников требований

Для своего проекта заполните таблицу:

Категория источников	Конкретные источники (люди, документы)
Заказчики / пользователи	(например: студенты, преподаватели, администратор)
Эксперты предметной области	(например: методист, врач, бухгалтер)
Документы	(например: аналоги, инструкции, законы)
Конкуренты / аналоги	(названия 1–2 похожих приложений)

Задание 2. Подготовка инструментов сбора требований

Вариант А (интервью): Составьте сценарий интервью с представителем целевой аудитории. Включите:

- 3–4 общих вопроса о текущей проблеме,
- 5–7 конкретных вопросов о желаемых функциях,
- 2 вопроса об ожидаемых ограничениях (безопасность, скорость, платформа).

Вариант Б (анкета): Разработайте анкету из 8–10 вопросов (минимум 5 закрытых, 3 открытых). Примеры закрытых: «Как часто вы сталкиваетесь с

проблемой?» (варианты: ежедневно, раз в неделю, реже). Открытый: «Какие дополнительные функции вы хотели бы видеть?»

Вариант В (анализ документов): Найдите в интернете описание или скриншоты приложения-аналога. Составьте список из 5–7 функциональных требований, которые можно позаимствовать, и 3–4 недостатков (то, что можно улучшить).

Задание 3. Формулировка требований

На основе проведённого опроса или анализа документов составьте **список требований** в виде таблицы:

ID	Тип	Требование	Приоритет (высокий/средний/низкий)	Источник
F1	Функциональное	Система должна отправлять уведомление на email после регистрации	Высокий	Интервью, вопрос 3
NF1	Нефункциональное	Время загрузки главной страницы не более 3 секунд	Средний	Анализ аналогов

Минимум 5 требований (3 функциональных, 2 нефункциональных).

Задание 4. Составление глоссария

Составьте глоссарий из 8–10 терминов, относящихся к вашему проекту. Используйте форму:

Термин	Определение (своими словами, без жаргона)	Синонимы (если есть)
...	...	...

Рекомендация: включите термины из предметной области (например, для проекта «Запись к врачу»: пациент, врач, специализация, талон, льготная категория и т.д.) и системные термины (сессия, профиль, уведомление).

Требования к отчёту

Отчёт сдаётся каждой командой в электронном виде (PDF, DOCX или ссылка на Google Docs).

Структура отчёта:

1. Титульный лист
2. Цель работы (переписать из методички).
3. Задачи (переписать из методички).
4. Краткое описание проекта (1–2 абзаца).
5. Таблица источников требований (задание 1).
6. Инструмент сбора требований (сценарий интервью / анкета / список проанализированных документов – в зависимости от выбранного варианта).
7. Результаты опроса (заметки по интервью или заполненные анкеты – можно в виде фото/скриншотов или текстовой расшифровки).
8. Таблица требований (задание 4) – минимум 5 строк.
9. Глоссарий (задание 5) – минимум 8 терминов.
10. Вывод – что удалось выявить, какие методы оказались наиболее полезными, какие трудности возникли при формулировке требований.

Технические требования

1. Формат файла: .pdf (предпочтительно) или .docx.
2. Шрифт: Times New Roman, 14 пт для основного текста, 12 пт для таблиц и подписей.
3. Межстрочный интервал: 1,5.
4. Поля: левое – 30 мм, правое – 15 мм, верх/низ – 20 мм.
5. Абзацный отступ (отступ первой строки) – 1,25 см.

6. Заголовки разделов – полужирные, выравнивание по ширине, абзацный отступ, нумерация (при необходимости).

7. Таблицы и рисунки (если есть) нумеруются и подписываются. Рисунок и подпись к рисунку выравниваются по центру. Подпись к рисунку располагается под ним, к таблице – над.

8. Диаграмма должна быть чёткой, текст на стрелках читаемым. Если использован ручной рисунок – качественное фото с хорошим освещением.

Контрольные вопросы

1. В чём разница между функциональными и нефункциональными требованиями? Приведите по одному примеру для своего проекта.

2. Какие три метода выявления требований были рассмотрены? В каких ситуациях каждый из них предпочтителен?

3. Почему интервью даёт более глубокую информацию, чем анкетирование? В чём его недостаток?

4. Что такое глоссарий проекта и зачем он нужен, если есть техническое задание?

5. Как определить приоритет требования: высокий, средний, низкий? Какие критерии использовать?

6. Кто должен участвовать в сборе требований – только заказчик или также конечные пользователи? Почему?

7. Как анализ документов конкурентов помогает в сборе требований?

8. Может ли требование противоречить другому требованию? Приведите пример и как поступить в такой ситуации.

9. Что делать, если заказчик не может чётко сформулировать требования?

10. Почему важно указывать источник каждого требования в таблице?

Практическая работа №5

Техническое задание (ТЗ). Структура

Цель работы: изучить структуру технического задания (ТЗ) на разработку программного обеспечения в соответствии с ГОСТ 34.602-89, а также познакомиться с современными аналогами (Smart Spec, шаблонами IT-компаний).

Задачи:

1. Изучить назначение и структуру ТЗ как основного документа, регламентирующего разработку ПО.
2. Ознакомиться с требованиями ГОСТ 34.602-89 к содержанию разделов ТЗ.
3. Сравнить классический подход (ГОСТ) с современными практиками (Smart Spec, шаблонами Agile-команд).
4. Научиться формулировать раздел «Основание для разработки», фиксируя исходные документы и причины начала проекта.
5. Научиться формулировать раздел «Назначение разработки», определяя цели и область применения будущего продукта.
6. Составить фрагмент ТЗ для своего проекта (первые три раздела).

Теоретическая часть

1. Понятие технического задания (ТЗ)

Техническое задание (ТЗ) – это основополагающий документ, определяющий требования и условия создания программного продукта (или автоматизированной системы). ТЗ фиксирует взаимные обязательства заказчика и исполнителя и служит правовой и технической основой для всех последующих этапов разработки.

Хорошо составленное ТЗ должно отвечать на три главных вопроса:

– **Что** должна делать система (функциональные требования)?

– **Как** она должна это делать (ограничения, технологии)?

– **По каким критериям** будет оцениваться результат (приёмка)?

Важно: согласно исследованиям, около **71% неудачных программных проектов** терпят крах именно из-за отсутствия чётких требований. ТЗ – это не бюрократия, а инструмент снижения рисков.

2. Два стандарта: ГОСТ 34.602-89 и современные подходы

ГОСТ 34.602-89

Действующий межгосударственный стандарт, регламентирующий состав, содержание и правила оформления ТЗ на создание автоматизированных систем. Применяется преимущественно в госсекторе, оборонной промышленности и на крупных предприятиях. В соответствии со стандартом, ТЗ включает следующие разделы:

1. **Общие сведения** – наименование системы, заказчик, разработчик, сроки, источники финансирования.
2. **Назначение и цели создания системы** – вид деятельности, цели и критерии эффективности.
3. **Характеристика объекта автоматизации** – информация об организации/процессах, которые автоматизируются.
4. **Требования к системе** – требования к функциям, видам обеспечения (информационное, программное, техническое).
5. **Состав и содержание работ по созданию системы** – перечень стадий и этапов.
6. **Порядок контроля и приемки системы** – критерии, процедуры тестирования и сдачи.
7. **Требования к составу и содержанию работ по подготовке объекта к вводу системы в действие** – подготовка персонала, установка.
8. **Требования к документированию** – перечень и формы документов.
9. **Источники разработки** – материалы и документы, использованные при составлении ТЗ.

Smart Spec и Agile-подходы

В коммерческой разработке, стартапах и Agile-среде используют облегчённые версии ТЗ (например, **Smart Spec**, **User Story Mapping**, **Specification by Example**). Основные черты современных подходов:

– **Ориентация на пользовательские истории:** «Как пользователь, я хочу…, чтобы…».

– **Критерии приёмки (Acceptance Criteria):** короткий список условий, при которых задача считается выполненной.

– **Живая документация:** требования хранятся в системах трекинга (Jira, Yougile) и обновляются итеративно.

– **Минимализм:** только то, что действительно нужно команде для начала работы.

Сравнительная таблица:

Характеристика	ГОСТ 34.602-89	Smart Spec / Agile
Полнота	Максимально полная, детальная	Лаконичная, только ключевое
Способ хранения	Отдельный документ (.docx, .pdf)	Wiki, задачи в трекере, карточки
Изменения	Требуют формального согласования	Вносятся в любой момент (до начала итерации)
Применение	Госзаказ, критические системы	Стартапы, веб-сервисы, мобильные приложения

3. Раздел «Основание для разработки»

Этот раздел фиксирует **юридические и организационные причины** начала разработки. Он отвечает на вопрос: «На каком основании мы делаем этот проект?».

Что указывается:

– Документ (или документы), на основании которого ведётся разработка (приказ, распоряжение, контракт, учебное задание).

– Организация, утвердившая документ, и дата утверждения.

– Плановые сроки начала и окончания работ.

Типовой шаблон для учебного проекта:

Основанием для выполнения работы является учебный план направления 09.03.04 «Программная инженерия» и задание на практическую работу, выданное преподавателем кафедры [название] «_» _____ **20** г.

Пример из реальной практики:

Основанием для разработки информационной системы «Автотранспортное предприятие» является техническое задание, утверждённое генеральным директором ООО «ТрансЛогистик» 15.01.2024 г.

4. Раздел «Назначение разработки»

Раздел «Назначение разработки» является **основополагающим** – от того, насколько чётко в нём сформулированы цели и задачи, зависит весь ход дальнейшей разработки. Он отвечает на вопросы: «Зачем нужна система?» и «Какую проблему она решает?».

Структура раздела по ГОСТ 34.602-89:

2.1. Назначение системы (вид деятельности и перечень объектов автоматизации)

Например: *Система предназначена для автоматизации процесса записи пациентов в частной медицинской клинике. Объектами автоматизации являются: регистратура, врачи трёх отделений, администратор клиники.*

2.2. Цели создания системы (конкретные, измеримые результаты)

Например: *Сократить время записи одного пациента с 5 минут до 1 минуты. Снизить количество неявок пациентов на 30% за счёт автоматических напоминаний.*

2.3. Критерии эффективности (как измерить достижение цели)

Например: Система считается эффективной, если после её внедрения среднее время записи не превышает 90 секунд, а процент неявок снизился не менее чем на 25%.

Пример формулировки для проекта «Мобильное приложение для записи к врачу»:

2.1. Назначение системы: Система предназначена для автоматизации записи пациентов к врачам частной клиники. Объектами автоматизации являются: регистратура, врачи-специалисты (терапевты, педиатры, хирурги), администратор клиники, пациенты клиники (через мобильное приложение).

2.2. Цели создания системы: 1) Сократить время записи одного пациента в регистратуре с 5 до 1 минуты. 2) Предоставить пациентам возможность самостоятельной записи через мобильное приложение 24/7. 3) Снизить количество неявок пациентов на 30% за счёт push-напоминаний.

2.3. Критерии эффективности: Система признаётся эффективной, если после её внедрения: среднее время записи составляет не более 90 секунд; доля записей через мобильное приложение достигает 40% от общего числа; процент неявок снизился не менее чем на 25%.

5. Жизненный цикл ТЗ

ТЗ не пишется один раз и навсегда. Типовой жизненный цикл включает:

1. Разработка – сбор требований, написание черновика.

2. Оформление – приведение к утверждённому формату.

3. Согласование – обсуждение с заказчиком, уточнение спорных моментов.

4. Утверждение – подписание сторонами, после чего ТЗ становится обязательным к исполнению.

5. Изменение – при необходимости вносятся корректировки (по согласованию).

Практическая часть

Выполняется в командах (сформированных в ПР2) с использованием проектной идеи, выбранной ранее.

Задание 1. Анализ структуры ТЗ

Индивидуальная работа: Каждый студент получает фрагмент ТЗ (реальный или учебный). Заполните таблицу анализа:

Раздел ТЗ	Какая информация содержится?	Присутствует ли в вашем фрагменте? (да/нет)
Общие сведения		
Назначение и цели		
Требования к системе		
Состав и содержание работ		
Порядок контроля и приемки		

Задание 2. Сравнение подходов

Команда обсуждает и заполняет таблицу сравнения для своего проекта:

Критерий	Подход ГОСТ 34.602-89	Современный подход (Smart Spec)
Какой объём документа?		
Кто будет читать этот документ?		
Как часто можно менять требования?		
Подходит ли этот подход для вашего проекта? (почему?)		

Вывод команды: кратко (2–3 предложения), какой подход вы выберете для своего учебного проекта и почему.

Задание 3. Составление раздела «Основание для разработки»

Сформулируйте раздел «Основание для разработки» для своего проекта. Используйте шаблон:

2. ОСНОВАНИЕ ДЛЯ РАЗРАБОТКИ

Основанием для разработки программного продукта «[название проекта]» является:

– учебный план направления 09.03.04 «Программная инженерия», утверждённый ректором «[название вуза]» «_» _____ 20 г.;

– задание на выполнение практических работ по дисциплине «Введение в проектную деятельность», выданное преподавателем кафедры [название] «_» _____ 20 г.

– Плановые сроки начала и окончания разработки: [дата] – [дата].

Дополнение (по желанию): Представьте, что ваш проект – реальный коммерческий. Придумайте реалистичное основание (приказ гендиректора компании, государственный контракт и т.п.).

Задание 4. Составление раздела «Назначение разработки»

Сформулируйте раздел «Назначение разработки» для своего проекта по структуре, аналогичной пункту 3.4 данной методички. Заполните таблицу:

Подраздел	Ваша формулировка
2.1. Назначение системы (вид деятельности, объекты автоматизации)	
2.2. Цели создания системы (минимум 2 измеримые цели)	
2.3. Критерии эффективности (минимум 2 измеримых критерия)	

Рекомендация: Формулируйте цели и критерии так, чтобы их можно было проверить цифрами (например, «скорость загрузки < 2 секунд», а не «хорошая скорость»).

Задание 5. Фиксация в Yougile (5 минут)

Вернитесь в свою доску Yougile (созданную в ПР2). Добавьте новую колонку **«ТЗ»** (перед колонкой «Бэклог»). Создайте в ней три карточки задач:

– **Карточка 1:** «Согласовать основание для разработки с преподавателем» (исполнитель: капитан, дедлайн: к следующему занятию).

– **Карточка 2:** «Доработать раздел "Назначение разработки" на основе рецензии» (исполнитель: аналитик, дедлайн: +2 дня).

– **Карточка 3:** «Перенести цели и критерии в README проекта» (исполнитель: разработчик, дедлайн: к концу спринта).

Требования к отчёту

Отчёт сдаётся **каждой командой** в электронном виде (PDF, DOCX или ссылка на Google Docs).

Структура отчёта:

1. Титульный лист
2. Цель работы (переписать из методички).
3. Задачи (переписать из методички).
4. Краткое описание проекта (1–2 абзаца из ПР2).
5. Задание 1 – заполненная таблица анализа фрагмента ТЗ.
6. Задание 2 – таблица сравнения подходов и вывод команды.
7. Задание 3 – раздел «Основание для разработки» (итоговый вариант).
8. Задание 4 – раздел «Назначение разработки» (итоговый вариант).
9. Вывод – что нового вы узнали о структуре ТЗ, какие разделы оказались самыми сложными для составления, какой подход выбрали и почему.

Технические требования

1. Формат файла: .pdf (предпочтительно) или .docx.
2. Шрифт: Times New Roman, 14 пт для основного текста, 12 пт для таблиц и подписей.
3. Межстрочный интервал: 1,5.
4. Поля: левое – 30 мм, правое – 15 мм, верх/низ – 20 мм.
5. Абзацный отступ (отступ первой строки) – 1,25 см.
6. Заголовки разделов – полужирные, выравнивание по ширине, абзацный отступ, нумерация (при необходимости).
7. Таблицы и рисунки (если есть) нумеруются и подписываются. Рисунок и подпись к рисунку выравниваются по центру. Подпись к рисунку располагается под ним, к таблице – над.

8. Диаграмма должна быть чёткой, текст на стрелках читаемым. Если использован ручной рисунок – качественное фото с хорошим освещением.

Контрольные вопросы

1. Для чего нужно техническое задание? Какие три главных вопроса оно должно прояснять?

2. Перечислите основные разделы ТЗ по ГОСТ 34.602-89. Какой из них является основополагающим?

3. Что указывается в разделе «Основание для разработки»? Почему этот раздел важен с юридической точки зрения?

4. Чем отличается формулировка цели в ТЗ от пожелания «сделать хорошо»? Приведите пример измеримой цели.

5. В чём разница между «назначением системы» и «целями создания системы» в ТЗ?

6. Какие критерии эффективности вы указали в своём ТЗ? Как их можно проверить?

7. Что такое Smart Spec? В каких случаях его применение предпочтительнее ГОСТ 34.602-89?

8. Почему в Agile-командах требования хранят в Jira/Yougile, а не в отдельном Word-файле?

9. Какой из двух подходов (ГОСТ или Smart Spec) вы выбрали для своего учебного проекта и почему?

10. Может ли ТЗ изменяться после утверждения? Если да, то как должен быть организован процесс изменений?

Практическая работа №6

ТЗ: Требования к функциональным характеристикам

Цель работы: научиться формулировать функциональные требования к программному продукту, описывать функции системы в структурированном виде (варианты использования, пользовательские истории) и применять метод MoSCoW для приоритезации требований в рамках технического задания.

Задачи:

1. Изучить понятие «функциональное требование» и его место в структуре ТЗ.
2. Освоить два способа описания функций: варианты использования (use cases) и пользовательские истории (user stories).
3. Научиться применять метод приоритезации MoSCoW (Must have, Should have, Could have, Won't have).
4. Составить список функциональных требований для своего проекта и распределить их по категориям MoSCoW.
5. Зафиксировать требования в виде карточек в Yougile, создав первую итерацию (MVP).

Теоретическая часть

1. Функциональные требования в ТЗ

В соответствии с ГОСТ 34.602-89, раздел «Требования к функциональным характеристикам» (или «Функции системы») описывает, **какие действия** должна выполнять система и **в каком режиме**. Это один из ключевых разделов ТЗ, так как именно на его основе разработчики будут писать код, а тестировщики – проверять результат.

Примеры функциональных требований:

– Система должна обеспечивать регистрацию пользователя по email и паролю.

– Система должна отправлять push-уведомление при изменении статуса заказа.

– Система должна формировать отчёт о посещаемости за выбранный период в формате Excel.

Важно: Функциональное требование отвечает на вопрос **«Что система делает?»**, а не **«Как она это делает?»** (это уже проектирование).

2. Способы описания функций

Варианты использования (Use Cases)

Метод, предложенный Иваром Якобсоном. Каждый вариант использования описывает взаимодействие **актора** (пользователя или внешней системы) с системой для достижения конкретной цели.

Структура краткого описания use case:

– **Название:** Действие + цель (например, «Зарегистрировать пользователя»).

– **Актор:** Кто инициирует (например, «Незарегистрированный пользователь»).

– **Предусловие:** Что уже должно быть верно (например, «Пользователь открыл форму регистрации»).

– **Основной сценарий:** Пошаговый список действий (1, 2, 3…).

– **Постусловие:** Состояние системы после успешного выполнения (например, «Пользователь зарегистрирован и вошёл в систему»).

– **Альтернативные сценарии:** Что при ошибках (например, «Email уже существует – показать сообщение об ошибке»).

Пользовательские истории (User Stories)

Подход из Agile (Scrum). История пишется от лица пользователя и фокусируется на **ценности**.

Формат: «Как [роль пользователя], я хочу [действие], чтобы [получить ценность]».

Пример: «Как зарегистрированный пользователь, я хочу сбросить пароль по email, чтобы восстановить доступ к аккаунту».

Критерии приёмки (Acceptance Criteria): краткий список условий, при которых история считается выполненной. Например:

– Пользователь вводит email, на который зарегистрирован аккаунт.

– Система отправляет письмо со ссылкой для сброса.

– После перехода по ссылке пользователь может задать новый пароль.

Сравнение:

Характеристика	Use Case	User Story
Детализация	Высокая (пошаговая)	Низкая (одно предложение + критерии)
Кому удобно	Аналитикам, тестировщикам	Заказчику, разработчику
Когда применять	Крупные, критичные функции	Agile, быстрая разработка
Формат	Таблица / структурированный текст	Короткий шаблон

Для учебного проекта допустимо использовать **оба подхода** в зависимости от сложности функции.

3. Приоритезация требований: метод MoSCoW

Метод MoSCoW (также пишется Moscow) – это техника расстановки приоритетов, широко применяемая в управлении проектами и разработке ПО. Название – аббревиатура:

Категория	Расшифровка	Значение	Доля в проекте (примерно)
M	Must have	Обязательно должно быть в продукте. Без этих функций проект не имеет смысла.	60%
S	Should have	Очень важно, но при	20%

		необходимости можно отложить на следующий релиз.	
C	Could have	Хорошо бы иметь, но не критично. Реализуется при наличии времени/ресурсов.	15%
W	Won't have	В данном релизе делать не будем (но, возможно, в будущем).	5%

Важно: «Won't have» – это сознательный отказ, а не забытая функция. Он помогает управлять объёмом и фокусироваться на главном.

Пример для проекта «Мобильное приложение для записи к врачу»:

Категория	Расшифровка	Значение	Доля в проекте (примерно)
Пользователь может посмотреть свободное время врачей	Must have	Без этого нет смысла в приложении	Пользователь может посмотреть свободное время врачей
Пользователь может записаться на приём	Must have	Ключевая функция	Пользователь может записаться на приём
Приложение отправляет напоминание за 2 часа до приёма	Should have	Важно, но можно реализовать в первой итерации попозже	Приложение отправляет напоминание за 2 часа до приёма
Интеграция с календарём телефона (Google Calendar)	Could have	Удобно, но не обязательно для MVP	Интеграция с календарём телефона (Google Calendar)
Видеозвонок с врачом через приложение	Won't have (в данном релизе)	Сложно, выходит за рамки текущего бюджета	Видеозвонок с врачом через приложение

4. Связь MoSCoW с MVP (Minimum Viable Product)

MVP – это минимальный жизнеспособный продукт, который содержит **только Must have** функции. MVP выпускается, чтобы получить обратную связь от реальных пользователей с минимальными затратами.

Правило: **всё, что не Must have, отправляется в следующие итерации.**

Для учебного проекта ваша задача – выделить **MVP** (обычно 3–5 ключевых функций) и описать их в формате user stories или use cases.

5. Типичные ошибки при формулировании требований

1. **Смешивание функциональных и нефункциональных требований.**
   Неправильно: «Система должна быстро обрабатывать запросы» (это нефункциональное, производительность).
   Правильно: «Система должна возвращать список свободных врачей за 2 секунды» – функциональное с количественным показателем (но лучше всё же вынести в нефункциональные).
2. **Слишком общие формулировки.**
   Неправильно: «Система должна быть удобной».
   Правильно: «Система должна предоставлять поиск врачей по специальности и району».
3. **Отсутствие проверяемости.**
   Неправильно: «Система должна хорошо работать».
   Правильно: «Система должна выдерживать 100 одновременных пользователей без падения времени отклика ниже 3 секунд».
4. **Смешивание реализации.**
   Неправильно: «Система должна использовать базу данных PostgreSQL».
   Правильно: «Система должна хранить данные о пользователях, врачах и записях». Выбор СУБД – это проектное решение.

Практическая часть

Выполняется в командах (из ПР2) на основе проекта, для которого уже составлены «Основание» и «Назначение разработки» (из ПР5).

Задание 1. Мозговой штурм функций (10 минут)

Каждая команда на стикерах или в общем документе генерирует **все возможные функции** своего продукта. Не ограничивайте себя – пишите всё, что приходит в голову. Цель – собрать 15–20 идей.

Инструмент: доска в Yougile (колонка «Бэклог идей»)

Задание 2. Приоритезация MoSCoW (15 минут)

Каждую функцию отнесите к одной из четырёх категорий MoSCoW. Заполните таблицу:

№	Функция (требование)	Категория (M/S/C/W)	Обоснование (почему именно так)
1			
2			
...			

Правила:

- Категорий **Must have** не должно быть больше 60% от общего числа (например, из 15 функций – максимум 9 Must have).
- Хотя бы 1–2 функции должны попасть в **Won't have** (чтобы вы научились сознательно отказываться).
- Обсуждение в команде – обязательно. Если спорите, используйте критерий: «Без этой функции продукт всё ещё решает основную проблему?» Если да – это не Must have.

Задание 3. Описание Must have функций

Выберите **3–5 функций с категорией Must have** (ваш MVP). Опишите каждую в формате **User Story + Acceptance Criteria** (или Use Case – по выбору команды).

Шаблон для User Story:

US-01. [Название]

Как [роль пользователя], я хочу [действие], чтобы [ценность].

Критерии приёмки:

– Условие 1

– Условие 2

– Условие 3

Пример (для проекта «Приложение для записи к врачу»):

US-01. Просмотр свободного времени врача

Как пациент, я хочу видеть доступные временные слоты выбранного врача, чтобы выбрать удобное время для визита.

Критерии приёмки:

– Пользователь выбирает врача из списка.

– Система отображает календарь с занятыми и свободными слотами (интервал 15 минут).

– Свободные слоты выделены зелёным, занятые – серым.

– При отсутствии свободных слотов на выбранный день показывается сообщение «Нет свободного времени».

Для сложных функций можно использовать Use Case с пошаговым сценарием.

Задание 4. Оформление раздела «Функции системы» для ТЗ

На основе выполненных заданий 1–3 составьте **официальный раздел технического задания** с заголовком **«Требования к функциональным характеристикам»** (или «Функции системы»). Используйте следующий шаблон:

3. ТРЕБОВАНИЯ К ФУНКЦИОНАЛЬНЫМ ХАРАКТЕРИСТИКАМ

3.1. Состав функций

Система должна выполнять следующие функции:

№	Наименование функции	Краткое описание	Приоритет (MoSCoW)
Ф1	[Название]	[1-2 предложения]	Must have
Ф2	...	...	Should have
...	...	...	...

3.2. Детальное описание приоритетных функций (Must have)

Ф1. [Название функции]

- **Актор(ы):** [кто использует]
- **Предусловие:** [что уже верно]
- **Основной сценарий:**
    1. [шаг 1]
    2. [шаг 2]
    3. ...
- **Постусловие:** [состояние системы после]
- **Альтернативные сценарии:** [если есть]

Ф2. [Название функции] – аналогично.

(Для функций Should have и Could have можно дать только краткое описание без детального сценария.)

Важно: Этот раздел впоследствии станет частью вашего сводного ТЗ (вместе с разделами из ПР5). Сохраните его в отдельном файле.

Задание 5. Документирование в Yougile

В доске Yougile создайте колонки по приоритетам: «Must have (MVP)», «Should have», «Could have», «Won't have». Перенесите карточки функций из заданий 1–3. Для Must have заполните описание (user story + критерии приёмки), назначьте ответственного.

Требования к отчёту

Отчёт сдаётся каждой командой в электронном виде (PDF, DOCX или ссылка на Google Docs).

Структура отчёта:

1. Титульный лист
2. Цель работы (переписать из методички).
3. Задачи (переписать из методички).
4. Краткое описание проекта (1–2 абзаца из ПР2).
5. Результаты мозгового штурма – список всех сгенерированных функций (задание 1).
6. Таблица приоритезации MoSCoW (задание 2) – минимум 10–15 строк.
7. Описание Must have функций (задание 3) – минимум 3 функции в формате User Story + Acceptance Criteria (или Use Case).
8. Оформленный раздел «Требования к функциональным характеристикам» для ТЗ (задание 4)
9. Скриншот доски Yougile с колонками «Must have (MVP)», «Should have», «Could have», «Won't have» – карточки должны быть видны.
10. Вывод – какие функции вошли в MVP, почему вы отказались от некоторых функций (Won't have), как метод MoSCoW помог в расстановке приоритетов.

Технические требования

1. Формат файла: .pdf (предпочтительно) или .docx.
2. Шрифт: Times New Roman, 14 пт для основного текста, 12 пт для таблиц и подписей.
3. Межстрочный интервал: 1,5.
4. Поля: левое – 30 мм, правое – 15 мм, верх/низ – 20 мм.
5. Абзацный отступ (отступ первой строки) – 1,25 см.
6. Заголовки разделов – полужирные, выравнивание по ширине, абзацный отступ, нумерация (при необходимости).

7. Таблицы и рисунки (если есть) нумеруются и подписываются. Рисунок и подпись к рисунку выравниваются по центру. Подпись к рисунку располагается под ним, к таблице – над.

8. Диаграмма должна быть чёткой, текст на стрелках читаемым. Если использован ручной рисунок – качественное фото с хорошим освещением.

Контрольные вопросы

1. Что такое функциональное требование? Приведите пример из вашего проекта.

2. В чём разница между вариантом использования (use case) и пользовательской историей (user story)?

3. Перечислите четыре категории метода MoSCoW. Что означает каждая?

4. Почему метод MoSCoW называется именно так (откуда буквы)?

5. Как определить, что требование – Must have, а не Should have? Какой вопрос нужно задать?

6. Что такое MVP? Как он связан с категорией Must have?

7. Почему в проекте важно выделять требования Won't have (сознательно отказываться)?

8. Может ли требование перейти из категории Could have в Must have в процессе разработки? Приведите пример.

9. Какую долю от всех требований обычно занимают Must have (приблизительно)?

10. Кто должен принимать решение о приоритете требований – команда разработки или заказчик? Почему?

Практическая работа №7

ТЗ: Требования к составу и параметрам технических средств

Цель работы: научиться выявлять и формулировать нефункциональные требования к программному продукту, в частности требования к техническому обеспечению (состав и параметры технических средств), а также к надежности, эргономике и программной совместимости.

Задачи:

1. Изучить классификацию нефункциональных требований и их место в структуре ТЗ.
2. Освоить способы формулирования требований к техническим средствам (железу, серверам, клиентским устройствам).
3. Научиться описывать требования к надежности (отказоустойчивость, время восстановления, резервирование).
4. Изучить требования к эргономике (удобство интерфейса, соответствие стандартам).
5. Освоить требования к программной совместимости (ОС, браузеры, СУБД, API).
6. Составить раздел ТЗ «Требования к составу и параметрам технических средств» для своего проекта.

Теоретическая часть

1. Нефункциональные требования: общее понятие

Нефункциональные требования (или требования к качеству) описывают, *как* система должна работать, в отличие от функциональных требований, которые описывают *что* система делает. Они определяют ограничения и атрибуты качества.

Согласно стандарту ISO/IEC 25010, основные категории нефункциональных требований включают:

– **Надёжность** (отказоустойчивость, восстанавливаемость)

– **Производительность** (быстродействие, пропускная способность)

– **Безопасность** (целостность, конфиденциальность)

– **Совместимость** (с другим ПО, оборудованием)

– **Удобство использования (эргономика)**

– **Сопровождаемость** (модульность, лёгкость изменений)

– **Переносимость** (адаптация к разным платформам)

В данной работе мы сосредоточимся на трёх группах, наиболее часто встречающихся в ТЗ для небольших IT-проектов: требования к техническим средствам, надежность, эргономика, программная совместимость.

2. Требования к составу и параметрам технических средств

Этот раздел (в ГОСТ 34.602-89 — п. 4.4 «Требования к техническому обеспечению») определяет, какое оборудование необходимо для функционирования системы. Он включает:

– **Требования к серверному оборудованию** (процессор, ОЗУ, дисковая подсистема, сетевое оборудование).

– **Требования к клиентским устройствам** (ПК, ноутбуки, планшеты, смартфоны: версия ОС, разрешение экрана, объём памяти).

– **Требования к периферии** (принтеры, сканеры, терминалы сбора данных и т.п.).

Пример формулировок:

Серверная часть системы должна функционировать под управлением ОС Linux (Ubuntu 20.04 или новее) на виртуальной машине с параметрами: не менее 4 vCPU, 8 ГБ RAM, 100 ГБ SSD.

Клиентское мобильное приложение должно работать на устройствах под управлением Android 11+ и iOS 15+. Минимальное разрешение экрана – 1080×1920 пикселей.

Важно: Не указывайте конкретные бренды (например, «Intel Core i5») без крайней необходимости. Лучше описывать минимальные характеристики («процессор с тактовой частотой не менее 2.0 ГГц, 4 ядра»).

3. Требования к надёжности

Надёжность — это способность системы сохранять работоспособность в заданных условиях. В ТЗ обычно указывают:

- **Вероятность безотказной работы** (например, 99,9% времени в месяц).
- **Время наработки на отказ** (MTBF).
- **Максимальное время восстановления** (MTTR) после сбоя.
- **Требования к резервированию** (горячее резервирование, репликация БД).
- **Устойчивость к некорректным действиям пользователя** (защита от дурака).

Примеры:

Система должна сохранять работоспособность при одновременной работе не менее 100 пользователей. Коэффициент готовности системы – не менее 0,98.

При отказе основного сервера автоматическое переключение на резервный должно происходить не более чем за 5 минут.

Система должна автоматически сохранять копии базы данных каждые 24 часа и хранить их не менее 30 дней.

4. Требования к эргономике

Эргономика (или удобство использования, usability) определяет, насколько продукт понятен, удобен и доступен целевой аудитории. В ТЗ могут быть указаны:

- **Стиль интерфейса** (единый для всех экранов, соответствие гайдлайнам платформы — Material Design, Human Interface Guidelines).

- **Требования к навигации** (например, «от главного экрана до целевого действия не более трёх кликов»).
- **Поддержка специальных возможностей** (экранный диктор, высокая контрастность, увеличение шрифта).
- **Язык интерфейса** (русский, английский).
- **Время реакции на действия пользователя** (здесь на стыке с производительностью).

Примеры:

Все экраны системы должны быть выполнены в единой цветовой гамме (фирменные цвета компании). Кнопки вызова основного действия должны быть зелёными, кнопки отмены — серыми.

Интерфейс веб-версии должен адаптироваться под ширину экрана от 320 до 1920 пикселей (резиновая вёрстка).

Система должна отображать всплывающие подсказки для всех полей ввода при наведении курсора.

5. Требования к программной совместимости

Этот раздел определяет, с каким другим программным обеспечением и внешними системами должен взаимодействовать ваш продукт. Включает:

- **Совместимость с операционными системами** (Windows, macOS, Linux, Android, iOS — конкретные версии).
- **Поддерживаемые браузеры** для веб-приложений (Chrome, Edge, Firefox, Safari — последние 2 версии).
- **Совместимость с СУБД** (например, PostgreSQL 14+, MySQL 8+).
- **Возможность интеграции с внешними API** (платёжные системы, сервисы уведомлений, календари).
- **Форматы импорта/экспорта данных** (CSV, JSON, XML, Excel).

Примеры:

Система должна корректно работать в браузерах Google Chrome (версия 120+), Mozilla Firefox (115+), Яндекс.Браузер (23+).

Должна быть обеспечена интеграция с REST API сервиса отправки SMS-уведомлений через протокол HTTPS.

Выгружаемые отчёты должны сохраняться в форматах .xlsx и .pdf.

6. Место в структуре ТЗ

По ГОСТ 34.602-89 или в современном шаблоне ТЗ разделы распределяются следующим образом:

Раздел ТЗ (пример)	Что входит
1. Общие сведения	Наименование, заказчик, разработчик
2. Назначение разработки	Цели, область применения
3. Требования к системе	3.1. Функциональные (ПР6), 3.2. Техническое обеспечение (эта работа), 3.3. Надёжность, 3.4. Эргономика, 3.5. Программная совместимость и др.
4. Состав работ	...
5. Порядок контроля	...

Таким образом, данная работа дополняет ваше ТЗ новым разделом.

Практическая часть

Задание 1. Определение требований к техническим средствам

Для своего проекта заполните таблицу. Если проект – веб-приложение, опишите требования к серверу и к клиентским браузерам. Если мобильное приложение – требования к смартфонам. Если десктопное – к ПК.

Компонент	Параметры / Требования	Обоснование (почему именно так)
Серверное оборудование	(CPU, RAM, диск, ОС)	(например, «для обработки 50 запросов/сек»)
Клиентские устройства (тип, ОС)		
Минимальные характеристики клиента	(процессор, память, разрешение)	

Сетевое оборудование	(пропускная способность, протоколы)	

Задание 2. Требования к надёжности

Сформулируйте не менее 4 требований к надёжности для вашего проекта. Используйте шаблоны:

- Система должна обеспечивать коэффициент готовности не менее ...%.
- Время восстановления после сбоя не должно превышать ... минут.
- Резервное копирование данных должно выполняться с периодичностью ...
- Система должна автоматически фиксировать критические ошибки в логах.
- При потере сетевого соединения клиентское приложение должно сохранять введённые данные локально.

Задание 3. Требования к эргономике

Сформулируйте 3–5 требований к удобству интерфейса. Учтите специфику вашей целевой аудитории (студенты, врачи, менеджеры и т.д.). Примеры:

- Все кнопки должны иметь подписи и иконки.
- Ошибки валидации должны отображаться красным цветом рядом с полем ввода.
- Приложение должно поддерживать тёмную тему оформления.
- Количество кликов для выполнения основного сценария (например, записи к врачу) не должно превышать 5.

Задание 4. Требования к программной совместимости

Перечислите, с какими внешними системами и программными платформами должен взаимодействовать ваш продукт. Заполните таблицу:

Категория	Конкретные требования (версии, протоколы, форматы)
Операционные системы (сервер)	
Операционные системы (клиент)	
Браузеры (если веб-приложение)	
СУБД	
Внешние API / сервисы	
Форматы обмена данными	

Задание 5. Оформление раздела ТЗ «Требования к составу и параметрам технических средств»

На основе заданий 1–4 составьте структурированный раздел ТЗ. Используйте следующий шаблон:

3. ТРЕБОВАНИЯ К СИСТЕМЕ

3.2. Требования к техническому обеспечению (состав и параметры технических средств)

3.2.1. Требования к серверному оборудованию: [текст]

3.2.2. Требования к клиентским устройствам: [текст]

3.3. Требования к надёжности [список требований]

3.4. Требования к эргономике [список требований]

3.5. Требования к программной совместимости [список требований]

Добавьте этот раздел в общий документ ТЗ вашей команды (который начали в ПР5 и ПР6).

Задание 6. Фиксация в Yougile (5 минут)

В доске Yougile создайте колонку **«Нефункциональные требования»** (или добавьте тег «нефункциональное»). Создайте карточки для каждого сформулированного требования (по одному на карточку). Для приоритетных (например, связанных с отказоустойчивостью) укажите исполнителя и дедлайн.

Требования к отчёту

Отчёт сдаётся каждой командой в электронном виде (PDF, DOCX или ссылка на Google Docs).

Структура отчёта:

11. Титульный лист
12. Цель работы (переписать из методички).
13. Задачи (переписать из методички).
14. Краткое описание проекта (1–2 абзаца).
15. Задание 1 – таблица требований к техническим средствам.
16. Задание 2 – список требований к надёжности (не менее 4).
17. Задание 3 – список требований к эргономике (не менее 3).
18. Задание 4 – таблица требований к программной совместимости.
19. Задание 5 – оформленный раздел ТЗ (полный текст).
20. Задание 6 – скриншот доски Yougile с карточками нефункциональных требований.
21. Вывод – что нового узнали о нефункциональных требованиях, какие из них оказались сложнее всего сформулировать.

Технические требования

1. Формат файла: .pdf (предпочтительно) или .docx.
2. Шрифт: Times New Roman, 14 пт для основного текста, 12 пт для таблиц и подписей.
3. Межстрочный интервал: 1,5.
4. Поля: левое – 30 мм, правое – 15 мм, верх/низ – 20 мм.
5. Абзацный отступ (отступ первой строки) – 1,25 см.
6. Заголовки разделов – полужирные, выравнивание по ширине, абзацный отступ, нумерация (при необходимости).
7. Таблицы и рисунки (если есть) нумеруются и подписываются. Рисунок и подпись к рисунку выравниваются по центру. Подпись к рисунку располагается под ним, к таблице – над.

8. Диаграмма должна быть чёткой, текст на стрелках читаемым. Если использован ручной рисунок – качественное фото с хорошим освещением.

Контрольные вопросы

1. Что такое нефункциональные требования? Чем они отличаются от функциональных?
2. Перечислите не менее четырёх категорий нефункциональных требований.
3. Что включает раздел «Требования к составу и параметрам технических средств»?
4. Почему не следует указывать конкретные бренды (например, «Intel Core i7») в требованиях к техническим средствам?
5. Какие показатели надежности обычно указывают в ТЗ?
6. Что такое коэффициент готовности (availability) системы?
7. Приведите пример требования к эргономике для мобильного приложения.
8. Зачем нужны требования к программной совместимости? Что будет, если их не определить?
9. Может ли требование к надёжности противоречить требованию к эргономике? Приведите гипотетический пример.
10. В каком разделе ТЗ (по ГОСТ или по вашему шаблону) находятся нефункциональные требования?

Практическая работа №8

Введение в UML и Markdown

Цель работы: освоить базовые инструменты документирования проектов: язык разметки Markdown для оформления текстовой документации и ознакомиться с основами языка UML для визуального моделирования.

Задачи:

1. Изучить синтаксис Markdown: заголовки, списки, ссылки, таблицы, код, изображения.
2. Познакомиться с основными диаграммами UML (краткий обзор) и их назначением.
3. Научиться создавать структурированные документы в Markdown с использованием любого редактора (VS Code, Obsidian, онлайн-редакторы).
4. Оформить итоговый отчёт по практической работе в формате .md (Markdown).
5. Экспортировать Markdown-документ в PDF или HTML для сдачи.

Теоретическая часть

1. Язык UML (Unified Modeling Language)

UML – это стандартизированный язык графического моделирования для описания, визуализации, проектирования и документирования программных систем. Он не зависит от языка программирования и позволяет разным участникам проекта (аналитикам, разработчикам, заказчикам) понимать архитектуру системы.

Основные типы диаграмм UML (кратко):

Диаграмма	Назначение	Пример использования
Диаграмма вариантов использования (Use Case)	Показывает, кто (акторы) и что (прецеденты) делает в системе	На этапе сбора требований
Диаграмма классов (Class)	Описывает структуру данных и отношения между классами	Для проектирования базы данных и ООП
Диаграмма последовательности (Sequence)	Отображает обмен сообщениями между объектами во времени	Для детализации сценариев
Диаграмма деятельности (Activity)	Похожа на блок-схему, показывает бизнес-процессы	Для моделирования алгоритмов

2. Язык Markdown: назначение и области применения

Markdown – это лёгкий язык разметки, созданный для форматирования текста в plain text (обычный текст) с последующим преобразованием в HTML, PDF и другие форматы. Он используется для:

- оформления README.md на GitHub/GitLab;
- написания технической документации (например, в MkDocs, Docusaurus);
- ведения заметок (Obsidian, Notion, Joplin);
- создания постов на форумах (Stack Overflow, Reddit);
- подготовки отчётов (как в вашем учебном проекте).

Преимущества Markdown перед Word:

- Лёгкость и читаемость даже в сыром виде.
- Версионируется в Git (различия между версиями видны построчно).
- Кроссплатформенность.
- Возможность автоматической генерации PDF/HTML.

3. Основные элементы синтаксиса

1. Абзац

Абзац – это одна или несколько подряд идущих строчек текста, отделённых одной или несколькими пустыми строчками. Если строка содержит только пробелы или табы, то она всё равно считается пустой.

Подряд идущие строчки будут склеены в одну, если не добавить жёсткий перенос. Существует несколько способов, как это можно сделать:

– добавить два (или больше) пробелов в конце строки **<пробел><пробел>**, чтобы сделать мягкий перенос;

– добавить обратную косую черту в конце строки \;

– добавить HTML-тег переноса строки **
.**

Пример работы с абзацами в Markdown:

```
1  Привет,
2  мир!
3
4  Привет,<пробел><пробел>
5  пробел!
6
7  Привет,\
8  косая черта!
9
10 Привет,<br>
11 тег бр-р-р!
12
```

```
Привет,
мир!

Привет,<пробел><пробел>
пробел!

Привет,
косая черта!

Привет,


тег бр-р-р!
```

Рисунок 1 – Разметка абзацев в Markdown и её результат

2. Заголовки

Назначение: создание иерархической структуры документа, выделение разделов и подразделов.

Синтаксис: символ # в начале строки, затем обязательно пробел, после которого пишется текст заголовка. Количество символов # определяет уровень заголовка:

– заголовок первого уровня (самый крупный, обычно один на документ);

– заголовок второго уровня;

…

– заголовок шестого уровня.

Правила: заголовок должен располагаться на отдельной строке. После заголовка не должно быть других символов # в этой же строке (они будут считаться частью текста). Пустые строки до и после заголовка не обязательны, но улучшают читаемость исходного кода.

Пример:

```
# Заголовок 1 уровня
## Заголовок 2 уровня
### Заголовок 3 уровня
#### Заголовок 4 уровня
##### Заголовок 5 уровня
###### Заголовок 6 уровня
```

Рисунок 2 – Разметка заголовков в Markdown

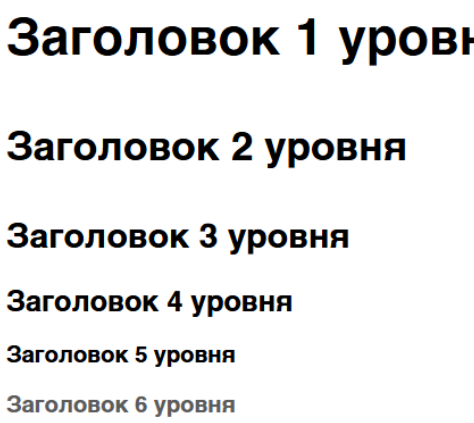


Рисунок 3 – Отображение заголовков

Ещё одним способом написания заголовков в Markdown является использование подчёркиваний (===).

«Подчёркивание» параграфа знаками равно (=) или дефисами (-) делает его заголовком первого или второго уровня соответственно. Уровень заголовка зависит только от типа «чёрточек», их количество значения не имеет.

Между текстом и «подчёркиванием» не должно быть пустых строк.

```
Заголовок 1 уровня
==================

Заголовок 2 уровня
------------------

Заголовок, который подчеркнули одним символом
-

Заголовок второго
уровня из нескольких
строчек текста
------------------
```

Рисунок 4 – Использование подчёркиваний для написания заголовков

Заголовок 1 уровня

Заголовок 2 уровня

Заголовок, который подчеркнули одним символом

Заголовок второго уровня из нескольких строчек текста

Рисунок 5 – Результат использования подчёркиваний для написания заголовков

3. Выделение текста (жирный, курсив)

Назначение: акцентирование важных слов, терминов, названий.

3.1. Жирный текст

Синтаксис: **текст** или __текст__ (два символа * или _ с обеих сторон).

Правила: символы выделения должны прилегать к выделяемому тексту без пробелов внутри (пробелы допустимы только вокруг всего выражения).

Пример:

Рисунок 6 – Выделение текста жирным

3.2. Курсив

Синтаксис: *текст* или _текст_ (один символ * или _).

Пример:

Рисунок 7 – Выделение текста курсивом

3.3. Комбинирование

Можно использовать оба вида выделения одновременно, например ***очень важно*** → ***очень важно*** (жирный курсив).

Рисунок 8 – Комбинированное выделение

4. Списки

Назначение: представление перечней элементов, инструкций, характеристик.

4.1. Маркированный список

Синтаксис: каждая строка начинается с *, + или -, затем пробел и текст элемента.

Правила: для вложенных элементов добавляется отступ в 2 или 4 пробела перед маркером.

Пример:

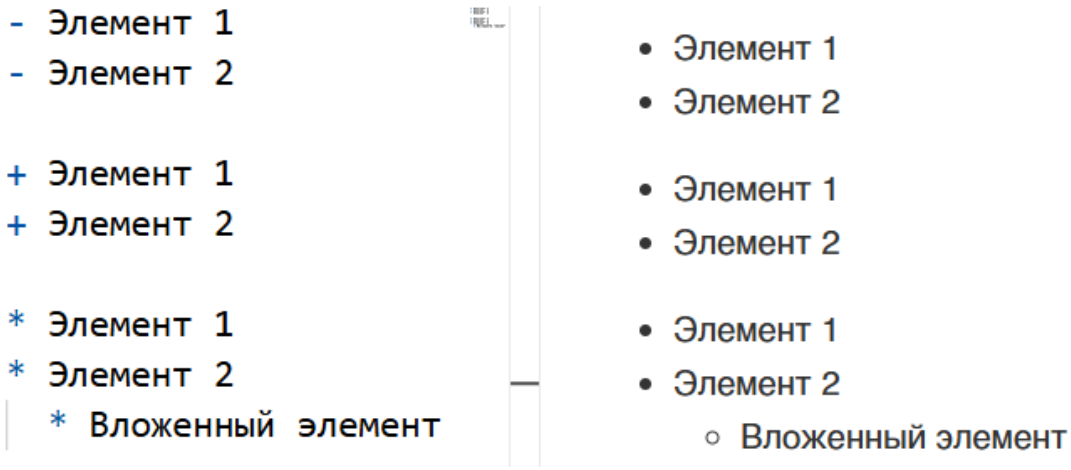


Рисунок 9 – Разметка списков в Markdown

4.2. Нумерованный список

Синтаксис: строка начинается с числа, точки и пробела (например, 1., 2. и т.д.).

Правила: нумерация может быть любой – Markdown автоматически выстраивает правильный порядок при рендеринге, но рекомендуется использовать последовательные числа для удобства чтения исходника. Для вложенных элементов добавляется отступ в 2 или 4 пробела перед номером пункта.

Пример:

Рисунок 10 – Оформление нумерованного списка

Номера пунктов в итоговой разметке будут идти по порядку (1, 2, 3), даже если в Markdown будут стоять 1., 4., 9.. Такая особенность позволяет нам использовать «ленивую нумерацию», когда перед каждым пунктом ставится одно и то же число.

Пример:

Рисунок 11 – Использование «ленивой нумерации»

Число перед первым пунктом показывает с чего нужно начинать нумеровать элементы списка, поэтому если в Markdown поставить 99., 1., 2., то в итоговой разметки пункты будут стоять под номерами 99, 100, 101.

Пример:

Рисунок 12 – Автоматический подбор нумерации списка

4.3. Вложенные списки

Можно смешивать типы: внутри нумерованного списка создавать маркированный подсписок и наоборот. Для этого достаточно изменить маркер и добавить отступ или таб.

Пример:

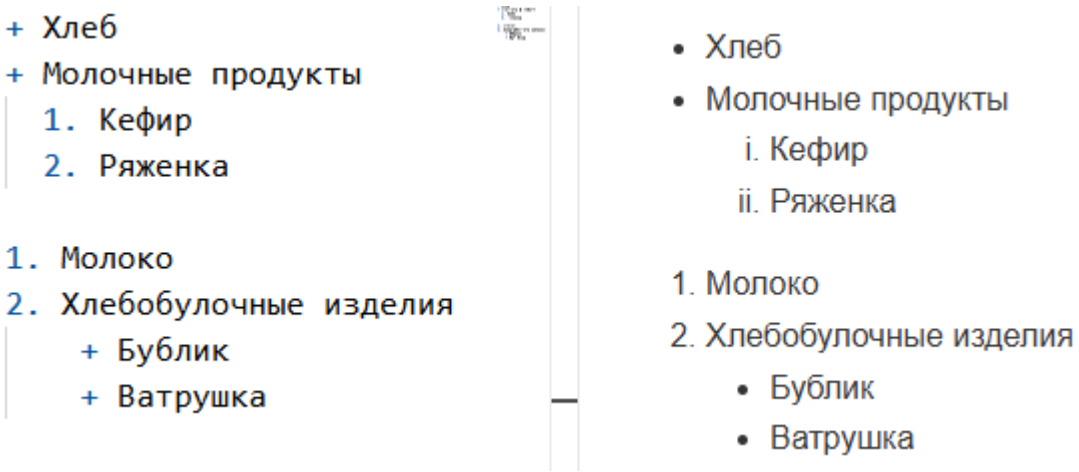


Рисунок 13 – Вложенные списки

Количество пробелов, которое нужно использовать, чтобы вложить один список в другой, может варьироваться. Оно зависит от количества символов в родительском маркере (L):

+ – 1 символ (L = 1)

1. – 2 символа (L = 2)

99. – 3 символа (L = 3)

Перед вложенным списком нужно поставить от L + 1 до L + 4 пробелов.

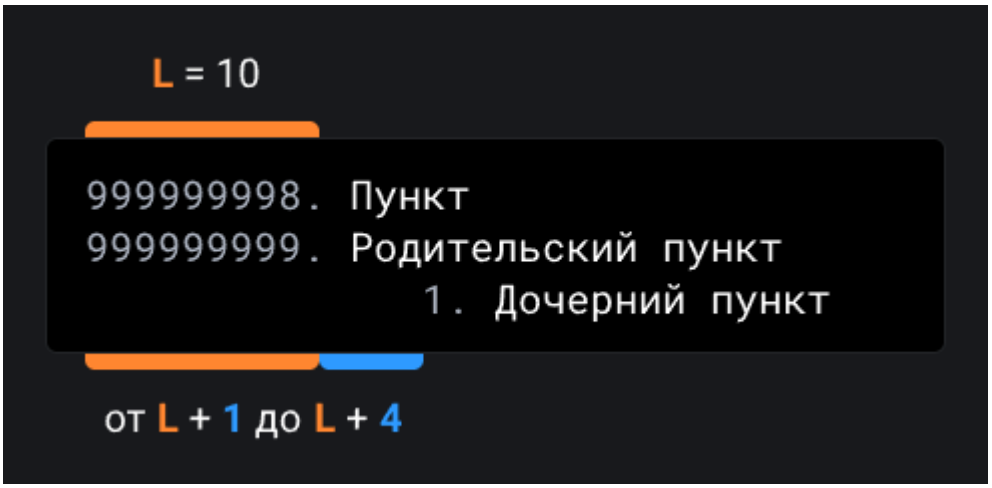


Рисунок 14 – Расчёт количества пробелов для вложенности

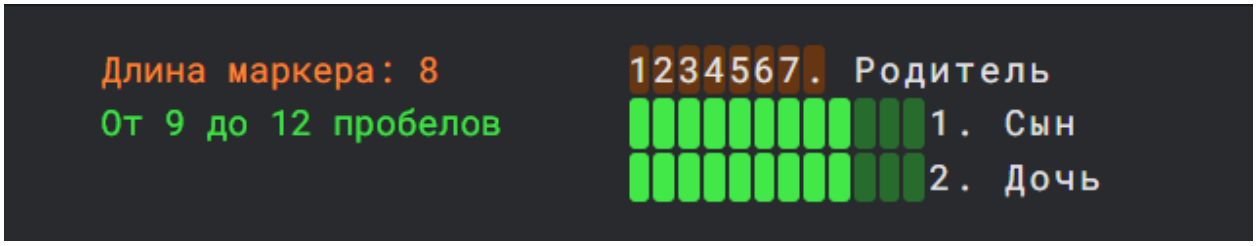


Рисунок 15 – Пример расчёта количества пробелов для вложенности

5. Ссылки

Назначение: создание гиперссылок на внешние ресурсы, локальные файлы или разделы документа.

Markdown предлагает 3 стиля разметки ссылок: **строчный**, **справочный** и **автоматический**.

Строчные ссылки:

Внешняя ссылка: [текст ссылки](URL).

Ссылка с подсказкой: [текст](URL "подсказка") – при наведении отображается всплывающий текст.

Относительная ссылка: можно указывать путь к файлу относительно текущего документа.

Внутренняя ссылка (якорь): ссылка на заголовок в том же документе – [текст](#название-заголовка). Пробелы в названии заменяются дефисами, буквы приводятся к нижнему регистру. Некоторые реализации требуют, чтобы заголовок был единственным в документе (или использовались дополнительные атрибуты).

Пример:

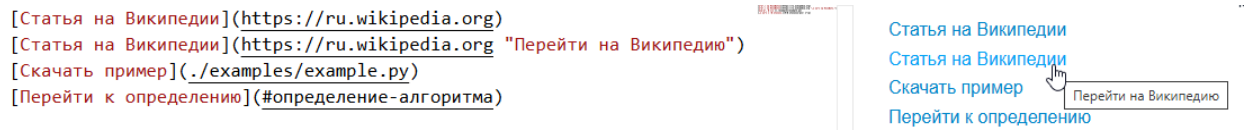


Рисунок 16 – Использование строчной ссылки

Справочные ссылки:

Для вставки ссылки в справочном стиле нужно написать [Текст ссылки][Ключ] в том месте, где вы хотите её поместить, а где-нибудь выше или ниже добавить сноску [Ключ]: URL "Подсказка".

Пример:

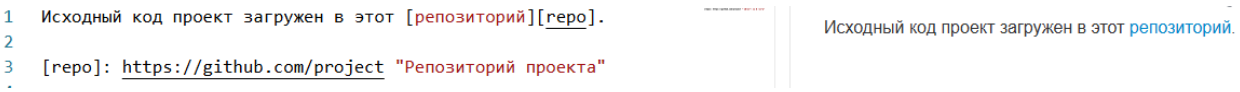


Рисунок 17 – Использование справочной ссылки

Автоматические ссылки:

Markdown позволяет использовать упрощённый вариант для вставки ссылок, для этого нужно просто обернуть URL треугольными скобками (<URI>).

Можно вставлять адреса электронной почты (<example@exmail.com>), тогда мы автоматически получим ссылку типа mailto:.

Пример:

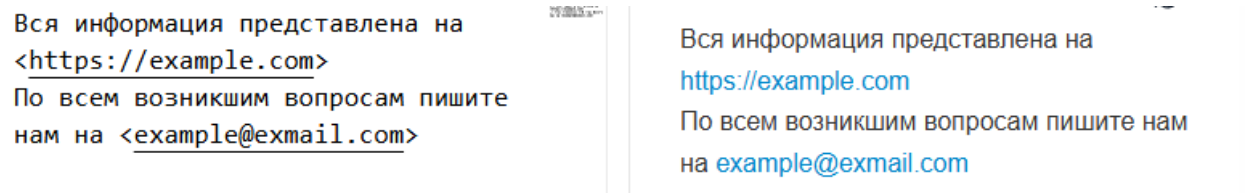


Рисунок 18 – Использование автоматической ссылки

6. Изображения

Назначение: вставка графических иллюстраций, блок-схем, скриншотов.

Конструкции для вставки изображений очень похожи на те, что используются для ссылок. Предлагается 2 стиля разметки: **строчный** и **справочный**.

Строчные изображения:

Синтаксис: ![альтернативный текст](путь_к_изображению)

– Альтернативный текст отображается, если изображение не загрузилось, и важен для доступности (скринридеры);

– Путь может быть URL или относительным/абсолютным путём к файлу.

Можно добавить **подсказку:**

![альтернативный текст](путь_к_изображению "подсказка")

Правила: для локальных файлов рекомендуется использовать относительные пути, чтобы документ оставался переносимым.

Пример:

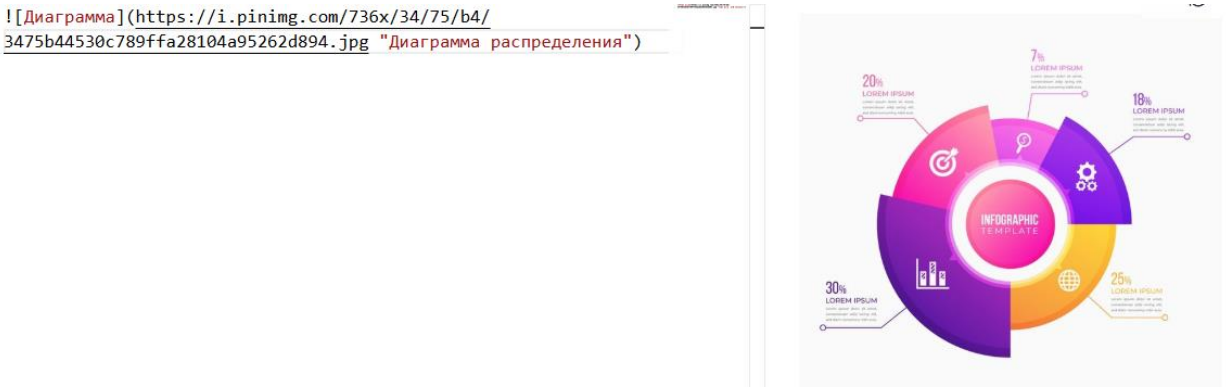


Рисунок 19 – Вставка изображения по URL строчным методом

```
![Локальный рисунок](./images/logo.jpg)
```

Рисунок 20 – Вставка локального изображения строчным методом

Справочные изображения:

Для вставки изображения в справочном стиле нужно написать

![альтернативный текст][Ключ]

в том месте, где вы хотите его поместить, а где-нибудь выше или ниже описать картинку по ключу [Ключ]: URL картинки "Подсказка".

Пример:

```
![Диаграмма][1]

[1]: https://i.pinimg.com/736x/34/75/b4/
3475b44530c789ffa28104a95262d894.jpg "Диаграмма распределения"
```

Рисунок 21 – Вставка изображения справочным методом

7. Блоки кода

Назначение: вставка многострочных фрагментов исходного кода с сохранением форматирования и возможной подсветкой синтаксиса.

Синтаксис:

```язык
код
```

– язык – необязательное указание языка программирования для подсветки (python, cpp, javascript и т.д.);

– Блок кода может содержать любые символы, включая обратные кавычки (если их больше, чем в открывающей последовательности).

Markdown позволяет специальным образом размечать исходный код, все символы внутри будут автоматически экранированы. Есть 3 способа, как это можно сделать:

1. Инлайн-код, т.е. обернуть код одной-двумя парами бэктиков (`код`):

Этот способ позволяет вставлять исходный код как строчный элемент. Даже если фактически у нас будет несколько строчек кода, обёрнутых бэктиками, мы всё равно получим одну строку после конвертации в HTML.

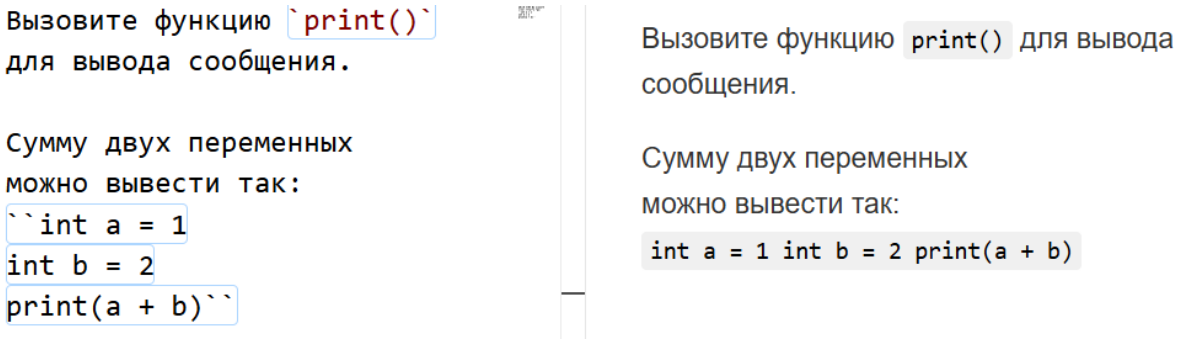


Рисунок 22 – Инлайн-код

2. Обернуть код тремя и более парами бэктиков (```код```).

Если обернуть одну строчку кода тремя или более парами бэктиков, то мы получим строчный элемент, а если несколько строчек, то – блочный. Второй вариант позволяет указывать язык программирования, который мы используем, для этого нужно написать его сразу после открывающих бэктиков.

После обозначения языка программирования визуально ничего не изменится, но это даст возможность дополнительным плагинам и скриптам подсветить код внутри блока.

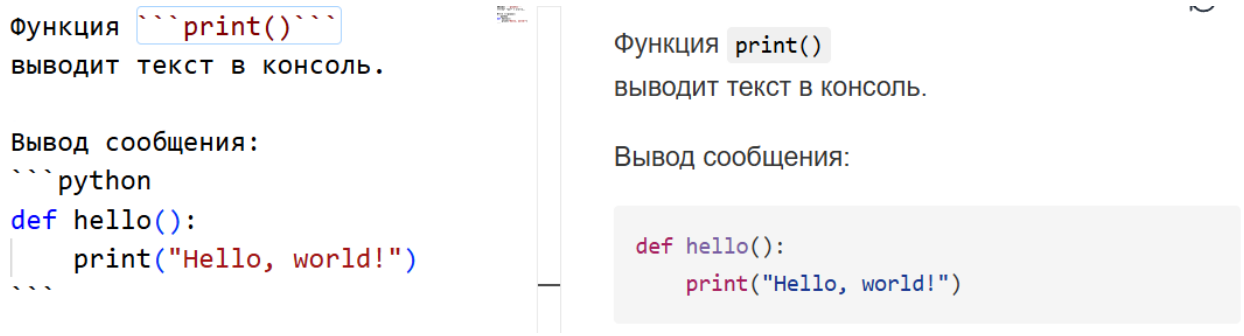


Рисунок 23 – Код, обёрнутый тремя и более парами бэктиков

3. Поставить таб или 4 пробела перед каждой строчкой кода.

В данном случае исходный код необходимо отделять пустой строкой от предыдущего блока.

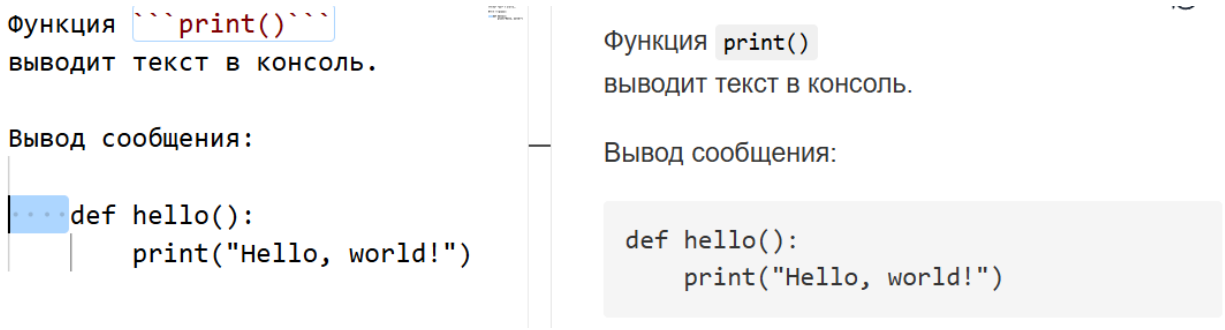


Рисунок 24 – Использование таба или 4 пробелов для выделения блока кода

8. Горизонтальный разделитель

Назначение: визуальное разделение смысловых блоков документа.

Синтаксис: три и более символа -, * или _ на отдельной строке. Между ними могут быть пробелы, но не другие символы.

Пример:

Рисунок 25 – Символы для разделителя

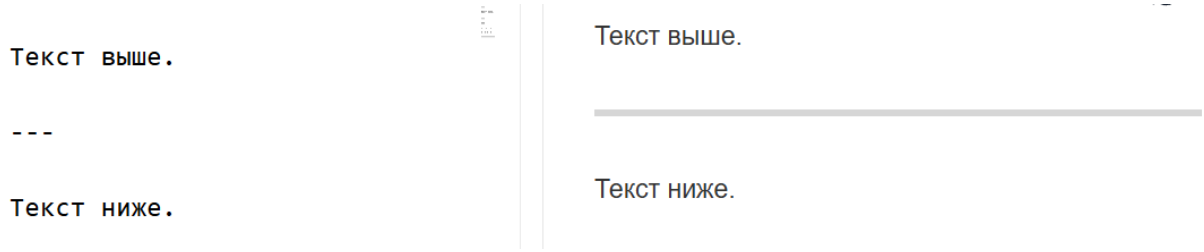


Рисунок 26 – Разделитель в тексте

В дополнение к базовым элементам (заголовки, списки, ссылки, изображения, блоки кода, горизонтальные разделители) Markdown поддерживает расширенный синтаксис, который реализован в большинстве современных платформ (GitHub, GitLab, VS Code с плагинами, StackEdit, Typora и др.).

9. Таблицы

Назначение: представление структурированных данных в виде строк и столбцов.

Синтаксис:

– Строки таблицы отделяются друг от друга переводами строк;

– Ячейки каждой строки разделяются вертикальными чертами «|»;

– Заголовок отделяется от остальных строк разделительной строкой, состоящей из дефисов «–» и вертикальных черт.

Разделительная строка:

Каждая ячейка разделителя содержит один или несколько дефисов «–».

Для выравнивания содержимого ячеек используются двоеточия:

– «:---» – выравнивание по левому краю;

– «:---:» – выравнивание по центру;

– «---:»– выравнивание по правому краю.

Если выравнивание не указано, по умолчанию используется левое.

Правила:

– Количество ячеек во всех строках должно быть одинаковым;

– Пустые ячейки возможны (две вертикальные черты подряд);

– Вертикальные черты по краям строки необязательны, но улучшают читаемость.

Пример:

```
| Функция | Описание | Сложность |
|:--------|:--------:|----------:|
| `push()` | Добавляет элемент в конец | O(1) |
| `pop()`  | Удаляет последний элемент | O(1) |
| `search()` | Поиск элемента | O(n) |
```

Рисунок 26 – Разметка таблицы в Markdown

Функция	Описание	Сложность
`push()`	Добавляет элемент в конец	O(1)
`pop()`	Удаляет последний элемент	O(1)
`search()`	Поиск элемента	O(n)

Рисунок 27 – Результат разметки таблицы в предпросмотре

10.Цитаты (блоки с выделением)

Назначение: выделение цитат, важных замечаний, примеров кода (не программного) или фрагментов из внешних источников.

Синтаксис: символ > в начале строки, затем пробел и текст цитаты.

Вложенные цитаты: несколько символов >>, >>> и т.д.

Пример:

```
> Это простая цитата.

> Первый уровень
>> Второй уровень
>>> Третий уровень
```

Рисунок 28 – Разметка блока с выделением

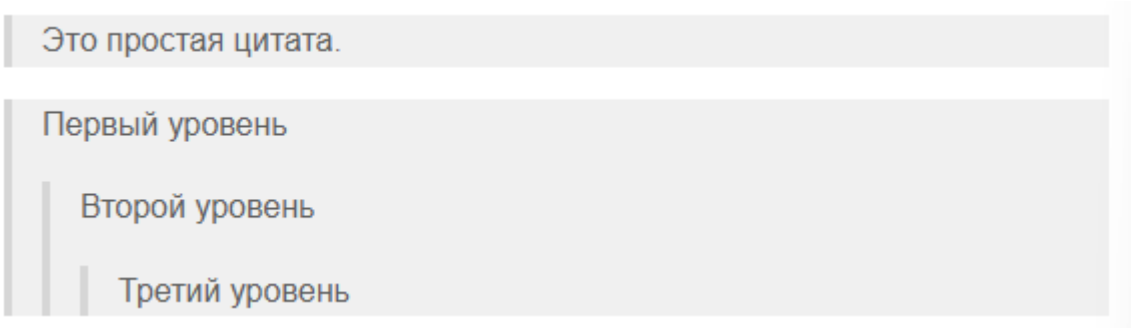


Рисунок 29 – Результат разметки блока с выделением в предпросмотре

11.Сноски

Назначение: добавление пояснений, комментариев или ссылок, которые не прерывают основной текст. Сноски отображаются внизу документа (или в конце раздела).

Синтаксис:

В тексте: [^ключ_сноски] (ключ может быть любым: число, слово, метка).

Внизу документа: [^ключ_сноски]: Текст сноски.

Пример:

```
Здесь упоминается термин[^1], а здесь другой
[^note].

[^1]: Определение термина.
[^note]: Дополнительное замечание.
```

Рисунок 30 – Использование сносок

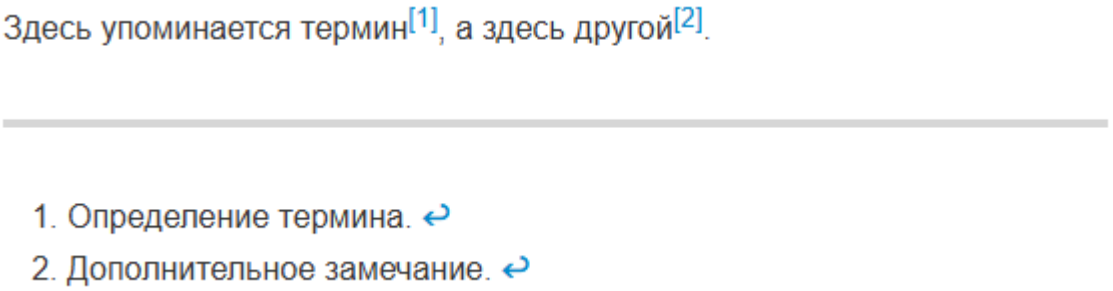


Рисунок 31 – Отображение сносок при рендере документа

12.Списки задач

Назначение: создание чек-листов для отслеживания прогресса (например, планов разработки, списка дел).

Синтаксис: маркированный список, в котором вместо «–» или «*» используется «- [ ]» (невыполненная задача) или «- [x]» (выполненная задача). Между скобками обязательно пробел.

Пример:

```
- [x] Изучить основы Markdown
- [ ] Освоить таблицы
- [ ] Написать README
  - [x] Подготовить заголовки
  - [ ] Добавить изображения
```

Рисунок 32 – Создание списка задач в Markdown

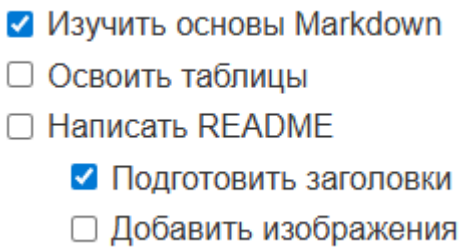


Рисунок 33 – Список задач в предпросмотре

13.Эмодзи

Назначение: добавление графических смайлов для улучшения визуального восприятия (особенно в README на GitHub).

Способы вставки:

1. Кодовые последовательности: :smile:, :rocket:, :warning: и т.д. Список доступных кодов: Emoji Cheat Sheet

2. Прямое копирование символов эмодзи. Это работает везде.

Пример:

```
**Важно:** :warning: Не забудьте сохранить
изменения!

Отлично! :tada:
```

Рисунок 34 – Разметка с использованием кодовых последовательностей для эмодзи

Важно: ⚠️ Не забудьте сохранить изменения!

Отлично! 🎉

Рисунок 35 – Результат использования эмодзи

14.Вставка HTML

Назначение: расширение возможностей Markdown с помощью встроенного HTML (допускается в большинстве реализаций). Позволяет добавить элементы, которых нет в Markdown: цветной текст, кнопки, <kbd>, <details> и т.д.

Синтаксис: любой корректный HTML-код внутри документа Markdown.

Важно: в средах, где Markdown преобразуется в безопасный HTML (например, на форумах), HTML-теги могут быть вырезаны. В локальных редакторах и на GitHub они работают.

Пример:

```
Нажмите <kbd>Ctrl</kbd> + <kbd>C</kbd> для копирования.

<details>
<summary>Подробнее о проекте</summary>
Этот текст скрыт до нажатия на заголовок.
</details>

<span style="color: red;">Красный текст</span> (работает не везде).
```

Рисунок 36 – Использование HTML-тегов в разметке Markdown

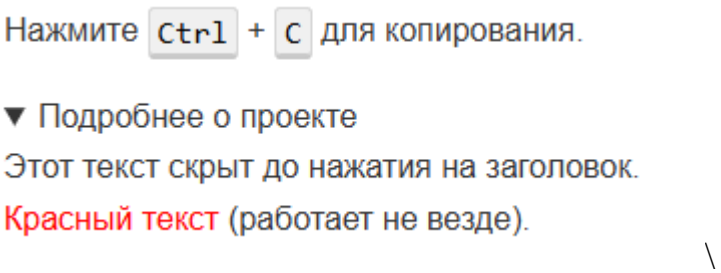


Рисунок 37 – Результат вставки HTML-тегов в документ Markdown

15.Оглавление

Назначение: автоматическое создание ссылок на все заголовки документа для удобной навигации.

Способы создания:

1. Ручной: собрать список ссылок вида [Текст](#заголовок) (пробелы заменяются дефисами, буквы приводятся к нижнему регистру, удаляются знаки препинания).

2. Автоматический (с помощью расширений): в VS Code с плагином Markdown All in One можно выполнить команду «Create Table of Contents». В StackEdit и Typora оглавление создаётся автоматически.

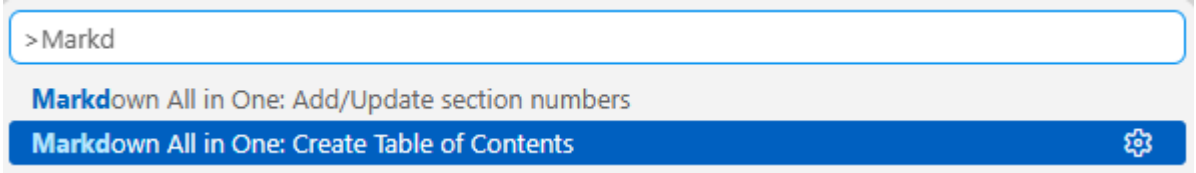
Как это работает:

1) Откройте Markdown-файл (.md).

2) Откройте Палитру команд (Command Palette):

- Windows/Linux: **Ctrl + Shift + P**
- macOS: **Cmd + Shift + P**

3) Введите Markdown All in One: Create Table of Contents и выберите эту команду:



4) Оглавление (TOC) будет вставлено на месте, где находится курсор.

Особенности использования:

– **Автообновление.** По умолчанию оглавление автоматически обновляется при сохранении файла (.md).

– **Игнорирование заголовков.** Чтобы исключить определенный заголовок из оглавления, добавьте <!-- omit from toc --> в конце строки заголовка.

– **Настройка уровней.** Можно настроить, какие уровни заголовков (H1-H6) включать в оглавление, через настройки плагина markdown.extension.toc.levels.

– **Настройка ID.** Для совместимости с GitHub или GitLab в настройках можно изменить slugifyMode.

3. С помощью HTML-якоря: некоторые платформы (GitHub) генерируют оглавление автоматически для README, но стандарт Markdown этого не предусматривает.

Пример ручного оглавления:

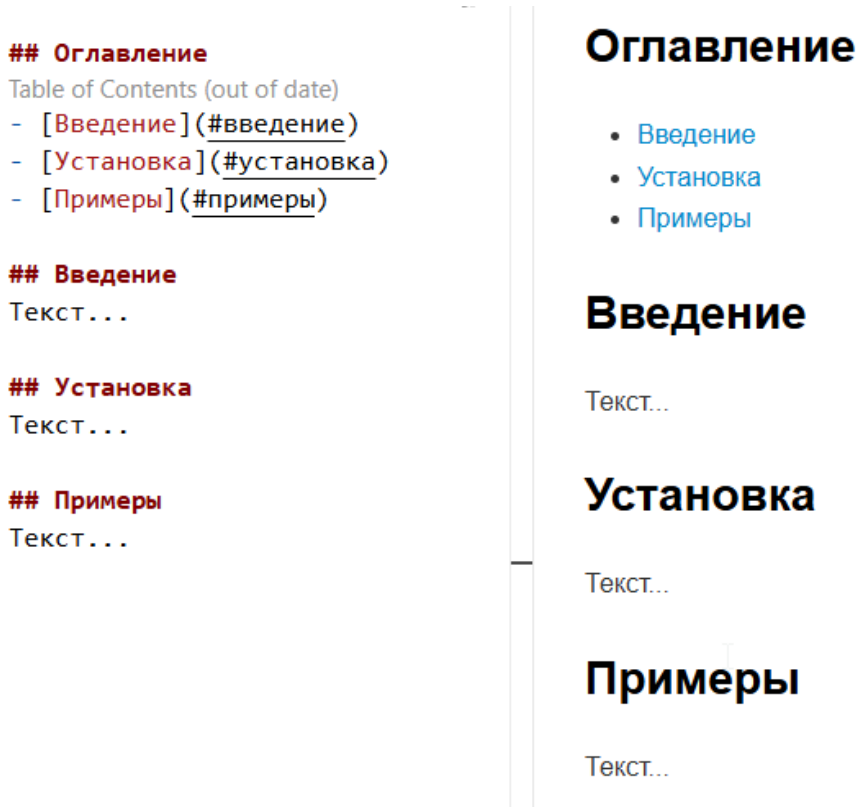


Рисунок 38 – Разметка и рендер ручного оглавления

4. Инструменты для работы с Markdown

Инструмент	Тип	Особенности
Любой текстовый редактор (Блокнот)	Базовый	Без предпросмотра
VS Code + плагин Markdown Preview Enhanced	Редактор кода	Подсветка, предпросмотр, экспорт PDF
Obsidian	Приложение для заметок	Двунаправленные ссылки, граф
Typora	Редактор WYSIWYG	Предпросмотр в реальном времени
GitHub / GitLab	Веб-интерфейс	Автоматический рендеринг README.md
Онлайн-редакторы (StackEdit, Dillinger)	Веб	Не требует установки

Практическая часть

Выполняется **индивидуально.**

1. Задание

Создайте файл README.md для вымышленного программного проекта. Тема проекта – на ваш выбор (например: «Менеджер задач», «Калькулятор калорий», «Бот для погоды», «Генератор паролей», «Консольная игра» и т.д.). Файл должен быть оформлен профессионально и содержать все перечисленные ниже элементы (как изученные в предыдущей работе, так и новые из этой).

Обязательные элементы:

1. Заголовки (H1, H2, H3) – структура документа.
2. Оглавление (ручное или автоматическое) с внутренними ссылками.
3. Абзацы с описанием проекта, его цели и функциональности.
4. Маркированный список (например, особенности проекта).
5. Нумерованный список (например, инструкция по установке).
6. Вложенный список (например, структура папок проекта).
7. Выделение текста (жирный, курсив, жирный курсив).

8. Таблица (например, технические характеристики, версии или список команд).

9. Цитата (например, миссия проекта или цитата из документации).

10. Сноска (пояснение термина или источника).

11. Список задач (task list) – планы на будущее (минимум 3 пункта, один из них выполнен [x]).

12. Эмодзи (минимум 3 разных).

13. Изображение (логотип, скриншот или блок-схема – можно вставить любую картинку по URL).

14. Блок кода (пример кода с указанием языка программирования).

15. Инлайн-код (внутри текста).

16. Ссылки (внешние – на документацию, GitHub, и внутренние – на разделы документа).

17. Горизонтальный разделитель (между логическими частями).

18. HTML-вставка (любые подходящие HTML- теги).

2. Рекомендуемая структура README

```
# Название проекта

Краткое описание (1-2 абзаца).

## Оглавление
- [О проекте](#о-проекте)
- [Возможности](#возможности)
- [Установка](#установка)
- [Пример использования](#пример-использования)
- [Планы по развитию](#планы-по-развитию)
- [Команда](#команда)
- [Лицензия](#лицензия)
```

О проекте

Текст с **выделением** и *курсивом*. Здесь можно разместить цитату.

> Цитата и блок с выделением.

Возможности

- Пункт 1
- Пункт 2
  - Подпункт 2.1
  - Подпункт 2.2

Установка

1. Склонируйте репозиторий: `git clone ...`
2. Установите зависимости:

```bash
pip install -r requirements.txt
```

3. Пошаговое выполнение

1) Выберите тему проекта и придумайте название.

2) Создайте файл с именем README.md (именно так, без вариантов – GitHub и другие платформы распознают его как описание репозитория).

3) Напишите черновик структуры, используя заголовки.

4) Заполните разделы по порядку, добавляя все обязательные элементы.

5) Проверьте предпросмотр – таблицы должны быть выровнены, сноски отображаться внизу, списки задач – с чекбоксами (в GitHub они интерактивны, в VS Code – статичны, но видны).

6) Добавьте эмодзи и HTML-вставки.

7) Убедитесь, что ссылки работают (внутренние ссылки на заголовки должны вести на соответствующие разделы).

8) Сохраните файл.

Требования к отчёту

Отчёт сдаётся индивидуально. Формат – файл с расширением .md (исходник) и/или экспортированный .pdf.

Структура отчёта:

Отчёт должен содержать все элементы, обозначенные в задании, в логической последовательности:

1. Заголовки (название работы, ФИО, группа, дата).
2. Указанные пункты из задания.

Технические требования

1. Исходный .md-файл должен называться PR8_Фамилия_Группа.md.
2. Изображение (скриншот) – приложить отдельным файлом или встроить ссылку на локальный файл (если сдаёте архивом). В онлайн-сдаче можно использовать прямые ссылки на изображения.
3. Экспортированный PDF – желательно, но необязательно, если исходный .md хорошо читается.

Контрольные вопросы

1. Что такое Markdown? Для каких целей он используется в IT-проектах?
2. Как создать заголовок второго уровня в Markdown?
3. Чем отличается нумерованный список от маркированного? Приведите синтаксис.
4. Как вставить гиперссылку с текстом «Нажми сюда» на сайт example.com?
5. Как оформить таблицу из 3 столбцов и 2 строк с выравниванием по центру второго столбца?
6. Как вставить блок кода на языке Python с подсветкой синтаксиса?
7. Какие инструменты для работы с Markdown вы знаете?

8. Что такое UML? Перечислите минимум три типа диаграмм UML и их назначение.

9. Можно ли в Markdown вставлять HTML-теги? Приведите пример.

10. Почему Markdown удобен для хранения документации в системах контроля версий (Git)?

Практическая работа №9

Диаграммы вариантов использования (Use Case)

Цель работы: освоить методологию построения диаграмм вариантов использования (Use Case Diagram) в нотации UML для моделирования функциональных требований к системе. Научиться выделять актёров, прецеденты и отношения между ними, а также создавать диаграммы с использованием текстового инструмента Mermaid.

Задачи:

1. Изучить основные элементы диаграммы вариантов использования: актёр, прецедент, границы системы.
2. Понять и научиться применять отношения между прецедентами: include (включение) и extend (расширение), а также обобщение актёров.
3. Освоить синтаксис Mermaid для построения use case диаграмм.
4. Построить диаграмму вариантов использования для своего проекта на основе ранее собранных функциональных требований (ПР6).
5. Сопоставить диаграмму с текстовыми сценариями (user stories / use case descriptions).

Теоретическая часть

1. Диаграмма вариантов использования: назначение

Диаграмма вариантов использования (Use Case Diagram) – это один из основных типов диаграмм UML, предназначенный для описания функциональных требований к системе с точки зрения внешних пользователей и других систем. Она показывает:

- **Кто** будет взаимодействовать с системой (актёры).
- **Что** система должна делать (прецеденты).
- **Как** прецеденты связаны между собой и с актёрами.

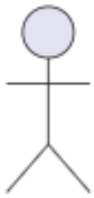
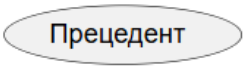
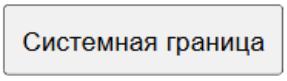
Диаграмма не показывает порядок действий или детали алгоритмов – это задача других диаграмм (последовательности, деятельности). Она даёт **общее представление о функциональности** и служит мостиком между заказчиком и разработчиками.

У диаграммы прецедентов (Use Case Diagram) в UML есть два основных устоявшихся названия, которые часто используются как синонимы:

– Диаграмма прецедентов (от англ. Use Case Diagram);

– Диаграмма вариантов использования (также переводится как Use Case Diagram).

2. Основные элементы диаграммы

Элемент	Обозначение в UML	Описание
Актор (Actor)	Человечек или прямоугольник с текстом «Актор» 	Роль пользователя или внешней системы, взаимодействующей с системой. Не обязательно человек – может быть внешний сервис, таймер.
Прецедент (Use Case)	Овал с названием внутри 	Действие, которое система выполняет по запросу актёра. Название – глагол + существительное («Зарегистрироваться», «Оплатить заказ»).
Границы системы	Прямоугольник с названием системы внутри 	Отделяет внутренние прецеденты от внешних актёров.
Отношение ассоциации	Сплошная линия 	Связывает актёра с прецедентом, указывая, что актёр инициирует или участвует в этом варианте использования.

Отношение включения (include)	Пунктирная стрелка с меткой <<include>> 	Указывает, что один прецедент всегда включает в себя другой. Базовый прецедент не может быть выполнен без включённого.
Отношение расширения (extend)	Пунктирная стрелка с меткой <<extend>> 	Указывает, что один прецедент расширяет другой в определённых условиях (дополнительное поведение).
Обобщение (generalization)	Пустая треугольная стрелка (от потомка к родителю)	Применяется для актёров: «Администратор» – обобщение для «Менеджера» и «Аналитика». Или для прецедентов (реже).

3. Отношения include и extend (подробно)

3.1. Ассоциация (Association)

Показывает, что актор участвует в данном прецеденте. Изображается сплошной линией между актором и прецедентом.

3.2. Include (включение)

Используется, когда один прецедент **всегда** вызывает другой. Включённый прецедент – это подпрограмма или общая часть поведения.

Обозначает, что один прецедент (базовый) включает поведение другого прецедента (включаемого) всегда, когда выполняется базовый. Включаемый прецедент не может существовать сам по себе.

Графически: пунктирная стрелка с надписью <<include>>, направленная от базового к включаемому.

Пример: «Оформить заказ» всегда включает «Аутентифицироваться», если гость не может оформить заказ.

Правило: Стрелка идёт от базового прецедента к включаемому (базовый <<include>> включаемый).

3.3. Extend (расширение)

Используется, когда один прецедент **иногда** (при выполнении условия) расширяет поведение другого. Расширяющий прецедент – это опциональное дополнение.

Показывает, что один прецедент (расширяющий) может дополнять поведение другого прецедента (расширяемого) при определённых условиях (в точке расширения).

Графически: пунктирная стрелка с надписью <<extend>>, направленная от расширяющего к расширяемому.

Пример: «Оформить заказ» может расширяться прецедентом «Применить промокод», если пользователь ввёл промокод.

Правило: Стрелка идёт от расширяющего прецедента к базовому (расширяющий <<extend>> базовый).

3.4. Обобщение (Generalization)

Применяется как к акторам, так и к прецедентам. Означает, что дочерний элемент наследует поведение родительского и может его дополнять или уточнять.

Графически: сплошная линия с пустой треугольной стрелкой, направленная от потомка к родителю.

4. Построение диаграммы: пошаговый алгоритм

1. Определите границы системы – что входит в вашу систему, а что вне её.

2. Найдите акторов – кто будет использовать систему? (пользователи, администраторы, внешние системы).

3. Выделите прецеденты – что каждый актёр хочет сделать с помощью системы? Используйте список функциональных требований (Must have из MoSCoW).

4. Свяжите актёров с прецедентами линиями ассоциации.

5. Выявите общие части – если одинаковое поведение встречается в нескольких прецедентах, вынесите его в отдельный прецедент и свяжите через <<include>>.

6. Выявите опциональные расширения – если есть действия, которые выполняются только при определённых условиях, свяжите через <<extend>>.

7. Упростите – не перегружайте диаграмму; при необходимости сделайте несколько диаграмм (для разных групп актёров).

5. Инструменты Mermaid PlantUML для построения use case диаграмм

Mermaid – это язык разметки для создания диаграмм и схем непосредственно в текстовом виде. Позволяет генерировать диаграммы, графики и схемы (блок-схемы, последовательности, классы) из простого текстового описания, напоминающего Markdown.

Диаграммы описываются с помощью простого синтаксиса, который затем автоматически преобразуется в графическое изображение (SVG).

Mermaid поддерживается во многих платформах: GitHub, GitLab, Notion, а также в редакторах Markdown (VS Code с расширениями, Typora, Obsidian и др.).

Преимущества Mermaid:

– Не требует внешних графических редакторов – всё в тексте;

– Диаграммы являются частью документации и могут версионироваться в Git;

– Поддерживает множество типов диаграмм: блок-схемы (Flowchart), диаграммы последовательности (Sequence), диаграммы классов (Class), диаграммы состояний состояний (State), диаграммы прецедентов, диаграммы Ганта и др;

– Легко встраивается в Markdown-документы (GitHub, GitLab, Obsidian) с помощью блока ```mermaid, работает в браузере и имеет онлайн-редактор Mermaid Live Editor: https://mermaid.live/

5.1. Основные типы диаграмм Mermaid

Тип диаграммы		Описание	Ключевое слово
Блок-схема	Flowchart	Алгоритмы, процессы, логические цепочки	graph / flowchart
Диаграмма взаимодействия	Sequence diagram	Взаимодействие объектов во времени	sequenceDiagram
Диаграмма классов	Class diagram	Структура классов в ООП	classDiagram
Диаграмма состояния	State diagram	Состояния и переходы	stateDiagram-v2
ER-диаграмма	Entity Relationship Diagram (ER-diagram)	Схемы баз данных	erDiagram
Диаграмма Ганта	Gantt chart	Планирование задач во времени	gantt

5.2. Синтаксис диаграммы прецедентов в PlantUML

PlantUML – это инструмент для создания диаграмм на основе текстового описания. Для построения use case диаграммы используется ключевое слово @startuml и @enduml. Основные элементы:

5.2.1. Определение актора

Актор может быть представлен текстом. Можно задать описание в квадратных скобках. В PlantUML акторы отображаются в виде человечков.

Синтаксис: actor Имя_актора [as псевдоним]

Пример:

actor Пользователь as user

actor Администратор as admin

5.2.2. Определение прецедента

Название может содержать пробелы и специальные символы, если заключено в кавычки.

Синтаксис: usecase Название_прецедента [as псевдоним]

Пример:

usecase "Вход в систему" as Login

usecase "Регистрация" as Register

5.3. Системная граница

Граница системы задаётся с помощью ключевого слова rectangle. Прецеденты, заключённые в фигурные скобки внутри rectangle, будут находиться внутри системной границы.

Синтаксис:

```
rectangle "Название системы" {
   usecase ...
   usecase ...
}
```

5.4. Отношения

Тип	Синтаксис	Описание
Ассоциация	Актор --> Прецедент	Линия между актором и прецедентом
Включение	Базовый <|-- Включаемый : <<include>>	Пунктирная стрелка от включаемого к базовому прецеденту (направление стрелки – к базовому)
Расширение	Расширяющий ..> Расширяемый : <<extend>>	Пунктирная стрелка от расширяющего к расширяемому прецеденту (направление стрелки – к расширяемому)
Обобщение	Дочерний --|> Родительский	Стрелка от потомка к родителю – сплошная линия с треугольной стрелкой

Важно: в PlantUML стрелки отношений строятся по принципу: <|-- означает, что стрелка направлена справа налево; --> – слева направо. Направление можно менять.

Рекомендуемый синтаксис:

actor User

actor Admin

usecase (Login) as UC_Login

usecase (Register) as UC_Register

User --> UC_Login

Admin --> UC_Login

Admin --> UC_Register

UC_Login <|-- UC_Register : <<include>>

5.5. Полный пример диаграммы прецедентов

usecase.puml

```
@startuml PR4_Example_Diagram
actor "Покупатель" as customer
actor "Администратор" as admin

rectangle "Интернет-магазин" {
    usecase "Поиск товаров" as Search
    usecase "Добавление в корзину" as AddToCart
    usecase "Оформление заказа" as Checkout
    usecase "Оплата заказа" as Pay
    usecase "Управление каталогом" as ManageCatalog
    usecase "Управление заказами" as ManageOrders
    usecase "Авторизация" as Auth
}

customer --> Search
customer --> AddToCart
customer --> Checkout
customer --> Pay
admin --> ManageCatalog
admin --> ManageOrders
admin --> Auth

Checkout <|-- Auth : <<include>>
Pay <|-- Auth : <<include>>
Pay <|-- Checkout : <<extend>>
@enduml
```

Рисунок 1 – Синтаксис создания диаграммы вариантов использования с помощью PlantUML

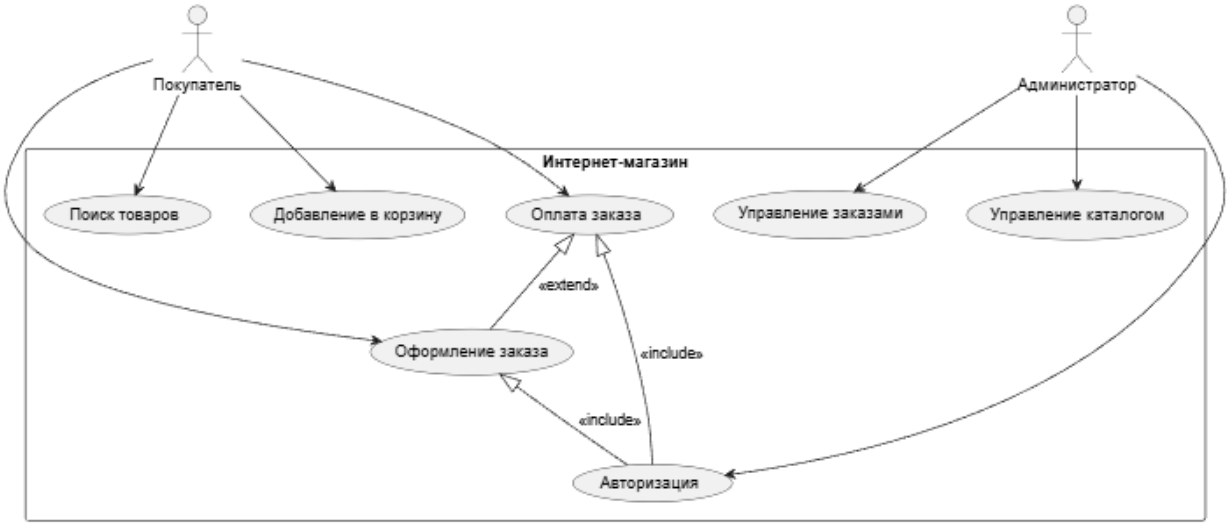


Рисунок 2 – Графическое представление диаграммы вариантов использования (прецедентов)

5.6. Правила построения качественной диаграммы прецедентов

1. Акторы именуются существительными, отражающими роль (не должность, а роль в системе). Актор всегда находится вне системы.

2. Прецеденты именуются глаголами с существительными (например, «Создать заказ», «Просмотреть отчёт»). Они описывают что делает система, а не как.

3. Границы системы показывают, что находится внутри разрабатываемой системы, а что – снаружи.

4. Отношения:

– Используйте include, когда один прецедент выполняется всегда при выполнении другого (общий фрагмент).

– Используйте extend, когда один прецедент может дополнять другой в особых случаях (условное расширение).

– Не перегружайте диаграмму излишними отношениями.

5. Количество – оптимально 5–15 прецедентов на одной диаграмме. При большем количестве следует группировать.

5.7. Ограничения Mermaid для диаграмм прецедентов

Mermaid **не поддерживает** синтаксис для полноценных use case диаграмм. Ключевые слова usecase, actor и rectangle в Mermaid не работают для этого типа диаграмм. В Mermaid можно лишь создать приблизительную визуализацию, используя обычные графы (graph), но это не соответствует стандарту UML и не позволяет задавать корректные отношения include и extend.

Пример приближённой диаграммы в Mermaid (только для ознакомления):

```mermaid
graph TD
    user((Пользователь))
    admin((Администратор))

    subgraph "Система управления задачами"
        UC_Login(Вход в систему)
        UC_Tasks(Управление задачами)
        UC_Users(Управление пользователями)
        UC_Auth(Аутентификация)
    end

    user --- UC_Login
    user --- UC_Tasks
    admin --- UC_Login
    admin --- UC_Users
    admin --- UC_Tasks

    UC_Tasks -.->|include| UC_Auth
    UC_Users -.->|include| UC_Auth
```

Рисунок 3 – Синтаксис построения графа в Mermaid

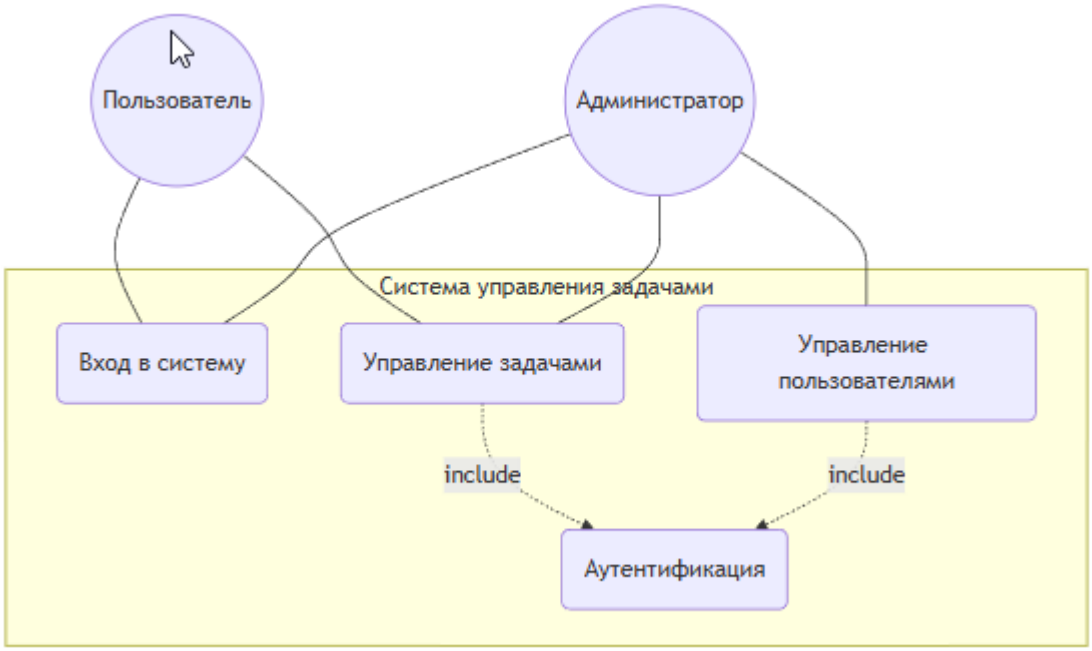


Рисунок 4 – Графическое представление приближённой диаграммы вариантов использования (прецедентов) в Mermaide

В этом примере:

- Акторы изображены в виде окружностей (не человечков);
- Системная граница – через subgraph;
- Отношения – пунктирные стрелки с подписями.

Однако такой подход не является полноценной use case диаграммой и не поддерживает все необходимые элементы UML. Поэтому для выполнения задания мы будем использовать PlantUML.

Практическая часть

1. Задание

Разработайте полноценную диаграмму прецедентов в PlantUML для выбранной предметной области. Предметную область можно выбрать из предложенного списка или согласовать свою с преподавателем.

Варианты предметных областей:

1. Интернет-магазин – акторы: Покупатель, Администратор, Платёжная система. Прецеденты: поиск товаров, добавление в корзину, оформление заказа, оплата, управление каталогом и т.д.

2. Библиотечная система – акторы: Читатель, Библиотекарь. Прецеденты: поиск книг, бронирование, выдача книг, возврат, управление фондом.

3. Система управления задачами (Trello-like) – акторы: Пользователь, Администратор. Прецеденты: создание задачи, назначение исполнителя, изменение статуса, создание проектов.

4. Банкомат (ATM) – акторы: Клиент, Инкассатор, Банковская система. Прецеденты: проверка баланса, снятие наличных, пополнение, авторизация.

5. Система онлайн-обучения – акторы: Студент, Преподаватель, Администратор. Прецеденты: просмотр курсов, прохождение тестов, загрузка материалов, выставление оценок.

2. Требования к диаграмме

Диаграмма должна содержать:

1. Не менее 3 акторов (один из них – внешняя система или роль).

2. Не менее 5 прецедентов.

3. Системную границу (прямоугольник) с названием системы.

4. Отношения:

– Ассоциации между акторами и прецедентами (каждый актор должен быть связан хотя бы с одним прецедентом).

– Хотя бы одно отношение include.

– Хотя бы одно отношение extend (если уместно) или generalization (для акторов или прецедентов).

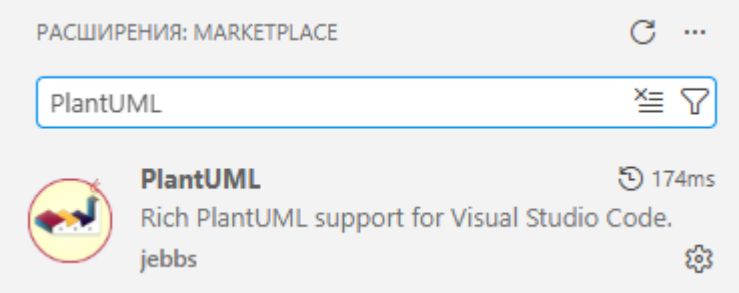
5. Корректные имена акторов и прецедентов (существительные для акторов, глаголы для прецедентов).

3. Подготовка рабочей среды

Вариант 1: Локальный редактор с PlantUML (VS Code)

1. Установите Visual Studio Code.

2. Установите расширение PlantUML (автор: jebbs). Это расширение позволяет рендерить диаграммы PlantUML внутри VS Code.



3. Для работы расширения потребуется Java Runtime Environment (JRE) – установите, если ещё нет.

4. Создайте файл с расширением .puml (например, usecase.puml).

5. Для предпросмотра используйте сочетание Alt + D (или через команду «Preview Current Diagram»).

Вариант 2: Онлайн-редактор

Используйте PlantUML Online Server (https://www.plantuml.com/plantuml/uml/) – введите код и получите изображение.

Альтернативно: PlantUML Viewer (https://plantuml.com/).

Вариант 3: Встроенный в Markdown (с расширением Markdown Preview Enhanced)

В VS Code с расширением Markdown Preview Enhanced можно вставлять блоки ```puml, и они будут отображаться, если установлен PlantUML и Java.

4. Пошаговое выполнение

Шаг 1. Выбор предметной области

Выберите одну из предложенных тем или свою. Продумайте, кто будет пользоваться системой (акторы), какие функции она должна выполнять (прецеденты).

Шаг 2. Создание файла

Создайте файл **usecase_ФИО.puml** (или **usecase_ФИО.md** с блоком PlantUML, если используете Markdown с поддержкой).

Шаг 3. Выделение актёров

На основе вашего проекта составьте список актёров. Заполните таблицу:

Актёр	Описание (роль, цель)	Пример для проекта «Запись к врачу»
...	...	Пациент – человек, который хочет записаться на приём
...	...	Врач – просматривает расписание, подтверждает приёмы
...	...	Администратор – управляет расписанием, отчётами

Минимум 2 актёра, максимум 5 (не перегружайте).

Шаг 4. Выделение прецедентов

Из списка функциональных требований (Must have и Should have из ПР6) выберите **5–7 ключевых прецедентов**. Запишите их в виде «глагол + существительное»:

Актёр	Описание (роль, цель)	Пример для проекта «Запись к врачу»
...	...	...

Шаг 5. Определение отношений

Проанализируйте прецеденты:

1. Найдите **общие части** (например, «Аутентифицироваться» нужно для многих прецедентов). Выделите их в отдельные прецеденты и отметьте как <<include>>.

2. Найдите **опциональные** действия (например, «Применить скидочный купон» при оформлении заказа). Отметьте как <<extend>>.

3. Если есть иерархия актёров (например, «Зарегистрированный пользователь» наследует возможности «Гостя»), добавьте обобщение.

Заполните таблицу отношений:

Тип отношения	**Базовый элемент**	**Включаемый/расширяющий элемент**	**Условие (для extend)**
include	...	...	...
extend	...	...	...

Шаг 6. Описание диаграммы в коде

Напишите код диаграммы в соответствии с синтаксисом PlantUML. Начните с @startuml и закончите @enduml.

Шаг 7. Проверка отображения

Откройте предпросмотр диаграммы. Убедитесь, что все элементы отображаются корректно, акторы – в виде человечков, прецеденты – в овалах, отношения – с правильными стрелками и надписями.

Шаг 8. Добавление текстового описания

В отдельном файле (например, **usecase_description_ФИО.md**) добавьте описание предметной области, акторов, прецедентов и пояснения к отношениям.

Шаг 9. Сохранение и сдача

Сохраните файл с диаграммой (**.puml**) и файл с пояснениями – **usecase_ФИО.puml** и **usecase_description_ФИО.md** соответственно.

5. Пример выполнения (Банкомат)

Файл **usecase_Иванов И.И..puml**:

```
@startuml
actor Клиент as client
actor Инкассатор as collector
actor БанковскаяСистема as bankSystem

rectangle "Банкомат" {
   usecase "Ввод карты и PIN" as Auth
   usecase "Проверка баланса" as Balance
   usecase "Снятие наличных" as Withdraw
   usecase "Пополнение счёта" as Deposit
   usecase "Печать чека" as PrintReceipt
   usecase "Загрузка наличных" as LoadCash
   usecase "Сбор выручки" as CollectCash
}

client --> Auth
client --> Balance
client --> Withdraw
client --> Deposit
client --> PrintReceipt
collector --> LoadCash
collector --> CollectCash
bankSystem --> Auth
bankSystem --> Withdraw
bankSystem --> Deposit

Withdraw <|-- Auth : <<include>>
Deposit <|-- Auth : <<include>>
```

Balance <|-- Auth : <<include>>

PrintReceipt <|-- Withdraw : <<extend>>

PrintReceipt <|-- Deposit : <<extend>>

@enduml

Сопроводительный файл **usecase_description_Иванов И.И..md** (пример):

Диаграмма прецедентов: Банкомат (ATM)

Описание предметной области

Система банкомата предоставляет клиентам возможность проводить банковские операции без участия сотрудника. Инкассаторы обслуживают устройство (загрузка наличных, сбор выручки). Банковская система обрабатывает транзакции.

Акторы

- **Клиент** – физическое лицо, использующее банкомат.
- **Инкассатор** – сотрудник банка, обслуживающий устройство.
- **Банковская система** – внешняя система, обрабатывающая транзакции.

Прецеденты

- **Ввод карты и PIN** – аутентификация клиента.
- **Проверка баланса** – запрос остатка на счёте.
- **Снятие наличных** – выдача денег.
- **Пополнение счёта** – внесение наличных.
- **Печать чека** – выдача чека после операции.
- **Загрузка наличных** – пополнение купюр в банкомате.
- **Сбор выручки** – изъятие излишков.

Отношения

- **include** – все основные операции (проверка баланса, снятие, пополнение) требуют предварительной аутентификации.

- **extend** – печать чека возможна только после выполнения снятия или пополнения.

Требования к отчёту

Отчёт сдаётся **индивидуально**. Формат – Markdown (.md), экспортированный PDF и файл с диаграммой в формате **.puml**. В отчёте должны быть текст и диаграмма.

Структура отчёта:

1. Титульная информация (название работы, ФИО, группа, название предметной области) – в виде заголовков Markdown.

2. Цель и задачи (кратко, можно скопировать из методички).

3. Таблица актёров (задание 1).

4. Таблица прецедентов (задание 2).

5. Таблица отношений (задание 3).

6. Диаграмма вариантов использования.

7. **Вывод** (чему научились, какие трудности возникли при моделировании).

Технические требования

1. Файл отчёта **PR9_Фамилия_Группа.md**.

2. При использовании скриншота – вставить его через ![Диаграмма](screenshot.png). Файл скриншота приложить к сдаче.

Контрольные вопросы

1. Что такое диаграмма прецедентов и для чего она используется?
2. Чем актор отличается от прецедента?
3. Что обозначает системная граница?
4. В чём разница между отношениями include и extend?
5. Почему Mermaid не подходит для построения полноценных use case диаграмм?
6. Какой синтаксис PlantUML используется для определения актора и прецедента?
7. Как в PlantUML задать отношение include?
8. Какие правила именования акторов и прецедентов вы знаете?

Практическая работа №10

Детализация сценариев

Цель работы: научиться детализировать варианты использования (use cases) путём описания потоков событий: основного (успешного) и альтернативных (включая ошибочные). Освоить структурированный формат описания сценария для ключевого прецедента проекта.

Задачи:

1. Изучить понятие «поток событий» в рамках варианта использования.
2. Освоить формат описания основного (счастливого) потока и альтернативных потоков (исключения, ошибки, ветвления).
3. Научиться выделять предусловия, постусловия и триггеры сценария.
4. Составить детализированный сценарий (спецификацию) для одного ключевого варианта использования своего проекта.
5. Связать текстовый сценарий с диаграммой use case, построенной в ПР9.

Теоретическая часть

1. Что такое детализация сценария

Вариант использования (use case) на диаграмме – это только «шапка»: название и связи. Чтобы он стал полезным для разработчиков и тестировщиков, его нужно **детализировать** – описать последовательность действий актёра и системы в виде текстового сценария. Такой документ называют **спецификацией варианта использования (use case specification)**.

Спецификация отвечает на вопросы:

- Как именно начинается взаимодействие?
- Какие шаги выполняет актёр, а какие – система?
- Что может пойти не так?
- Какое состояние системы после успешного завершения?

2. Структура спецификации варианта использования

Стандартный шаблон (по А. Коберну, А. Якобсону и др.):

Элемент	Описание
Название	Название прецедента (глагол + существительное)
ID	Уникальный идентификатор (например, UC-01)
Актёры	Кто инициирует (основной актёр) и вспомогательные
Предусловия	Что должно быть истинно до начала сценария
Постусловия (успех)	Состояние системы после успешного выполнения
Постусловия (отказ)	Состояние при неудаче (если отличается)
Триггер	Событие, запускающее прецедент
Основной поток (Basic Flow)	Нумерованный список шагов, ведущих к успеху (счастливый путь)
Альтернативные потоки (Alternative Flows)	Другие ветви, включая исключения и ошибки
Бизнес-правила (опционально)	Дополнительные ограничения

3. Правила написания шагов основного потока

- Каждый шаг – одно простое действие.
- Используйте **активный глагол**.
- Чётко указывайте, кто выполняет действие: **Актёр** или **Система**.
- Нумеруйте шаги (1, 2, 3…).
- Избегайте технических деталей (базы данных, API) – оставляйте их для проектирования.

Пример плохого шага:
«Система проверяет введённый логин в таблице Users и отправляет запрос на аутентификацию через API.»

Пример хорошего шага:
«Система проверяет правильность введённых учётных данных и авторизует пользователя.»

4. Основной поток (Basic Flow)

Это **счастливый путь** – всё идёт по плану, без ошибок и отклонений. Шаги обычно нумеруются от 1 до N.

Пример для прецедента «Зарегистрироваться»:

Основной поток:

1. Пользователь открывает форму регистрации.
2. Система отображает поля: Имя, Email, Пароль, Подтверждение пароля.
3. Пользователь заполняет все поля корректными данными и нажимает «Зарегистрироваться».
4. Система проверяет, что email не занят.
5. Система создаёт новую учётную запись.
6. Система отправляет письмо с подтверждением на указанный email.
7. Система отображает сообщение «Регистрация успешна. Подтвердите email».
8. Сценарий завершается.

5. Альтернативные потоки (Alternative Flows)

Альтернативные потоки описывают **отклонения** от основного сценария. Они делятся на:

- **Ветвления** – пользователь выбирает другой путь (например, «Запомнить пароль»).
- **Ошибки** – некорректный ввод, нарушение условий.
- **Исключения** – внешний сбой (нет соединения, сервер не отвечает).

Каждый альтернативный поток получает **идентификатор** (например, A1, A2) и ссылается на шаг основного потока, после которого он возникает.

Пример для прецедента «Зарегистрироваться»:

Альтернативный поток A1 (Email уже занят):

- *Возникает после шага 4 основного потока.*
- Система отображает сообщение «Пользователь с таким email уже существует».

- Сценарий возвращается к шагу 2 (поля остаются заполненными).

Альтернативный поток А2 (Пароли не совпадают):

- *Возникает после шага 3 (когда пользователь нажал «Зарегистрироваться»).*
- Система проверяет совпадение паролей и обнаруживает несовпадение.
- Система подсвечивает поля паролей красным и показывает сообщение «Пароли не совпадают».
- Сценарий возвращается к шагу 3 (пользователь исправляет пароли).

6. Предусловия и постусловия

Предусловие – условие, которое обязательно должно выполняться до начала прецедента. Если оно не выполнено, прецедент не может стартовать (или должен быть вызван другой прецедент).

Постусловие – состояние системы после завершения прецедента (успешного или неуспешного).

Пример для прецедента «Оплатить заказ»:

- *Предусловие:* Пользователь аутентифицирован, корзина не пуста.
- *Постусловие (успех):* Заказ переведён в статус «Оплачен», списаны средства со счёта пользователя.
- *Постусловие (отказ):* Заказ остаётся в статусе «Ожидает оплаты», средства не списаны.

7. Связь с диаграммой use case и требованиями

Детальный сценарий соответствует **одному прецеденту** на диаграмме. Отношения include и extend реализуются через вызовы: если прецедент включает другой, то в основном потоке может быть шаг «Выполнить включённый прецедент [название]» без детализации его шагов (дублирования не нужно).

Кроме того, каждый шаг сценария может быть прослежен до функционального требования из ПР6.

Практическая часть

Выполняется **индивидуально**. Используется проект и диаграмма use case из ПР9.

Задание 1. Выбор ключевого варианта использования

Из списка прецедентов вашего проекта выберите **один самый важный** (как правило, тот, который отражает основную ценность продукта). Например, для приложения записи к врачу – «Записаться на приём». Для интернет-магазина – «Оформить заказ».

Запишите:

- **Название прецедента**.
- **ID** (например, UC-01).
- **Актёр(ы)**.

Задание 2. Определение предусловий, постусловий и триггера

Заполните для выбранного прецедента:

Элемент	Ваш вариант
Триггер (что запускает сценарий)	(например: «Пользователь нажимает кнопку "Записаться" после выбора врача»)
Предусловия (что должно быть истинно)	(например: «Пользователь аутентифицирован», «Выбран врач и дата»)
Постусловие (успех)	(например: «Создана запись в расписании, пользователь получил подтверждение»)
Постусловие (отказ)	(например: «Запись не создана, пользователь видит сообщение об ошибке»)

Задание 3. Описание основного потока

Напишите **основной поток** (счастливый путь) из 5–10 шагов. Используйте нумерацию. Для каждого шага указывайте действующее лицо: **Пользователь:** или **Система:**.

Шаблон:

Основной поток:

1. **Пользователь:** [действие]
2. **Система:** [реакция]
3. **Пользователь:** [действие]

   и т.д.

Требования:

- Шаги должны быть логически неделимы.
- Не смешивайте действия пользователя и системы в одном шаге.
- Фиксируйте переходы на другие экраны / состояния.

Задание 4. Описание альтернативных потоков

Выделите **минимум 2 альтернативных потока**. Для каждого укажите:

- **ID** (A1, A2…).
- **Точку вставки** (после какого шага основного потока возникает).
- **Условие** (что пошло не так или какое решение принял пользователь).
- **Последовательность шагов** (что происходит вместо основного потока или после него).
- **Куда переходит сценарий** (завершается, возвращается к определённому шагу, вызывает другой прецедент).

Пример оформления:

Альтернативный поток A1 (Недостаточно средств на счёте)

- *Возникает после шага 5 основного потока (Система пытается списать средства).*
- Система проверяет баланс и обнаруживает недостаток средств.
- Система отображает сообщение «Недостаточно средств. Пополните баланс».
- Сценарий завершается (заказ не оформлен).

Альтернативный поток A2 (Пользователь отменяет операцию)

- *Возникает на любом шаге до подтверждения (шаги 1–4).*
- Пользователь нажимает кнопку «Отмена».

- Система возвращает пользователя на предыдущую страницу без сохранения изменений.
- Сценарий завершается.

Задание 5. Проверка полноты

Используйте **чек-лист** для самопроверки вашего сценария:

- Каждый шаг основного потока начинается с активного глагола?
- Все ли альтернативные потоки имеют точку вставки?
- Для каждого альтернативного потока указано, куда переходит управление (завершение, возврат к шагу N)?
- Учтены ли ошибки ввода пользователя?
- Учтено ли сетевое соединение (при необходимости)?
- Сценарий не содержит технических деталей (SQL, API)?

Задание 6. Оформление в Markdown

Оформите спецификацию варианта использования в отдельном .md-файле (или как раздел отчёта). Используйте заголовки, таблицы, списки. Приложите ссылку на диаграмму use case из ПР9, к которой относится этот сценарий.

Пример шапки файла:

```
# Спецификация варианта использования: Записаться на приём

**ID:** UC-03
**Актёры:** Пациент (основной), Система расписания (внешняя)
**Приоритет:** Must have (MVP)

## Предусловия
- Пользователь аутентифицирован в системе.
- Выбран врач и временной слот.
```

Постусловия

Успех

- Создана запись на приём с уникальным номером.
- Пациенту отправлено push-уведомление.

Отказ

- Запись не создана.
- Система показывает сообщение об ошибке.

Триггер

Пациент нажимает кнопку «Подтвердить запись» на экране выбора времени.

Основной поток

1. **Система:** Проверяет, что выбранный слот ещё свободен.
2. **Система:** Создаёт запись в базе данных (статус "подтверждена").
3. **Система:** Отправляет push-уведомление пациенту.
4. **Система:** Отображает экран «Запись успешно создана» с номером талона.

Альтернативные потоки

A1. Выбранный слот уже занят (возникает после шага 1)

- Система отображает сообщение «Это время уже занято, выберите другое».
- Сценарий возвращается к шагу выбора времени (повторный выбор).

A2. Ошибка отправки уведомления (возникает после шага 3)

- Система логирует ошибку, но запись сохраняется.
- Сценарий продолжается с шага 4 (уведомление не критично).

Бизнес-правила

- Пациент не может записаться к врачу более чем на 30 минут вперёд.

- Запись возможна не более чем за 14 дней вперёд.

Задание 7. Связь с требованиями

В конце спецификации добавьте таблицу трассировки: какие функциональные требования (из ПР6) покрывает этот сценарий.

ID требования	Описание требования	Шаг(и) сценария
F1	Аутентификация	Предусловие
F5	Создание записи	Шаг 2

Требования к отчёту

Отчёт сдаётся индивидуально. Формат – Markdown (.md) или экспортированный PDF.

Структура отчёта:

1. Титульная информация (название работы, ФИО, группа).
2. Цель и задачи (кратко).
3. Выбор прецедента (название, ID, актёры).
4. Таблица предусловий, постусловий, триггера (задание 2).
5. Основной поток (нумерованный список, задание 3).
6. Альтернативные потоки (не менее 2, задание 4).
7. Чек-лист проверки (подтверждение, что проверено, задание 5).
8. Полная спецификация (оформленный вариант из задания 6).
9. Таблица трассировки (задание 7).
10. Вывод (что нового узнали, какие трудности при детализации).

Технические требования

1. Файл: **PR10_Фамилия_Группа.md**.

2. Допускается вставка скриншотов (например, фрагментов), но предпочтителен текст.

3. Если спецификация очень большая, можно вынести её в отдельный файл и дать ссылку.

Контрольные вопросы

1. Что такое спецификация варианта использования? Чем она отличается от диаграммы use case?

2. Перечислите основные элементы спецификации (не менее 5).

3. Что такое «основной поток»? Как его иначе называют?

4. Что такое «альтернативный поток»? Приведите пример.

5. Зачем нужны предусловия и постусловия?

6. В чём разница между постусловием успеха и постусловием отказа?

7. Как обозначается точка вставки альтернативного потока?

8. Может ли альтернативный поток завершаться возвратом не к началу, а к конкретному шагу? Приведите пример.

9. Как связаны детальные сценарии с отношениями include и extend на диаграмме?

Практическая работа №11

Понятие класса в UML

Цель работы: освоить базовые понятия объектно-ориентированного моделирования: класс, объект, атрибут, метод. Научиться представлять сущности предметной области в виде классов UML и понимать диаграмму классов как модель данных для программной системы.

Задачи:

1. Изучить понятия «класс», «объект», «атрибут», «метод (операция)» в контексте UML.

2. Познакомиться с нотацией изображения класса на диаграмме (секции: имя, атрибуты, методы).

3. Научиться различать типы атрибутов и методов (видимость, типы данных).

4. Выполнить упражнения по выделению классов, атрибутов и методов из текстового описания предметной области (без связей между классами).

Теоретическая часть

1. Объектно-ориентированное моделирование

Объектно-ориентированное моделирование (ООМ) – это подход к описанию системы, при котором мир представляется как совокупность взаимодействующих объектов, каждый из которых обладает состоянием (данными) и поведением (операциями). UML предоставляет диаграмму классов – основной инструмент для статического описания структуры системы.

2. Класс и объект

Класс – это шаблон (описание) для создания объектов. Он определяет, какие данные (атрибуты) и какое поведение (методы) будут у всех объектов этого класса.

Объект – это конкретный экземпляр класса, имеющий собственные значения атрибутов.

Пример:

Класс Студент описывает, что у любого студента есть имя, возраст, номер группы.

Объекты: студент Иван (18 лет, группа ПИ-101), студент Мария (19 лет, группа ПИ-102).

3. Атрибуты (свойства)

Атрибут – это характеристика класса, которая описывает состояние объекта. Атрибуты имеют:

- **Имя** (например, имя, возраст).
- **Тип** (например, String, int, Date).
- **Видимость** (обычно - для private, + для public, # для protected).

Формат записи атрибута в UML: [видимость] имя : тип [= значение_по_умолчанию]

Пример:- имя : String + возраст : int = 18

4. Методы (операции)

Метод – это действие, которое может выполнять объект класса. В UML методы называют операциями. Метод имеет:

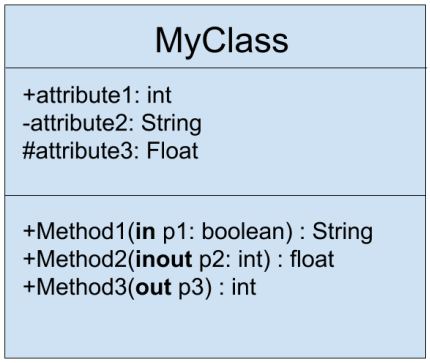
- **Имя** (глагол, например увеличитьВозраст()).
- **Список параметров** (в скобках).
- **Тип возвращаемого значения** (если есть).
- **Видимость**.

Формат записи метода: [видимость] имя(параметры) : возвращаемый_тип

Пример: + увеличитьВозраст(лет : int) : void+ получитьИмя() : String

5. Графическое изображение класса на диаграмме

Класс изображается прямоугольником, разделённым на три секции:



Секция имени – обязательно;

Секция атрибутов – может быть пустой (показываются только значимые атрибуты);

Секция методов – может быть пустой или показывать только ключевые методы.

6. Типы данных в UML

UML не привязан к конкретному языку программирования. Используются стандартные типы:

- String – строка
- Integer, int – целое число
- Boolean – истина/ложь
- Date – дата
- Real – вещественное число
- А также пользовательские классы (например, Студент, Группа).

7. Диаграмма классов как модель данных

Диаграмма классов в UML часто используется как **концептуальная модель данных** (аналог ER-диаграммы, но для ООП). Она показывает:

- Какие сущности (классы) есть в системе.
- Какими свойствами (атрибутами) они обладают.
- Какие операции (методы) они предоставляют.

- Как классы связаны друг с другом (ассоциации, агрегации, композиции) – это будет в ПР13.

На этапе анализа (ПР12) мы выделяем классы-сущности – те, которые отражают реальные объекты предметной области (пользователь, заказ, продукт, счёт и т.д.).

8. Как выделять классы, атрибуты и методы из текста

Классы – обычно существительные, которые обозначают важные понятия предметной области (не второстепенные).

Пример: в системе учёта задач: Задача, Пользователь, Проект.

Атрибуты – существительные, описывающие свойства классов.

Пример: для класса Задача – название, датаСоздания, статус.

Методы – глаголы или действия, которые нужны для манипуляции атрибутами или бизнес-логики.

Пример: создатьЗадачу(), изменитьСтатус(), рассчитатьДедлайн().

Правило: На этапе анализа (без учёта реализации) не нужно перечислять все методы доступа (геттеры/сеттеры) – достаточно бизнес-методов.

Практическая часть

Выполняется **индивидуально**.

Задание 1. Различение класса и объекта

Для каждого из следующих понятий определите, является ли оно **классом** (К) или **объектом** (О):

Понятие	Ответ (К/О)
Автомобиль	
Моя машина с госномером А123ВС	
Студент	
Студент Иванов, рост 180 см	
Счёт в банке	
Мой текущий баланс 5000 руб.	

Задание 2. Анализ текста: выделение классов

Прочитайте описание предметной области:

«В университете есть студенты и преподаватели. Каждый студент учится в одной группе. Группа имеет название и курс. Студенты сдают экзамены по дисциплинам. Каждый экзамен характеризуется оценкой (от 2 до 5) и датой проведения. Преподаватель ведёт несколько дисциплин. По каждой дисциплине есть список студентов, которые её изучают.»

Выпишите **все потенциальные классы** (существительные, которые важны для системы). Должно получиться 5–6 классов.

Задание 3. Определение атрибутов для классов

Для **трёх** классов из задания 2 (выберите любые, например: Студент, Группа, Экзамен) укажите:

- Не менее 2 атрибутов для каждого класса.
- Для каждого атрибута укажите его тип (String, int, Date, Boolean и т.п.).
- Примерную видимость (- private, + public – обычно все атрибуты private).

Используйте таблицу:

Класс	Атрибут	Тип	Видимость
Студент	имя	String	-
...	...	...	...

Задание 4. Определение методов

Для **двух** классов из задания 3 определите **минимум 2 метода** (операции), которые могут понадобиться в системе. Например:

- Для класса Студент: + сдатьЭкзамен(дисциплина, оценка) : Boolean, + перевестисьВГруппу(новаяГруппа) : void.
- Для класса Группа: + добавитьСтудента(студент) : void, + получитьСреднийБалл() : Real.

Методы записывайте в формате UML: [видимость] имя(параметры : тип) : возвращаемый_тип.

Задание 5. Изображение класса в нотации UML

Выберите **один класс** (любой из задания 2) и нарисуйте его в виде UML-прямоугольника с тремя секциями. Можно нарисовать от руки и сфотографировать, либо использовать текстовый псевдографический рисунок. Вставьте изображение в отчёт.

Задание 6. Преобразование в код – дополнительное

Напишите **псевдокод** или код на знакомом языке (Java/Python/C++) для класса, который вы изобразили в задании 5. Достаточно заголовка класса и объявления полей и методов (без реализации).

Пример на Python:

```
class Student:
    def __init__(self, name, age, student_id):
        self.__name = name
        self.__age = age
        self.__student_id = student_id

    def take_exam(self, subject, grade):
        pass

    def transfer_to_group(self, new_group):
        pass
```

Требования к отчёту

Отчёт сдаётся индивидуально в электронном виде (Markdown, PDF или DOCX).

Структура отчёта:

1. Титульный лист.
2. Цель и задачи (переписать из методички).
3. Задание 1 – таблица с ответами.
4. Задание 2 – список классов из текста.
5. Задание 3 – таблица атрибутов для 3 классов.
6. Задание 4 – таблица методов для 2 классов (или просто список).
7. Задание 5 – изображение класса в UML-нотации (рисунок или текстовая диаграмма).
8. Задание 6 (если выполнялось) – фрагмент кода.
9. Вывод – что узнали о классах, атрибутах, методах; какие трудности возникли при выделении.

Технические требования

1. Формат: .md, .pdf или .docx.
2. Шрифт: Times New Roman, 14 пт для основного текста, 12 пт для таблиц и подписей.
3. Межстрочный интервал: 1,5.
4. Поля: левое – 30 мм, правое – 15 мм, верх/низ – 20 мм.
5. Абзацный отступ (отступ первой строки) – 1,25 см.
6. Заголовки разделов – полужирные, выравнивание по ширине, абзацный отступ, нумерация (при необходимости).
7. Таблицы и рисунки (если есть) нумеруются и подписываются. Рисунок и подпись к рисунку выравниваются по центру. Подпись к рисунку располагается под ним, к таблице – над.
8. Диаграмма должна быть чёткой, текст на стрелках читаемым. Если использован ручной рисунок – качественное фото с хорошим освещением.

Контрольные вопросы

1. Что такое класс в UML? Чем класс отличается от объекта?
2. Назовите три секции прямоугольника, изображающего класс.
3. Как обозначается видимость private и public в UML?
4. Приведите пример атрибута с указанием имени, типа и видимости.
5. Что такое метод (операция) класса? Приведите пример.
6. Зачем нужна диаграмма классов при разработке ПО?
7. Как отличить класс от атрибута при чтении текста предметной области?
8. Может ли метод возвращать значение типа void? Что это значит?
9. Какие типы данных чаще всего используются в UML-моделях?
10. Почему на диаграмме классов важно указывать не все атрибуты и методы, а только значимые для анализа?

Практическая работа №12

Построение диаграммы классов (анализ)

Цель работы: научиться выделять классы-сущности предметной области на основе анализа текстовых артефактов проекта (технического задания, сценариев вариантов использования). Сформировать начальную диаграмму классов без связей как основу для последующего моделирования ассоциаций.

Задачи:

1. Освоить метод «поиска существительных» для выявления потенциальных классов.
2. Научиться отсеивать лишние кандидаты (атрибуты, действия, внешние системы) и оставлять только классы-сущности.
3. Определить атрибуты и методы для выделенных классов на основе описания предметной области.
4. Построить диаграмму классов в нотации UML (только классы, без связей) для своего проекта.

Теоретическая часть

1. Диаграмма классов на этапе анализа

На этапе **анализа** (не проектирования) диаграмма классов используется для понимания предметной области. Она показывает **классы-сущности** – понятия, которые важны для заказчика и пользователей, и которые система должна хранить и обрабатывать.

В отличие от диаграммы классов проектирования, анализ-модель:

- не содержит технических деталей (типы id, ссылки на БД);
- может не включать все методы (только ключевые бизнес-операции);
- фокусируется на **что**, а не **как**.

2. Метод «поиска существительных»

Это простейший способ выделения классов из текста:

1. Берём текстовое описание предметной области (ТЗ, сценарии use case, требования).
2. Выписываем все **существительные** (в единственном числе, нормализуем форму).
3. Анализируем каждое существительное:
    - **Класс** – если оно обозначает важную сущность, данные о которой нужно хранить и обрабатывать.
    - **Атрибут** – если оно описывает свойство другой сущности (например, «цвет» для «машины»).
    - **Действие / процесс** – если оно отглагольное (например, «запись» может быть процессом или сущностью – решайте по контексту).
    - **Внешняя система** – если оно относится к внешнему интерфейсу, может быть актёром, но не классом-сущностью.
4. Отбрасываем синонимы и слишком общие понятия («система», «объект», «данные»).

3. Критерии хорошего класса-сущности

- **Значимость** – без этого понятия система не имеет смысла.
- **Наличие данных** – у класса должны быть атрибуты (хотя бы 1-2).
- **Независимость** – класс не является просто свойством другого класса (иначе это атрибут).
- **Идентифицируемость** – экземпляры класса можно различить (уникальный идентификатор).

4. Атрибуты и методы на уровне анализа

Атрибуты выделяются из прилагательных, числительных и существительных, описывающих свойства класса. Примеры: имя, дата рождения, цвет, стоимость.

Методы – обычно глаголы, описывающие важные действия, которые система выполняет с классом (или класс выполняет сам). На этапе анализа достаточно перечислить ключевые бизнес-операции, а не все getter/setter.

5. Пример выделения классов из текста

Исходный фрагмент ТЗ для системы «Библиотека»:

Читатель может найти книгу по названию или автору. Книга находится в определенном разделе библиотеки. Читатель может взять книгу на абонемент на срок до 30 дней. За каждую книгу начисляется штраф за просрочку. Библиотекарь регистрирует выдачу и возврат книг.

Выписываем существительные: читатель, книга, название, автор, раздел, библиотека, абонемент, срок, штраф, просрочка, библиотекарь, выдача, возврат.

Анализ:

- **Читатель** – класс (пользователь системы, хранит ФИО, номер читательского).
- **Книга** – класс (название, автор, год, раздел).
- **Раздел** – класс (название, место).
- **Библиотекарь** – класс (сотрудник, логин, права).
- **Выдача** – класс (связующая сущность: читатель, книга, дата выдачи, срок).
- **Штраф** – класс (сумма, причина, оплачен ли).
- Название, автор, срок, просрочка – атрибуты (книги или выдачи).
- Библиотека (здание) – возможно не нужно, если не храним данные о здании.

6. Связь с ПР10 (сценарии use case)

Сценарии, детализированные в ПР10, являются отличным источником для выделения классов. Каждое существительное, которое фигурирует в

потоках событий как «объект», над которым совершаются действия, вероятно, является классом-сущностью.

7. Оформление диаграммы классов на этапе анализа

На данном этапе рисуют **только классы** (без связей) или с минимальными связями. Каждый класс изображается прямоугольником с тремя секциями: имя, атрибуты, методы. Инструменты: Draw.io, Lucidchart, Mermaid (можно использовать текстовую запись классов).

Практическая часть

Выполняется индивидуально. Используются материалы вашего проекта: ценарии use case из ПР10, требования из ПР6.

Задание 1. Подготовка текстов для анализа

Соберите в одном документе (можно скопировать в общий файл команды) следующие артефакты проекта:

- Раздел «Назначение разработки» из ТЗ (ПР5).
- Список функциональных требований (Must have из ПР6) – кратко.
- Один-два детальных сценария use case (из ПР10) – можно выдержки.

Задание 2. Поиск существительных

Каждый член команды (или совместно) выписывает **все существительные** из этих текстов. Игнорируйте предлоги, союзы, местоимения. Записывайте в столбик.

Совет: Используйте текстовый редактор с поиском или ментально проходите по каждому предложению.

После составления общего списка (10–20 слов) объедините синонимы («клиент», «пользователь», «читатель» – скорее всего один класс).

Задание 3. Фильтрация кандидатов

Для каждого существительного из списка определите его роль. Заполните таблицу:

Существительное	Роль (класс / атрибут / действие / внешнее / мусор)	Обоснование
...	...	...

Правила фильтрации:

- Если это свойство другого слова → атрибут.
- Если это глагол в существительной форме («выдача», «регистрация») – может быть классом, если важно хранить данные о событии, иначе действием.
- Если это внешняя система или человек за пределами системы → актёр (не класс-сущность) или исключить.
- Если это слишком общее («система», «информация», «процесс») → мусор.

Задание 4. Определение атрибутов для каждого класса

Для **каждого** выделенного класса (не менее 3, максимум 7) определите:

- 2–4 атрибута (с типами данных).
- Как минимум один ключевой атрибут, который может служить идентификатором.

Используйте таблицу:

Класс	Атрибут	Тип	Краткое описание
...	...	...	...

Задание 5. Определение ключевых методов

Для 2–3 самых важных классов укажите **бизнес-методы** (не геттеры/сеттеры). Например, для класса «Заказ»: «рассчитатьИтоговуюСтоимость()», «подтвердить()», «отменить()».

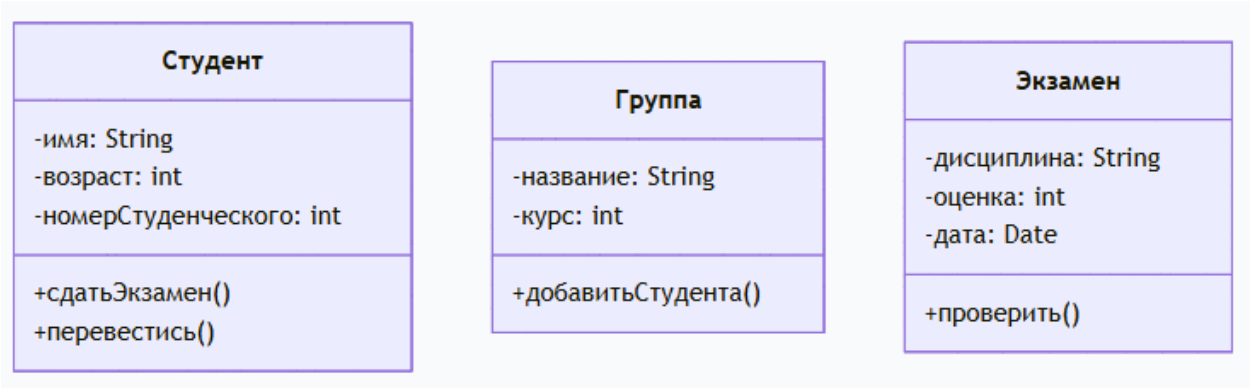
Задание 6. Построение диаграммы классов

Нарисуйте диаграмму, содержащую все выделенные классы. Пока **без связей**. Каждый класс изобразите прямоугольником с секциями. Можно использовать:

- **Draw.io** (вставить в отчёт скриншотом).
- **Mermaid** (синтаксис classDiagram, но без связей).
- **Ручной рисунок** (сфотографировать).

Пример Mermaid (только классы, без линий):

```
classDiagram
    class Студент {
        -имя: String
        -возраст: int
        -номерСтуденческого: int
        +сдатьЭкзамен()
        +перевестись()
    }
    class Группа {
        -название: String
        -курс: int
        +добавитьСтудента()
    }
    class Экзамен {
        -дисциплина: String
        -оценка: int
        -дата: Date
        +проверить()
    }
```



Требования к отчёту

Отчёт сдаётся индивидуально в электронном виде (Markdown, PDF или DOCX).

Структура отчёта:

10. Титульный лист.
11. Цель и задачи (переписать из методички).
12. Краткое описание проекта.
13. Задание 1 – список использованных текстов (ТЗ, сценарии) – можно просто указать ссылки на файлы.
14. Задание 2 – полный список существительных (сырые результаты).
15. Задание 3 – таблица фильтрации кандидатов.
16. Задание 4 – таблица атрибутов для каждого класса.
17. Задание 5 – список методов для 2–3 классов.
18. Задание 6 – диаграмма классов (изображение или код Mermaid).
19. Задание 7 – таблица трассировки.
20. Вывод – что удалось выделить, какие классы оказались неочевидными, как метод «поиска существительных» помог.

Технические требования

1. Формат: .md, .pdf или .docx.
2. Шрифт: Times New Roman, 14 пт для основного текста, 12 пт для таблиц и подписей.

3. Межстрочный интервал: 1,5.

4. Поля: левое – 30 мм, правое – 15 мм, верх/низ – 20 мм.

5. Абзацный отступ (отступ первой строки) – 1,25 см.

6. Заголовки разделов – полужирные, выравнивание по ширине, абзацный отступ, нумерация (при необходимости).

7. Таблицы и рисунки (если есть) нумеруются и подписываются. Рисунок и подпись к рисунку выравниваются по центру. Подпись к рисунку располагается под ним, к таблице – над.

8. Диаграмма должна быть чёткой, текст на стрелках читаемым. Если использован ручной рисунок – качественное фото с хорошим освещением.

Контрольные вопросы

1. Для чего нужна диаграмма классов на этапе анализа (а не проектирования)?

2. В чём суть метода «поиска существительных» для выделения классов?

3. Как отличить класс от атрибута при анализе текста?

4. Почему не все существительные становятся классами? Приведите пример «мусорного» существительного.

5. Что такое класс-сущность? Назовите три критерия.

6. Может ли действие (например, «регистрация») быть классом? Когда это оправдано?

7. Какие атрибуты вы добавили к своему главному классу? Почему именно они?

8. Нужно ли на диаграмме анализа показывать все методы (например, get/set)? Почему?

9. Как связаны сценарии use case и выделение классов?

10. Какой инструмент вы использовали для рисования диаграммы? Какие сложности возникли?

Практическая работа №13

Ассоциации и множественность

Цель работы: нучиться определять и изображать связи между классами на диаграмме классов UML: ассоциации, агрегацию, композицию. Освоить запись множественности (один-к-одному, один-ко-многим, многие-ко-многим) для отражения бизнес-правил предметной области.

Задачи:

1. Изучить виды связей между классами: простая ассоциация, агрегация, композиция, наследование (обобщение).
2. Освоить нотацию множественности (multiplicity) на концах ассоциаций.
3. Научиться различать агрегацию и композицию по степени «жёсткости» связи (жизненный цикл).
4. Проанализировать классы, выделенные в ПР12, и добавить между ними подходящие связи.
5. Построить полную диаграмму классов для своего проекта, включая ассоциации с множественностью.

Теоретическая часть

1. Зачем нужны связи между классами

Классы в реальной системе не существуют изолированно. Между ними есть отношения: студент учится в группе, заказ содержит товары, пользователь создаёт задачи. Диаграмма классов должна отражать эти связи, чтобы:

- разработчики понимали, как объекты ссылаются друг на друга;
- проектировщики БД могли строить внешние ключи;
- аналитики проверяли полноту модели.

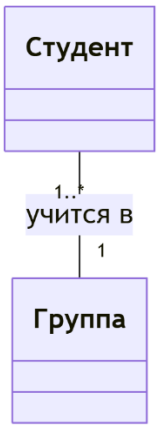
2. Простая ассоциация (Association)

Ассоциация – это самая общая связь, означающая, что объекты одного класса как-то связаны с объектами другого класса. Обозначается **сплошной линией** между классами. На линии может быть указано название ассоциации (глагол) и направление (треугольник).

Пример: «Студент учится в Группе».

classDiagram

Студент "1..*" -- "1" Группа : учится в



3. Множественность (Multiplicity)

Множественность показывает, сколько экземпляров одного класса может быть связано с одним экземпляром другого. Записывается у концов линии.

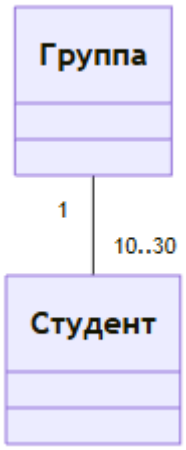
Стандартные обозначения:

Обозначение	Значение
1	Ровно один
0..1	Ноль или один
* или 0..*	Ноль или более
1..*	Один или более
m..n	От m до n включительно

Пример: У группы может быть от 10 до 30 студентов, а студент принадлежит ровно одной группе.

classDiagram

Группа "1" -- "10..30" Студент



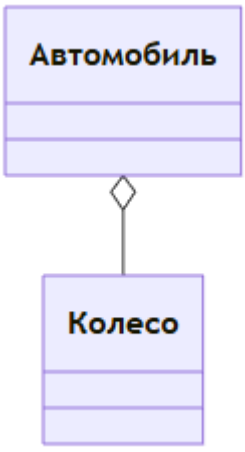
4. Агрегация (Aggregation)

Агрегация – это разновидность ассоциации, обозначающая отношение «часть–целое», но части могут существовать независимо от целого. Обозначается **пустым ромбом** со стороны «целого».

Пример: Автомобиль состоит из колёс. Колесо можно снять и переставить на другой автомобиль – оно живёт своей жизнью.

classDiagram

Автомобиль o-- Колесо



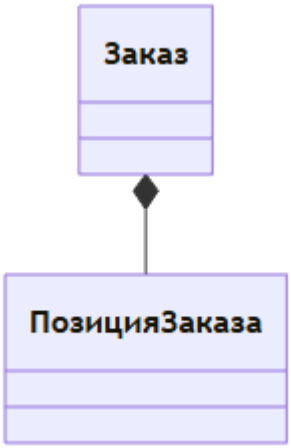
5. Композиция (Composition)

Композиция – более сильный вариант агрегации: части не могут существовать без целого. Если целое уничтожается, уничтожаются и все его части. Обозначается **закрашенным ромбом** со стороны «целого».

Пример: Заказ состоит из позиций заказа. Если удалить заказ, позиции заказа тоже удаляются (они не имеют смысла сами по себе).

classDiagram

Заказ *-- ПозицияЗаказа



Сравнение агрегации и композиции:

Характеристика	Агрегация	Композиция
Жизненный цикл частей	Независимый	Зависит от целого
Ромб	Пустой	Закрашенный
Пример	Команда – игроки (игрок может перейти)	Дом – комнаты (без дома комнаты нет)

6. Наследование (Generalization / Inheritance)

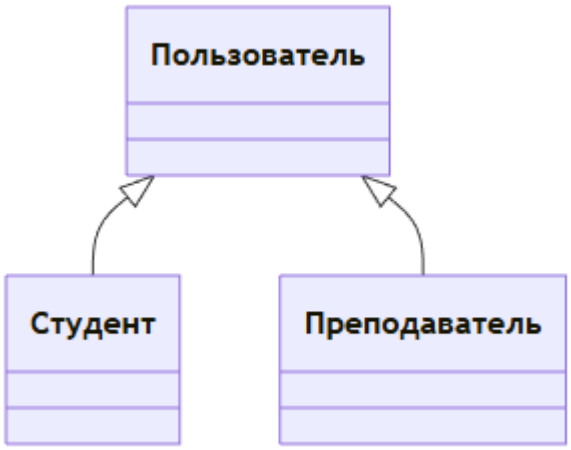
Наследование – отношение «является разновидностью» (is-a). Класс-потомок (подкласс) наследует все атрибуты и методы родительского класса. Обозначается **пустой треугольной стрелкой** от потомка к родителю.

Пример: «Студент» и «Преподаватель» – обобщаются в «Пользователь».

classDiagram

Пользователь <|-- Студент

Пользователь <|-- Преподаватель



7. Навигация (Navigation)

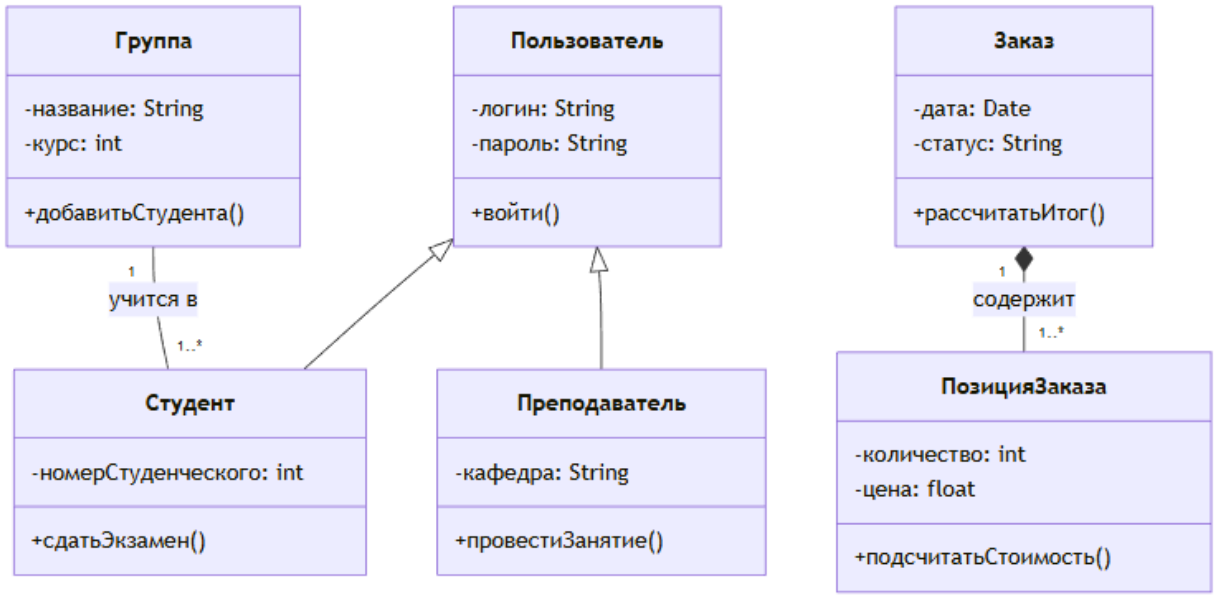
Иногда важно показать направление, в котором можно перемещаться по ассоциации. Стрелка на конце линии означает, что от исходного класса можно получить связанный объект. Если стрелки нет – ассоциация двунаправленная.

8. Полный пример диаграммы классов

```
classDiagram
    class Пользователь {
        -логин: String
        -пароль: String
        +войти()
    }
    class Студент {
        -номерСтуденческого: int
        +сдатьЭкзамен()
    }
    class Преподаватель {
        -кафедра: String
        +провестиЗанятие()
    }
    class Группа {
```

```
    -название: String
    -курс: int
    +добавитьСтудента()
}
class Заказ {
    -дата: Date
    -статус: String
    +рассчитатьИтог()
}
class ПозицияЗаказа {
    -количество: int
    -цена: float
    +подсчитатьСтоимость()
}

Пользователь <|-- Студент
Пользователь <|-- Преподаватель
Группа "1" -- "1..*" Студент : учится в
Заказ "1" *-- "1..*" ПозицияЗаказа : содержит
```



Практическая часть

Выполняется **индивидуально** на основе классов, выделенных в ПР12.

Задание 1. Анализ предметной области на предмет связей

Возьмите список классов из ПР12 (3–7 классов). Для каждой пары классов, которые могут быть связаны, ответьте на вопросы:

- Существует ли между ними отношение в реальном мире?
- Какая множественность? (сколько А связано с одним В и наоборот)
- Является ли связь «часть–целое»? Если да, то агрегация или композиция?

Заполните таблицу:

Класс А	Класс В	Тип связи (ассоциация/агрегация/композиция/обобщение)	Множественность (А→В)	Множественность (В→А)	Обоснование
...	...	...	...	...	...

Задание 2. Определение обобщений (10 минут)

Посмотрите на ваши классы. Есть ли среди них иерархия? Например:

- «Пользователь» ← «Студент», «Преподаватель».
- «Транспортное средство» ← «Автомобиль», «Мотоцикл».

Если есть – добавьте строки в таблицу задания 1 с типом «обобщение». Если нет – обоснуйте, почему обобщения не требуются (классы достаточно независимы).

Задание 3. Детализация ассоциаций

Для каждой ассоциации (не агрегации/композиции) добавьте:

- **Название ассоциации** (глагол, описывающий связь, например «работает над», «содержит»).
- **Направление** (если связь несимметрична).

Заполните дополнительную колонку в таблице из задания 1: «Имя ассоциации и направление».

Задание 4. Построение полной диаграммы классов

Используя инструмент (Draw.io, Mermaid, Lucidchart или ручной рисунок), нарисуйте диаграмму классов, включающую:

- Все классы из ПР12.
- Атрибуты и методы (из ПР12).
- Все связи (ассоциации, агрегации, композиции, обобщения) с указанием множественности и имён ассоциаций.

Требования:

- Минимум 3 связи между классами.
- Хотя бы одна связь с множественностью *, 1..* или подобной.
- Хотя бы одна агрегация или композиция (если уместно по предметной области).

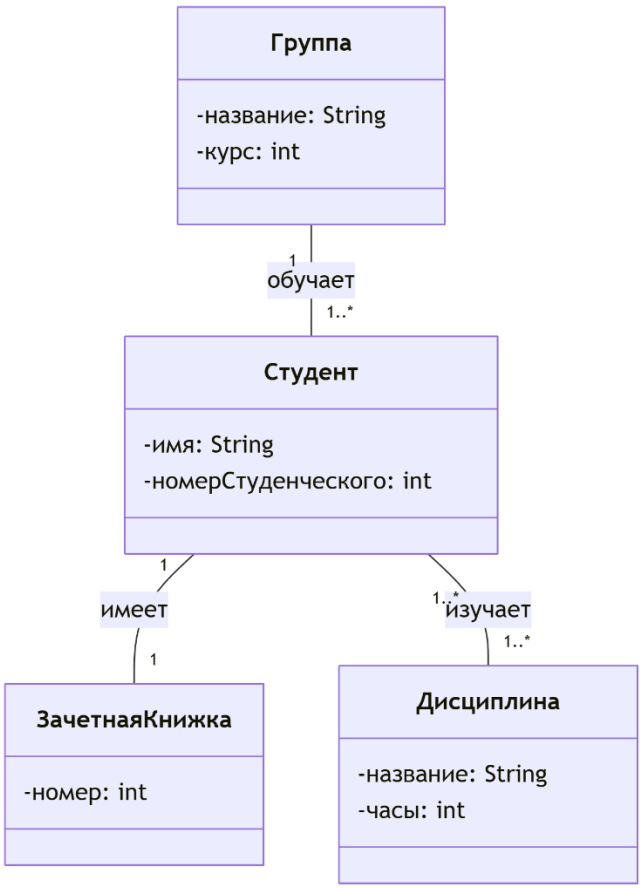
Пример Mermaid-кода (связи):

```
classDiagram
    class Студент {
        -имя: String
        -номерСтуденческого: int
    }
    class Группа {
        -название: String
        -курс: int
    }
    class ЗачетнаяКнижка {
        -номер: int
    }
    class Дисциплина {
        -название: String
        -часы: int
    }
```

Группа "1" -- "1..*" Студент : обучает

Студент "1" -- "1" ЗачетнаяКнижка : имеет

Студент "1..*" -- "1..*" Дисциплина : изучает



Задание 5. Проверка модели на соответствие бизнес-правилам

Выберите **два бизнес-правила** из вашего ТЗ (например, «Студент может записаться не более чем на 5 дисциплин в семестр») и проверьте, отражает ли ваша диаграмма классов эти ограничения через множественность. Если нет – скорректируйте множественность.

Задание 6. Сопоставление со сценариями use case

Для одной из ассоциаций найдите, в каком сценарии (из ПР10) эта связь используется. Например, ассоциация между «Заказ» и «ПозицияЗаказа» проявляется в сценарии «Оформить заказ», когда система добавляет позиции. Запишите это в отчёт.

Требования к отчёту

Отчёт сдаётся индивидуально в электронном виде (Markdown, PDF или DOCX).

Структура отчёта:

21. Титульный лист.

22. Цель и задачи (переписать из методички).

23. Список классов из ПР12 (повторить кратко).

24. Задание 1–3 – сводная таблица связей (тип, множественность, имя, обоснование).

25. Задание 4 – полная диаграмма классов (изображение или код Mermaid).

26. Задание 5 – проверка бизнес-правил (какие правила и как отражены).

27. Задание 6 – сопоставление со сценариями (таблица или текст).

28. Вывод – что нового узнали о связях, какие типы связей оказались самыми сложными для определения.

Технические требования

1. Формат: .md, .pdf или .docx.

2. Шрифт: Times New Roman, 14 пт для основного текста, 12 пт для таблиц и подписей.

3. Межстрочный интервал: 1,5.

4. Поля: левое – 30 мм, правое – 15 мм, верх/низ – 20 мм.

5. Абзацный отступ (отступ первой строки) – 1,25 см.

6. Заголовки разделов – полужирные, выравнивание по ширине, абзацный отступ, нумерация (при необходимости).

7. Таблицы и рисунки (если есть) нумеруются и подписываются. Рисунок и подпись к рисунку выравниваются по центру. Подпись к рисунку располагается под ним, к таблице – над.

8. Диаграмма должна быть чёткой, текст на стрелках читаемым. Если использован ручной рисунок – качественное фото с хорошим освещением.

Контрольные вопросы

1. Чем простая ассоциация отличается от агрегации и композиции?
2. Как обозначается композиция на диаграмме классов?
3. Что означает множественность 1..* на конце ассоциации?
4. Приведите пример отношения «один-ко-многим» из реальной жизни.
5. Когда следует использовать агрегацию, а когда композицию? Приведите по одному примеру.
6. Может ли существовать связь «многие-ко-многим»? Как она обозначается?
7. Что такое обобщение (наследование) в UML? Как оно рисуется?
8. Какой ромб (пустой или закрашенный) означает, что части не могут существовать без целого?
9. Зачем на диаграмме классов указывать множественность?
10. Может ли класс быть связан сам с собой (рефлексивная ассоциация)? Приведите пример.

Практическая работа №14

Документирование API / Интерфейсов

Цель: освоить базовые принципы документирования программных интерфейсов (API): научиться описывать входные и выходные данные для функций системы, понимать структуру API-запроса и ответа, а также познакомиться с распространёнными форматами описания (текстовое, табличное, OpenAPI).

Задачи:

6. Изучить понятие API (Application Programming Interface) и его роль в современной разработке ПО.

7. Познакомиться с основными элементами описания API: эндпоинт (URL), метод HTTP, параметры запроса, тело запроса, формат ответа, коды состояния.

8. Научиться выделять ключевые функции системы, требующие API-интерфейса (на основе функциональных требований и сценариев).

9. Составить документацию для 2–3 функций своего проекта в структурированном виде (таблица + примеры).

10. Оформить описание API в виде, понятном для разработчиков и тестировщиков.

Теоретическая часть

1. Что такое API

API (Application Programming Interface) – это набор правил и инструментов, с помощью которых одна программа может взаимодействовать с другой. В веб-разработке чаще всего говорят о **REST API**, где взаимодействие происходит через HTTP-запросы (GET, POST, PUT, DELETE) и данные передаются в формате JSON (или XML).

API нужен для:

- связи фронтенда и бэкенда (например, мобильное приложение общается с сервером);
- интеграции с внешними сервисами (платёжные системы, карты, соцсети);
- модульности внутри системы (микросервисы общаются через API).

Даже в рамках одного монолитного приложения полезно документировать **внутренние интерфейсы** – методы классов, функции модулей, их параметры и возвращаемые значения.

2. Что включает документация API

Хорошая документация API должна отвечать на вопросы:

- **Какой эндпоинт (URL)?** (например, POST /api/orders)
- **Какие параметры нужно передать?** (обязательные/необязательные, типы данных)
- **Какой формат данных?** (JSON, XML, form-data)
- **Что придёт в ответе?** (успех, ошибка, структура данных)
- **Какие коды состояния HTTP возможны?** (200 OK, 400 Bad Request, 404 Not Found)

Пример описания для функции «Создать задачу» в учебном проекте:

Поле	Значение
Название	Создать задачу
Метод	POST
URL	/api/tasks
Заголовки	Content-Type: application/json, Authorization: Bearer {token}
Тело запроса (JSON)	{"title": "string", "description": "string", "dueDate": "2025-05-01"}
Ответ при успехе (200)	{"id": 123, "status": "created"}
Ошибки	400 – неверный формат, 401 – не авторизован, 409 – задача уже существует

3. Основные HTTP-методы и их назначение

Метод	Назначение	Пример
GET	Получение данных (без изменения)	GET /api/users/5
POST	Создание нового ресурса	POST /api/users
PUT	Полная замена ресурса	PUT /api/users/5
PATCH	Частичное обновление	PATCH /api/users/5
DELETE	Удаление ресурса	DELETE /api/users/5

4. Форматы данных: JSON

JSON (JavaScript Object Notation) – текстовый формат обмена данными, основанный на синтаксисе JavaScript. Стал стандартом для REST API.

Пример:

```
{
  "user": {
    "id": 101,
    "name": "Иван Петров",
    "email": "ivan@example.com"
  },
  "role": "student"
}
```

5. Коды состояния HTTP (главные)

Код	Значение	Когда используется
200	OK	Успешный запрос, есть тело ответа
201	Created	Ресурс успешно создан (обычно после POST)
204	No Content	Успешно, но тело ответа пустое (например, DELETE)
400	Bad Request	Неверный запрос (ошибка валидации)
401	Unauthorized	Требуется аутентификация
403	Forbidden	Нет прав доступа
404	Not Found	Ресурс не найден
500	Internal Server Error	Ошибка на сервере

6. Инструменты для документирования API

- **OpenAPI (Swagger)** – спецификация для описания REST API в формате YAML/JSON. Позволяет генерировать интерактивную документацию.
- **Postman / Insomnia** – инструменты для тестирования API, позволяют экспортировать коллекции запросов как документацию.
- **Табличное описание** – простой вариант для учебных проектов (Word, Markdown, Google Docs).
- **Markdown + примеры** – удобно для небольших проектов.

7. Что описывать для внутренних функций (не HTTP)

Если в вашем проекте нет веб-сервера (например, десктопное приложение или библиотека), API – это набор публичных методов. Документация включает:

- сигнатуру метода (имя, параметры, тип возвращаемого значения);
- предусловия (какое состояние объектов требуется);
- постусловия (что изменится после вызова);
- возможные исключения/ошибки.

Пример для метода класса TaskManager**:**

```
def add_task(title: str, priority: int = 1) -> int:
    """
    Добавляет новую задачу в менеджер.

    Параметры:
        title (str): Название задачи (обязательно, не пустое).
        priority (int): Приоритет от 1 (низкий) до 5 (высокий). По умолчанию 1.

    Возвращает:
        int: Уникальный идентификатор созданной задачи.

    Исключения:
        ValueError: если title пустой или priority вне диапазона 1..5.
    """
```

Практическая часть

Выполняется **в командах** на основе выбранного проекта (мобильное приложение, веб-сервис, десктопное приложение).

Задание 1. Выбор функций для документирования

Из функциональных требований (Must have из ПР6) выберите **3 ключевые функции**, которые будут предоставлять внешний интерфейс (API). Например:

- Регистрация пользователя.
- Получение списка доступных записей (к врачу, к мастеру и т.п.).
- Создание новой записи.

Для десктопного приложения без сервера – выберите 3 публичных метода основного класса.

Задание 2. Определение формата API

Решите, будете ли вы описывать **HTTP REST API** (если проект предполагает клиент-серверную архитектуру) или **внутренние методы классов** (для локального приложения). Заполните таблицу:

Функция	Тип API (REST/внутренний)	Обоснование
...	...	...

Задание 3. Описание API для каждой функции

Для каждой выбранной функции составьте описание в виде таблицы (для REST) или спецификации метода (для внутреннего API).

Шаблон для REST API:

Функция 1: [Название]

- **Метод:** GET / POST / PUT / DELETE
- **URL:** /api/...
- **Параметры запроса (query):** (если есть)
- **Тело запроса (JSON):** пример
- **Заголовки:** (например, Authorization, Content-Type)

- **Ответ при успехе (200):** пример JSON
- **Возможные ошибки:** коды и описания

Шаблон для внутреннего метода (на примере Python, но можно псевдокод):

Функция 1: название_метода(параметры) -> возвращаемый_тип

- **Описание:** краткое объяснение.
- **Параметры:**
  - param1: тип – описание.
  - param2: тип = значение – описание (необязательный).
- **Возвращает:** что и в каком формате.
- **Исключения:** какие ошибки могут возникнуть.
- **Пример вызова:** код.

Задание 4. Создание примера запроса и ответа

Для **одной** из функций напишите **реальный пример** JSON-запроса и JSON-ответа (можно гипотетический, но правдоподобный). Пример оформите в виде блоков кода в Markdown.

Пример:

```
// Запрос POST /api/appointments
{
  "doctorId": 42,
  "patientId": 101,
  "dateTime": "2025-05-10T14:30:00Z"
}

// Успешный ответ (201 Created)
{
  "appointmentId": 1001,
  "status": "confirmed"
}

// Ошибка (400 Bad Request)
```

```
{
  "error": "This time slot is already taken"
}
```

Требования к отчёту

Отчёт сдаётся командой в электронном виде (Markdown, PDF или DOCX).

Структура отчёта:

29. Титульный лист.
30. Цель и задачи (переписать из методички).
31. Краткое описание архитектуры проекта (клиент-серверное или локальное приложение).
32. Задание 1 – список выбранных функций.
33. Задание 2 – таблица с типом API.
34. Задание 3 – подробное описание API для каждой функции (3 шт.).
35. Задание 4 – примеры запросов и ответов (JSON или код).
36. Задание 5 – скриншот доски Yougile с карточками API (или ссылка).
37. Задание 6 – результаты рецензии (замечания и предложения).
38. Вывод – что нового узнали об API, какие сложности возникли при описании.

Технические требования

9. Формат: .md, .pdf или .docx.
10. Шрифт: Times New Roman, 14 пт для основного текста, 12 пт для таблиц и подписей.
11. Межстрочный интервал: 1,5.
12. Поля: левое – 30 мм, правое – 15 мм, верх/низ – 20 мм.
13. Абзацный отступ (отступ первой строки) – 1,25 см.
14. Заголовки разделов – полужирные, выравнивание по ширине, абзацный отступ, нумерация (при необходимости).

15. Таблицы и рисунки (если есть) нумеруются и подписываются. Рисунок и подпись к рисунку выравниваются по центру. Подпись к рисунку располагается под ним, к таблице – над.

16. Диаграмма должна быть чёткой, текст на стрелках читаемым. Если использован ручной рисунок – качественное фото с хорошим освещением.

Контрольные вопросы

1. Что такое API? Приведите пример из реальной жизни (не IT).
2. Какие бывают виды API (по способу вызова)?
3. Перечислите основные HTTP-методы, используемые в REST API.
4. Какой код состояния HTTP означает успешное создание ресурса?
5. Что такое JSON? Почему он популярен для API?
6. Какие элементы должны быть в описании REST-эндпоинта?
7. Чем отличается документация для внутреннего метода класса от документации для веб-API?
8. Что такое Swagger / OpenAPI? Зачем они нужны?
9. Какой код ошибки возвращается, если пользователь не авторизован?
10. Почему важно документировать API даже для небольшого учебного проекта?

Практическая работа №15

Системы контроля версий (Git)

Цель: освоить базовые операции системы контроля версий Git для индивидуальной и коллективной разработки: инициализация репозитория, фиксация изменений (commit), синхронизация с удалённым репозиторием (push, pull), а также ознакомиться с принципами работы с ветками.

Задачи:

1. Понять назначение систем контроля версий (Git) в разработке ПО.
2. Научиться устанавливать Git и настраивать базовые параметры (имя пользователя, email).
3. Освоить основные команды: init, add, commit, status, log, clone, push, pull.
4. Познакомиться с концепцией ветвления: создание ветки, переключение между ветками, слияние (ознакомительно).
5. Зарегистрироваться на GitHub (или GitLab) и создать удалённый репозиторий.
6. Выполнить базовый сценарий коллективной работы: клонирование, добавление файлов, отправка изменений.

Теоретическая часть

1. Что такое система контроля версий (VCS)

Система контроля версий – это инструмент, который отслеживает изменения в файлах (обычно исходном коде) и позволяет возвращаться к любой предыдущей версии, работать над проектом в команде, не мешая друг другу, и фиксировать историю изменений.

Git – распределённая система контроля версий (каждый разработчик имеет полную копию истории). Самый популярный инструмент в мире разработки ПО.

2. Основные понятия Git

Термин	Описание
Репозиторий (repo)	Хранилище файлов и истории изменений. Может быть локальным (на компьютере) и удалённым (на сервере, например GitHub).
Коммит (commit)	Фиксация состояния файлов в репозитории в определённый момент. Каждый коммит имеет уникальный хеш, автора, дату и сообщение.
Ветка (branch)	Отдельная линия разработки. Позволяет вести параллельную работу над разными функциями. Основная ветка обычно называется main или master.
Индекс (staging area)	Промежуточная область, куда помещаются файлы перед коммитом (git add).
Удалённый репозиторий (remote)	Репозиторий на сервере (GitHub, GitLab, Bitbucket).

3. Жизненный цикл файла в Git

Untracked – файл не отслеживается Git (новый файл).

Staged – файл добавлен в индекс (готов к коммиту).

Committed – изменения сохранены в репозитории (коммит).

Modified – файл изменён после последнего коммита.

Рабочая директория → git add → Индекс (staging area) → git commit → Локальный репозиторий → git push → Удалённый репозиторий

4. Базовые команды Git

Команда	Назначение
git init	Создать новый локальный репозиторий в текущей папке
git clone <url>	Скопировать удалённый репозиторий на локальный компьютер
git status	Показать состояние файлов (изменённые, добавленные, неотслеживаемые)
git add <файл> или git add .	Добавить файл(ы) в индекс
git commit -m "сообщение"	Зафиксировать изменения из индекса в репозитории
git log	Показать историю коммитов
git push origin <ветка>	Отправить локальные коммиты в удалённый репозиторий

git pull origin <ветка>	Забрать изменения из удалённого репозитория и объединить с локальными
git branch	Показать список веток
git branch <имя>	Создать новую ветку
git checkout <имя>	Переключиться на другую ветку
git merge <ветка>	Слить указанную ветку с текущей

5. Работа с ветками (ознакомительно)

Ветка – это отдельный изолированный поток разработки. Например, вы можете создать ветку feature-login для разработки входа в систему, не затрагивая основную ветку main. Когда работа завершена, ветка сливается (merge) с main.

Простейший сценарий:

```
git branch feature-task      # создали ветку
git checkout feature-task    # переключились на неё
# ... делаем изменения, коммитим
git checkout main            # вернулись на основную ветку
git merge feature-task       # слили изменения
```

6. Удалённые репозитории (GitHub)

GitHub – это платформа для хостинга Git-репозиториев. Основные действия:

- Создать репозиторий на сайте (кнопка «New»).
- Связать локальный репозиторий с удалённым: git remote add origin https://github.com/ваш_логин/название.git
- Отправить изменения: git push -u origin main (первый раз с флагом -u).

7. .gitignore – исключение файлов

Файл .gitignore содержит шаблоны имён файлов и папок, которые Git не должен отслеживать (например, node_modules/, *.log, .idea/, __pycache__/).

Практическая часть

Выполняется **индивидуально** (каждый студент создаёт свой репозиторий).

Задание 1. Установка и настройка Git

1. Если Git не установлен – скачайте и установите с git-scm.com.
2. Откройте терминал (Git Bash на Windows, терминал на macOS/Linux).
3. Настройте имя пользователя и email (они будут привязаны к коммитам):
   git config --global user.name "Ваше Имя Фамилия"
   git config --global user.email "ваш_email@example.com"
4. Проверьте настройки: git config –list

Задание 2. Создание локального репозитория и первый коммит

1. Создайте папку git-practice на рабочем столе или в документах.
2. Перейдите в неё через терминал: cd путь/к/git-practice
3. Инициализируйте репозиторий: git init
4. Создайте файл README.md с текстом # Моя первая практика Git
5. Проверьте статус: git status (файл должен быть красным – неотслеживаемый)
6. Добавьте файл в индекс: git add README.md
7. Сделайте коммит: git commit -m "Добавлен README"
8. Посмотрите историю: git log (должен быть один коммит)

Задание 3. Создание удалённого репозитория на GitHub

1. Зарегистрируйтесь на github.com (если нет аккаунта).
2. Нажмите «New repository».
3. Назовите репозиторий git-practice-<ваша_фамилия> (например, git-practice-ivanov).
4. Оставьте публичным (Public) или приватным (Private) – не важно.
5. Не создавайте README, .gitignore и лицензию через сайт (уже есть локально).

6. Скопируйте URL репозитория (например, https://github.com/username/git-practice-ivanov.git).
7. Свяжите локальный репозиторий с удалённым:
   git remote add origin https://github.com/ваш_логин/git-practice-иванов.git
8. Отправьте изменения на GitHub:
   git push -u origin main
   (Если основная ветка называется master, используйте git branch -M main, чтобы переименовать)
9. Обновите страницу GitHub – файл README.md должен появиться.

Задание 4. Добавление файлов и второй коммит

1. Создайте файл about.txt с текстом: Это учебный проект по Git.
2. Создайте папку src и внутри – файл app.py (или main.js, program.cpp на выбор) с простым кодом, например:
   print("Hello from Git!")
3. Добавьте оба новых файла в индекс: git add . (добавляет всё)
4. Сделайте коммит: git commit -m "Добавлены about.txt и app.py"
5. Отправьте изменения на GitHub: git push
6. Проверьте в веб-интерфейсе GitHub – появились новые файлы.

Задание 5. Изменение файла и pull

1. Внесите изменения в README.md – добавьте новую строку: - Учусь работать с Git.
2. Сделайте git add README.md и git commit -m "Обновлён README"
3. Отправьте git push.
4. **Имитируем работу другого разработчика (или себя с другого компьютера):**
   Создайте другую папку git-practice-clone где-нибудь отдельно. Выполните git clone <URL вашего репозитория>. В клонированной копии проверьте, что все файлы и последний коммит на месте.

5. В основной рабочей папке (где вы работали) выполните git pull – он скажет, что уже актуально (если не было изменений извне).

Задание 6. Работа с ветками (ознакомительно)

1. Создайте новую ветку feature-hello:
   git branch feature-hello
2. Переключитесь на неё:
   git checkout feature-hello
   (Можно одной командой: git checkout -b feature-hello)
3. В файле app.py добавьте функцию say_hello() и вызов.
4. Сделайте коммит в этой ветке: git add app.py, git commit -m "Добавлена функция приветствия"
5. Отправьте ветку на GitHub: git push origin feature-hello
6. Вернитесь в ветку main: git checkout main
7. Слейте изменения из ветки feature-hello в main:
   git merge feature-hello
8. Отправьте обновлённый main: git push

Задание 7. Создание .gitignore

1. Создайте в корне репозитория файл .gitignore со следующим содержимым:
   # Логи
   *.log
   # Временные файлы
   *.tmp
   # Папка с зависимостями (если бы была)
   node_modules/
   __pycache__/
   # Файлы IDE
   .vscode/

.idea/

2. Добавьте .gitignore и сделайте коммит: git add .gitignore, git commit -m "Добавлен .gitignore"
3. Отправьте изменения: git push

Задание 8. Проверка знаний

В терминале выполните git log --oneline – вы должны увидеть цепочку коммитов (минимум 4 коммита). Сделайте скриншот результата.

Требования к отчёту

Отчёт сдаётся индивидуально в электронном виде – ссылка на GitHub репозиторий + текстовый отчёт.

Структура отчёта:

39. Титульный лист.

40. Цель и задачи (переписать из методички).

41. Ссылка на созданный репозиторий GitHub.

42. Скриншот терминала с результатом git log –oneline (задание 8).

43. Скриншот веб-интерфейса GitHub, показывающий файлы репозитория.

44. Ответы на контрольные вопросы (письменно, кратко).

45. Вывод – что освоили, какие команды были самыми трудными, что поняли о коллективной работе.

Технические требования

1. Формат: .md или .pdf.

2. Репозиторий должен быть **публичным.**

3. Шрифт: Times New Roman, 14 пт для основного текста, 12 пт для таблиц и подписей.

4. Межстрочный интервал: 1,5.

5. Поля: левое – 30 мм, правое – 15 мм, верх/низ – 20 мм.

6. Абзацный отступ (отступ первой строки) – 1,25 см.

7. Заголовки разделов – полужирные, выравнивание по ширине, абзацный отступ, нумерация (при необходимости).

8. Таблицы и рисунки (если есть) нумеруются и подписываются. Рисунок и подпись к рисунку выравниваются по центру. Подпись к рисунку располагается под ним, к таблице – над.

9. Диаграмма должна быть чёткой, текст на стрелках читаемым. Если использован ручной рисунок – качественное фото с хорошим освещением.

Контрольные вопросы

1. Что такое система контроля версий? Зачем она нужна?
2. Чем Git отличается от GitHub?
3. Что такое коммит? Из каких частей он состоит?
4. Для чего нужна команда git add перед git commit?
5. Как посмотреть текущее состояние файлов в репозитории?
6. Что делает команда git push? А git pull?
7. Как создать новую ветку и переключиться на неё?
8. Зачем нужен файл .gitignore? Приведите три примера, что в него стоит добавить.
9. Что произойдёт, если выполнить git merge без предварительного git pull (если удалённый репозиторий изменился)?
10. Как посмотреть историю коммитов в сокращённом виде (одна строка на коммит)?

Практическая работа №16

Платформы для совместной работы (GitLab/GitHub)

Цель: освоить инструменты командной работы на платформах GitLab/GitHub: ведение проектной документации с помощью Wiki и управление задачами через Issues. Научиться оформлять Wiki проекта в формате Markdown и ставить задачи через Issue с использованием шаблонов.

Задачи:

1. Изучить назначение и возможности Wiki как инструмента документирования проекта.
2. Освоить создание и структурирование Wiki-страниц с использованием Markdown.
3. Научиться настраивать боковую панель (sidebar) для навигации по Wiki.
4. Изучить функционал Issues для отслеживания задач, багов и фич.
5. Освоить создание Issue с использованием шаблонов (Markdown и YAML-формы).
6. Научиться применять метки (labels), назначение ответственных и вехи (milestones).
7. Понять, как связать Wiki и Issues в единый рабочий процесс.

Теоретическая часть

1. Wiki: Документирование проекта

Wiki — это встроенный инструмент GitLab и GitHub для создания и хранения документации, неразрывно связанной с репозиторием. Каждая Wiki представляет собой отдельный Git-репозиторий, что позволяет отслеживать изменения документации так же, как и изменения кода.

Что можно размещать в Wiki:

- Техническое задание и требования

- Архитектурные решения и диаграммы
- Инструкции по установке и настройке
- Правила оформления кода и ревью
- Сценарии вариантов использования
- Документацию по API
- Протоколы встреч и принятые решения

Преимущества использования Wiki:

- **Версионирование** — каждое изменение сохраняется как коммит с возможностью отката
- **Markdown-поддержка** — полная поддержка форматирования, таблиц, диаграмм и кода
- **Гибкая навигация** — возможность настройки боковой панели для структурирования документации
- **Доступ по ролям** — права доступа наследуются от основного репозитория
- **Связь с задачами** — можно ссылаться на Wiki-страницы из Issues и наоборот

Создание Wiki в GitHub:

1. Перейдите на главную страницу репозитория
2. Нажмите на вкладку **Wiki**
3. Нажмите **«Create the first page»**
4. Введите название и содержимое страницы
5. Нажмите **«Save Page»**

Создание Wiki в GitLab:

1. На боковой панели выберите **Plan → Wiki**
2. Нажмите **«Create your first page»**

3. Введите заголовок (на его основе автоматически генерируется путь)
4. Выберите формат (Markdown, RDoc, AsciiDoc)
5. Добавьте содержимое и нажмите **«Create page»**

2. Markdown для оформления Wiki

Базовый синтаксис Markdown:

Элемент	Синтаксис	Пример
Заголовки	# H1, ## H2, ### H3	## Архитектура системы
Жирный текст	**текст**	**Важно:** это обязательное требование
Курсив	*текст*	*Примечание:* см. раздел 3
Маркированный список	- пункт	- Аутентификация
Нумерованный список	1. пункт	1. Зарегистрироваться
Ссылка	[текст](URL)	[ТЗ](https://github.com/.../SRS.md)
Изображение	![alt](URL)	![Диаграмма](images/diagram.png)
Таблица	| A | B |	| Функция | Приоритет |
Блок кода	```язык | ```python print("Hello")	
Задачи	- [ ] задача	- [x] Реализовано

Рекомендации по структурированию Wiki:

1. **Главная страница (Home)** — содержит общее описание проекта, навигацию и ключевые ссылки.
2. **Техническая документация** — отдельный раздел для ТЗ, требований, архитектуры.
3. **Руководства** — инструкции по установке, настройке, развёртыванию.
4. **Командные процессы** — правила работы, определение готовности, регламенты.

Настройка боковой панели:

- **GitHub:** На странице Wiki нажмите **«Add custom sidebar»** и отредактируйте файл _Sidebar.md.

- **GitLab:** Боковая панель создаётся автоматически на основе иерархии страниц. Для кастомизации создайте страницу с именем _sidebar.

3. Issues: Управление задачами

Issue — это единица работы в GitLab/GitHub, используемая для отслеживания задач, сообщений об ошибках, запросов на новые функции и других рабочих элементов.

Основные компоненты Issue:

- **Заголовок** – краткое, но информативное описание.
- **Описание** – подробное изложение с использованием Markdown.
- **Метки (labels)** – категоризация (bug, enhancement, documentation, help wanted).
- **Ответственные (assignees)** – один или несколько участников, отвечающих за выполнение.
- **Вехи (milestones)** – группировка Issue по срокам или релизам.
- **Проект/Доска** – визуализация статуса выполнения (Kanban-доска).
- **Связи (linked items)** – связывание Issue с другими Issue, Pull Request'ами, Wiki-страницами.

Создание Issue в GitHub:

1. Перейдите на вкладку **Issues** репозитория
2. Нажмите **«New issue»**
3. Выберите шаблон (если настроен) или начните с пустого
4. Заполните заголовок и описание
5. Назначьте метки, ответственных, веху
6. Нажмите **«Submit new issue»**

Создание Issue в GitLab:

1. На боковой панели выберите **Plan → Issues**

2. Нажмите **«New issue»**
3. Заполните заголовок и описание
4. Назначьте метки, ответственных, веху, эпопею (epic)
5. Нажмите **«Create issue»**

4. Шаблоны Issues

Шаблоны Issues стандартизируют информацию, которую участники предоставляют при создании задач. Это повышает качество и полноту описаний, ускоряет процесс обработки.

GitHub: Markdown-шаблоны

Создайте файл .github/ISSUE_TEMPLATE/bug_report.md:

```
---
name: Сообщение об ошибке
about: Создайте отчёт, чтобы помочь нам улучшить продукт
title: "[BUG]: "
labels: bug, needs-triage
assignees: username

---

**Описание ошибки**
Чёткое и краткое описание проблемы.

**Воспроизведение**
Шаги для воспроизведения:
1. Перейти на '...'
2. Нажать на '....'
3. Увидеть ошибку

**Ожидаемое поведение**
```

Чёткое описание того, что должно было произойти.

Скриншоты
Если уместно, добавьте скриншоты.

Окружение:
- ОС: [например, Windows 11]
- Браузер: [например, Chrome 120]
- Версия: [например, 1.0.0]

GitHub: YAML-формы (рекомендуемый способ)

Более продвинутый вариант – создание .github/ISSUE_TEMPLATE/bug_form.yml:

```
name: Bug Report
description: Сообщить об ошибке
title: "[Bug]: "
labels: ["bug", "triage"]
assignees:
  - project-lead
body:
  - type: markdown
    attributes:
      value: |
        Спасибо за сообщение! Пожалуйста, заполните форму.
  - type: input
    id: contact
    attributes:
      label: Контактные данные
      description: Как с вами связаться для уточнения?
      placeholder: email@example.com
```

```
  validations:
    required: false
- type: textarea
  id: what-happened
  attributes:
    label: Что произошло?
    description: Опишите проблему и ожидаемое поведение
    placeholder: Расскажите, что пошло не так...
  validations:
    required: true
- type: dropdown
  id: version
  attributes:
    label: Версия
    options:
      - 1.0.0
      - 1.0.1
      - dev
  validations:
    required: true
- type: checkboxes
  id: terms
  attributes:
    label: Правила
    options:
      - label: Я подтверждаю, что проблема воспроизводится
        required: true
```

GitLab: Markdown-шаблоны

Создайте файл .gitlab/issue_templates/Bug.md:

/label ~bug ~"needs-triage"

/assign @project-manager

Описание ошибки

<!-- Краткое описание проблемы -->

Шаги воспроизведения

1. <!-- первый шаг -->
2. <!-- второй шаг -->

Ожидаемое поведение

<!-- Что должно было произойти -->

Фактическое поведение

<!-- Что произошло на самом деле -->

Окружение

- Версия ПО:
- Операционная система:
- Браузер (если применимо):

Дополнительная информация

<!-- Скриншоты, логи, другие детали -->

5. Интеграция Wiki и Issues

Мощная особенность GitLab и GitHub — возможность связывать Wiki-страницы и Issues в единое информационное пространство.

Ссылки из Issues на Wiki (GitHub/GitLab):

Документация

См. подробную спецификацию в Wiki: [[API-Design-Standards]]

Или с пользовательским текстом: [Стандарты проектирования API][API-Design-Standards]

Ссылки из Wiki на Issues:

Текущие задачи спринта

- Реализовать аутентификацию: #123
- Исправить баг с производительностью: #456
- Обновить документацию API: #789

В GitLab есть ещё более тесная интеграция: Wiki может отображать живые представления Issues с помощью GitLab Query Language (GLQL), превращая документацию в динамические дашборды.

6. Рабочий процесс: Wiki + Issues

1. Проект создан → Создаётся Wiki → Добавляется Home.md с общим описанием

↓

2. Планирование → В Issues создаются задачи → Каждая задача ссылается на Wiki-раздел с требованиями

↓

3. Разработка → Issue перемещается по доске (Todo → In Progress → Review → Done)

↓

4. Документация → Wiki обновляется параллельно с кодом → В Wiki указываются ссылки на закрытые Issues

↓

5. Итог → Полная прослеживаемость от документации до реализации

Практическая часть

Выполняется **в командах**. Используется проект, выбранный в предыдущих работах.

Задание 1. Создание Wiki и главной страницы (10 минут)

1. Перейдите в репозиторий вашего проекта на GitHub/GitLab.
2. Откройте раздел **Wiki**.
3. Создайте главную страницу **Home** со следующим содержанием (с использованием Markdown):
   - Заголовок с названием проекта (# Название проекта)
   - Краткое описание проекта (2–3 предложения)
   - Ссылки на ключевые разделы Wiki (техническое задание, архитектура, руководства)
   - Список участников команды с указанием ролей
4. Сохраните страницу.

Задание 2. Создание структуры Wiki (10 минут)

Создайте следующие страницы Wiki (каждую в отдельном файле):

Название страницы	Путь (slug)	Содержание
Техническое задание	technical-specification	Цели, функциональные требования, нефункциональные требования (из ПР5–ПР7)
Архитектура	architecture	Основные компоненты системы (можно схематично, из ПР12)
API	api-documentation	Описание основных эндпоинтов (из ПР14)

Командные процессы	team-processes	Правила работы с Git, регламент ревью, определение готовности задачи

Для каждой страницы:

- Используйте заголовки для структурирования (#, ##, ###).
- Используйте маркированные и нумерованные списки.
- Вставьте хотя бы одну таблицу (например, для функциональных требований или API).

Задание 3. Настройка боковой панели (5 минут)

Создайте или отредактируйте боковую панель для навигации по Wiki.

GitHub:

1. На странице Wiki нажмите **«Add custom sidebar»**
2. Добавьте ссылки на созданные страницы в формате Markdown:

   - [Главная](/Home)

   - [Техническое задание](/technical-specification)

   - [Архитектура](/architecture)

   - [API](/api-documentation)

   - [Командные процессы](/team-processes)

GitLab:

1. Создайте страницу с именем _sidebar
2. Добавьте аналогичное содержание

Задание 4. Создание Issues для задач проекта (15 минут)

Создайте **3–4 Issue** для вашего проекта на основе функциональных требований из ПР6. Используйте один из предложенных шаблонов.

Пример Issue «Функциональная задача»:

Название: Реализовать аутентификацию пользователя

Описание:

Пользователь должен иметь возможность зарегистрироваться и войти в систему.

Приоритет: Must have (MVP)

Приёмка:

- [ ] Реализована форма регистрации (имя, email, пароль)
- [ ] Реализована форма входа (email, пароль)
- [ ] Пароли хранятся в зашифрованном виде
- [ ] При успешном входе пользователь перенаправляется на главную страницу
- [ ] При неверных данных отображается сообщение об ошибке

Ссылка на Wiki: [[technical-specification#Функциональные требования]]

Ожидаемый срок: Конец 1-й итерации

Для каждого Issue:

- Выберите подходящие **метки (labels)**: enhancement, documentation, bug и т.п.
- Назначьте **ответственного** из вашей команды.
- При необходимости создайте **веху (milestone)** для группировки задач по спринтам.

Задание 5. Создание Issue-шаблона

Создайте в репозитории шаблон для Issue одного из типов:

- **Bug report** (сообщение об ошибке) — для GitHub или GitLab
- **Feature request** (запрос на новую функциональность)

Для GitHub:

1. Создайте папку .github/ISSUE_TEMPLATE/
2. Добавьте файл bug_report.md (Markdown-шаблон) или bug_form.yml (YAML-форма)
3. Закоммитьте и отправьте изменения в репозиторий

Для GitLab:

1. Создайте папку .gitlab/issue_templates/
2. Добавьте файл Bug.md
3. Закоммитьте и отправьте изменения

Задание 6. Связывание Wiki и Issues

1. В одном из созданных Issue добавьте ссылку на соответствующую Wiki-страницу (например, на раздел с требованиями).
2. На Wiki-странице «Текущие задачи» (создайте её, если ещё нет) добавьте список созданных Issues со ссылками на них.
3. Продемонстрируйте, как документация и задачи связаны в единую систему.

Задание 7. Использование Kanban-доски

GitHub:

1. Перейдите на вкладку **Projects**
2. Создайте новый проект с шаблоном **Kanban**
3. Добавьте созданные Issue в колонку **To do**
4. Переместите одно Issue в колонку **In progress**

GitLab:

1. Перейдите на вкладку **Issues → Boards**
2. Используйте стандартную доску или создайте новую
3. Перемещайте Issue между колонками

Задание 8. Итоговая проверка

Сделайте скриншоты:

1. Главной страницы Wiki
2. Боковой панели навигации
3. Списка созданных Issues
4. Kanban-доски с задачами
5. Содержимого одного Issue с заполненным шаблоном

Требования к отчёту

Отчёт сдаётся командой в электронном виде (Markdown, PDF или DOCX).

Структура отчёта:

46. Титульный лист.
47. Цель и задачи (переписать из методички).
48. Ссылка на репозиторий (GitHub/GitLab), содержащий Wiki и Issues.
49. Скриншоты:
    a. Главная страница Wiki
    b. Боковая панель навигации
    c. Список Issues (не менее 3)
    d. Один Issue с заполненным шаблоном
    e. Kanban-доска (Projects/Boards)
50. Список созданных страниц Wiki (названия и краткое описание содержания).
51. Список созданных Issues (названия, приоритет, ответственный).
52. Пример связи Wiki и Issues (какой Issue на какую Wiki-страницу ссылается).
53. Ответы на контрольные вопросы (письменно, кратко).
54. Вывод — что освоили, какие трудности возникли, как Wiki и Issues помогут в дальнейшей работе над проектом.

Технические требования

10. Формат: .md или .pdf.

11. Репозиторий должен быть **публичным.**

12. Шрифт: Times New Roman, 14 пт для основного текста, 12 пт для таблиц и подписей.

13. Межстрочный интервал: 1,5.

14. Поля: левое – 30 мм, правое – 15 мм, верх/низ – 20 мм.

15. Абзацный отступ (отступ первой строки) – 1,25 см.

16. Заголовки разделов – полужирные, выравнивание по ширине, абзацный отступ, нумерация (при необходимости).

17. Таблицы и рисунки (если есть) нумеруются и подписываются. Рисунок и подпись к рисунку выравниваются по центру. Подпись к рисунку располагается под ним, к таблице – над.

18. Диаграмма должна быть чёткой, текст на стрелках читаемым. Если использован ручной рисунок – качественное фото с хорошим освещением.

Контрольные вопросы

1. Что такое Wiki в GitLab/GitHub? Чем она отличается от обычного текстового файла в репозитории?

2. Какие преимущества даёт использование Wiki для документирования проекта?

3. Как создать новую страницу в Wiki и настроить боковую панель?

4. Какие элементы Markdown наиболее полезны для оформления технической документации?

5. Что такое Issue? Какие компоненты входят в структуру Issue?

6. Для чего нужны метки (labels) и вехи (milestones) в Issues?

7. В чём разница между Markdown-шаблонами и YAML-формами для Issue в GitHub?

8. Как связать Wiki-страницу с Issue и наоборот? Приведите пример синтаксиса.

9. Почему важно использовать шаблоны для Issues в командной разработке?

10. Как Kanban-доска (Projects/Boards) помогает отслеживать прогресс выполнения задач?

Практическая работа №17

Управление задачами (Time Management)

Цель: сформировать практические навыки управления временем и задачами в IT-проекте: освоить декомпозицию работ (WBS), научиться оценивать трудозатраты и визуализировать поток задач с помощью Kanban-досок.

Задачи:

1. Изучить методологию WBS (Work Breakdown Structure) – иерархическую декомпозицию работ.
2. Научиться разбивать функциональные требования и пользовательские истории на атомарные задачи.
3. Освоить методы оценки трудозатрат (экспертная оценка, аналогия, Planning Poker).
4. Познакомиться с Kanban-системой: колонки потока работ, лимиты незавершённых задач (WIP), Lead Time.
5. Применить Kanban-доску (Yougile, Trello или GitHub Projects) для визуализации задач и ограничения незавершённой работы.
6. Провести ретроспективу эффективности планирования.

Теоретическая часть

1. Управление временем в IT-проектах

Эффективное управление временем на уровне проекта включает:

- **Декомпозицию** – разбиение крупных задач на мелкие, измеримые.
- **Оценку трудоёмкости** – прогнозирование необходимых часов/дней.
- **Приоритезацию** (см. MoSCoW из ПР6).
- **Визуализацию потока** – Kanban-доска.
- **Контроль выполнения** – ежедневные стендапы, бёрндаун-диаграммы.

2. Декомпозиция задач (WBS)

WBS (Work Breakdown Structure) – иерархическое разбиение проекта на управляемые компоненты. Принципы:

- **Правило 100%** – сумма работ нижних уровней полностью покрывает работу родительского элемента.
- **Атомарность** – задача нижнего уровня длится не более 1 дня (или 8 часов) для одного исполнителя.
- **Измеримость** – результат задачи можно проверить (критерий приёмки).

Пример WBS для учебного проекта «Приложение для записи к врачу»:

1. Разработка MVP
   1.1. Регистрация и аутентификация
      1.1.1. Сверстать экран входа (фронт) – 4 ч
      1.1.2. Сверстать экран регистрации – 3 ч
      1.1.3. Реализовать API регистрации (бэк) – 4 ч
      1.1.4. Реализовать API входа – 2 ч
      1.1.5. Написать тесты на аутентификацию – 2 ч
   1.2. Просмотр расписания врачей
      1.2.1. Создать модель врача и расписания – 2 ч
      1.2.2. Реализовать получение списка врачей – 3 ч
      1.2.3. Реализовать получение свободных слотов – 3 ч
      1.2.4. Отобразить расписание на фронте – 4 ч
   1.3. Запись на приём
      1.3.1. Создать модель записи – 1 ч
      1.3.2. Реализовать API создания записи – 3 ч
      1.3.3. Реализовать проверку конфликтов – 2 ч
      1.3.4. Связать фронт с API – 3 ч

3. Оценка трудозатрат

Методы оценки:

Метод	Описание	Когда использовать
Экспертная оценка	Опытный разработчик называет часы	Небольшие задачи, знакомый стек
Аналогия	Сравнение с выполненной ранее задачей	Есть похожие задачи в истории
Параметрическая	На основе метрик (например, 1 час на форму)	Шаблонные задачи
Planning Poker (покер планирования)	Командная оценка через анонимные голосования	Сложные, неоднозначные задачи

Единицы оценки:

- Человеко-часы (ч) – реальное время работы одного человека.
- Story Points (SP) – относительная сложность (для Agile). Студентам на 1 курсе удобнее использовать часы.

Коэффициенты неопределённости:

- Для известных задач – +20% к оценке.
- Для новых задач – +50-100%.
- Учебные проекты: студенты часто недооценивают в 2-3 раза, поэтому лучше добавить коэффициент 1,5-2.

4. Kanban: принципы и элементы

Kanban – метод управления потоком работы, визуализирующий задачи на доске и ограничивающий количество незавершённых задач (WIP – Work In Progress).

Базовые колонки:

- **Backlog** – все задачи, которые только предстоит делать.
- **To Do** – задачи, взятые в работу на ближайшее время (обычно верхушка бэклога).
- **In Progress** – задачи, над которыми идёт работа прямо сейчас.

- **Review / Test** – задачи, переданные на проверку (тестирование, код-ревью).
- **Done** – завершённые задачи.

Лимиты WIP: ограничение на количество задач в колонке (например, не более 3 в «In Progress»). Это предотвращает многозадачность и ускоряет поток.

Метрики Kanban:

- **Lead Time** – время от появления задачи в бэклоге до её завершения.
- **Cycle Time** – время от начала работы над задачей до её завершения.
- **Throughput** – количество завершённых задач в единицу времени.

Инструменты для Kanban

Инструмент	Особенности
Yougile	Российский, бесплатный для малых команд, уже использовали в ПР2
Trello	Простой, бесплатный, с интеграциями
GitHub Projects	Встроен в GitHub, удобно связывать с Issues
GitLab Boards	Встроен в GitLab
Jira	Мощный, но избыточен для учебных проектов

Практическая часть

Задание 1. Построение WBS для MVP (20 минут)

Используя список функциональных требований **Must have** из ПР6 (MVP), постройте WBS (дерево работ) минимум для **трёх ключевых функций**.

Формат: иерархический список (можно в текстовом виде или в виде схемы). Каждая атомарная задача должна быть оценена в часах (экспертно, с учётом коэффициента 1,5-2 для учебного проекта).

Требования:

- Глубина декомпозиции – до задач длительностью 2–8 часов.
- Общее количество атомарных задач – не менее 10.
- Указать исполнителя (роль: фронтенд, бэкенд, тестировщик, аналитик) для каждой задачи.

Пример шапки таблицы WBS:

ID	Название задачи	Роль	Оценка (ч)	Зависимости
1.1.1	Сверстать экран входа	Фронтенд	4	–
1.1.2	Реализовать API входа	Бэкенд	3	–
1.1.3	Протестировать вход	Тестировщик	2	1.1.1, 1.1.2

Задание 2. Оценка суммарных трудозатрат и планирование итерации (10 минут)

1. Суммируйте все оценки часов из WBS. Получите **общую трудоёмкость MVP**.
2. Предположим, что команда может работать **10 часов в неделю на проект** (примерно 2–3 часа на человека в неделю). Рассчитайте, за сколько недель команда выполнит MVP.
3. Определите длину **итерации (спринта)** – например, 1 неделя или 2 недели. Разбейте задачи WBS на 2–3 итерации.

Заполните таблицу:

Итерация	Задачи (ID из WBS)	Суммарные часы	Ожидаемая дата завершения
Спринт 1	...	...	...
Спринт 2	...	...	...

Задание 3. Настройка Kanban-доски в Yougile / GitHub Projects (15 минут)

1. Откройте вашу доску в **Yougile** (создана в ПР2) или создайте новую **GitHub Project** (вкладка Projects).
2. Настройте колонки (столбцы) следующим образом:
    - **Бэклог** (все задачи из WBS)
    - **Готово к разработке** (отобранные на спринт, приоритетные)
    - **В работе** (с лимитом WIP = 2–3)
    - **На проверке** (тестирование / код-ревью)
    - **Готово**
3. Добавьте все задачи из WBS в колонку «Бэклог». Для каждой задачи укажите:
    - Исполнителя (роль или конкретного человека)
    - Оценку в часах
    - Дедлайн (по итерациям)
4. Установите **лимиты WIP** (ограничение на количество задач в колонке «В работе»). Например, не более 3 задач одновременно.
5. Переместите задачи первого спринта в колонку «Готово к разработке».

Задание 4. Имитация работы над задачами

Разыграйте короткий цикл работы (5–7 минут):

- Каждый член команды выбирает одну задачу из «Готово к разработке» и перемещает её в «В работе».
- Через 2 минуты имитируйте завершение – переместите задачу в «На проверку».
- Другой член команды (тестировщик) «проверяет» задачу и перемещает в «Готово» (или возвращает в «В работе» с комментарием).

Цель – почувствовать поток и ограничения WIP.

Задание 5. Расчёт метрик Kanban

На основе вашей доски (или гипотетически) рассчитайте:

- **Cycle Time** для одной задачи (время от входа в «В работе» до «Готово»).
- **Lead Time** для одной задачи (время от появления в бэклоге до «Готово»).
- **Throughput** – сколько задач ваша команда может завершить за неделю (на основе оценок из WBS).

Запишите результаты.

Задание 6. Ретроспектива планирования

Обсудите в команде:

- Какие задачи оказались недооценены? Переоценены?
- Какие зависимости между задачами не учли?
- Как лимит WIP повлиял на вашу «работу»?

Зафиксируйте 2–3 вывода, которые помогут лучше планировать в следующих итерациях.

Требования к отчёту

Отчёт сдаётся командой в электронном виде (Markdown, PDF или DOCX).

Структура отчёта:

1. Титульный лист.
2. Цель и задачи (переписать из методички).
3. WBS-таблица (задание 1) – минимум 10 атомарных задач с оценками.
4. План итераций (задание 2) – таблица разбиения задач по спринтам.
5. Скриншот Kanban-доски (Yougile или GitHub Projects) с колонками и задачами.
6. Результаты расчёта метрик (Cycle Time, Lead Time, Throughput) – задание 5.
7. Выводы ретроспективы (задание 6) – 2–3 пункта.
8. Эссе по УК-6 (задание 7).

9. Ответы на контрольные вопросы (кратко, письменно).

10. Ссылка на доску (Yougile / GitHub Project).

Технические требования

1. Формат: .md или .pdf.

2. Шрифт: Times New Roman, 14 пт для основного текста, 12 пт для таблиц и подписей.

3. Межстрочный интервал: 1,5.

4. Поля: левое – 30 мм, правое – 15 мм, верх/низ – 20 мм.

5. Абзацный отступ (отступ первой строки) – 1,25 см.

6. Заголовки разделов – полужирные, выравнивание по ширине, абзацный отступ, нумерация (при необходимости).

7. Таблицы и рисунки (если есть) нумеруются и подписываются. Рисунок и подпись к рисунку выравниваются по центру. Подпись к рисунку располагается под ним, к таблице – над.

8. Диаграмма должна быть чёткой, текст на стрелках читаемым. Если использован ручной рисунок – качественное фото с хорошим освещением.

Контрольные вопросы

1. Что такое WBS? Какое правило является основным при декомпозиции?

2. Назовите три метода оценки трудозатрат. Какой из них наиболее подходит для учебного проекта и почему?

3. Что такое лимит WIP? Зачем он нужен в Kanban?

4. В чём разница между Lead Time и Cycle Time?

5. Какой должна быть максимальная длительность атомарной задачи в WBS?

6. Какие колонки являются минимально необходимыми на Kanban-доске?

7. Почему студенты часто недооценивают трудоёмкость? Какой коэффициент вы использовали?

8. Что такое Throughput? Как его измерить?

9. Как WBS связана с MVP (Minimum Viable Product)?

10. Какие универсальные компетенции (УК-6) развивает умение планировать задачи и оценивать время?

Практическая работа №18

Прототипирование интерфейсов

Цель: освоить методы создания low-fidelity макетов (wireframes) для визуализации пользовательского интерфейса программной системы. Научиться преобразовывать функциональные требования и сценарии использования в наглядные схемы экранов, которые служат основой для дальнейшего дизайна и разработки.

Задачи:

1. Изучить понятие и назначение прототипирования в жизненном цикле ПО.
2. Познакомиться с уровнями детализации прототипов: low-fidelity, mid-fidelity, high-fidelity.
3. Освоить базовые принципы построения wireframes (структура, навигация, размещение элементов).
4. Научиться выделять ключевые экраны системы на основе пользовательских сценариев (use cases) и функциональных требований.
5. Создать набор low-fidelity макетов (не менее 5 экранов) для своего проекта с использованием бумаги / цифровых инструментов (Balsamiq, Figma, Draw.io, Miro).
6. Организовать простейшее тестирование прототипа с «пользователем» (одногруппником) и собрать обратную связь.

Теоретическая часть

1. Что такое прототипирование и зачем оно нужно

Прототипирование – это процесс быстрого создания упрощённой версии будущего продукта для проверки идей, сценариев и интерфейсных решений до начала разработки.

Цели прототипирования:

- Визуализировать функциональные требования (сделать их осязаемыми).
- Проверить удобство навигации и логику работы.
- Обнаружить нестыковки и пропущенные сценарии на ранних этапах.
- Согласовать видение между заказчиком, аналитиком и разработчиками.
- Снизить риски переделок на этапе кодирования.

2. Уровни детализации прототипов

Уровень	**Название**	**Особенности**	**Инструменты**	**Когда использовать**
Low-fidelity	Низкая детализация	Схематичные «скелеты» экранов, нет цветов, шрифтов, реальных изображений. Фокус на структуре и навигации.	Бумага, маркеры, Balsamiq Wireframes, Draw.io	Начало анализа, проверка концепции, мозговой штурм
Mid-fidelity	Средняя детализация	Более проработанные схемы с приблизительным расположением элементов, иногда с кликабельными связями.	Figma (без дизайн-системы), Axure, Moqups	Уточнение требований, демонстрация заказчику
High-fidelity	Высокая детализация	Пиксель-перфектные макеты, цвета, типографика, интерактивные переходы.	Figma, Sketch, Adobe XD	Передача в разработку, юзабилити-тестирование

В данной работе фокус на **low-fidelity** (wireframes).

3. Принципы построения wireframes

Wireframe — это схема экрана, показывающая:

- расположение ключевых блоков (шапка, меню, контентная область, подвал);
- иерархию элементов (что важнее — крупнее или выше);
- основные навигационные элементы (ссылки, кнопки, вкладки);
- поля ввода, кнопки действия, чекбоксы и т.д.

Правила хорошего wireframe:

- Использовать только оттенки серого (чтобы не отвлекать дизайном).
- Обозначать изображения крестиком или прямоугольником с надписью «[image]».
- Писать названия элементов простым текстом («Логин», «Пароль», «Войти»).
- Показывать связь между экранами (например, стрелками или номерами).
- Не углубляться в детали контента — достаточно заполнителей (lorem ipsum).

4. Как выделить ключевые экраны

Экраны (views/pages) определяются на основе:

- **Пользовательских сценариев (use cases)** из ПР9-10: каждый сценарий порождает один или несколько экранов.
- **Функциональных требований** (Must have): если есть функция «регистрация», значит нужен экран регистрации.
- **Навигационной структуры**: как пользователь перемещается между экранами (главный экран → список → детали → форма).

Типовой набор экранов для большинства проектов:

- Экран входа / регистрации
- Главный экран (дашборд / лента / список)
- Экран просмотра деталей (объекта, задачи, записи)
- Экран создания / редактирования (форма)

- Экран профиля / настроек
- Экран с результатом поиска (если применимо)

Для учебного проекта достаточно **4–6 экранов**, покрывающих основной пользовательский путь (happy path).

5. Инструменты для low-fidelity прототипирования

Инструмент	Плюсы	Минусы
Бумага + маркер	Быстро, доступно, не требует ПО	Неудобно пересылать, трудно вносить изменения
Balsamiq Wireframes	Специально под low-fidelity, «ручной» стиль	Платная (есть бесплатная пробная версия)
Draw.io / diagrams.net	Бесплатно, интеграция с Google Drive, есть библиотека UI-элементов	Не оптимизирован специально под прототипы
Figma (режим wireframe)	Бесплатно для студентов, мощный, можно потом улучшить	Избыточен для простых схем
Miro	Совместная работа в реальном времени, готовые шаблоны	Требуется регистрация

6. Связь с ранее созданными артефактами

- **Use case диаграммы (ПР9)** → определяют актёров и сценарии.
- **Детальные сценарии (ПР10)** → дают пошаговую логику работы экранов.
- **Диаграмма классов (ПР12-13)** → подсказывает, какие сущности и атрибуты нужно отображать.
- **API (ПР14)** → уточняет, какие данные приходят с сервера.

Перед началом рисования полезно выписать: какие экраны нужны, какие действия на них совершает пользователь, какие данные отображаются.

Практическая часть

Выполняется **в командах**. Используются материалы предыдущих работ (сценарии, требования).

Задание 1. Определение списка экранов (5 минут)

На основе функциональных требований (Must have из ПР6) и основных пользовательских сценариев (из ПР10) составьте список экранов для вашего MVP. Минимум **4 экрана**.

Формат таблицы:

№ экрана	Название экрана	Какой сценарий (use case) обслуживает	Ключевые элементы
1	Вход	Аутентификация	Поля логин/пароль, кнопка «Войти»
2	Регистрация	Создание аккаунта	Поля имени, email, пароля
3	Главный экран (список записей)	Просмотр расписания	Список врачей / слотов
4	Форма записи	Запись на приём	Выбор времени, подтверждение
...	...	...	...

Задание 2. Построение навигационной карты (5 минут)

Нарисуйте схему переходов между экранами (можно от руки, сфотографировать). Например:

[Вход] --(успех)--> [Главный экран] --(кнопка "Записаться")--> [Форма записи]

Покажите, как пользователь перемещается от одного экрана к другому. Укажите действия, которые вызывают переход (нажатие кнопки, выбор элемента списка и т.п.).

Задание 3. Создание wireframes (30 минут)

Выполните low-fidelity макеты всех экранов из задания 1. Используйте один из инструментов:

- **Draw.io** → библиотека «Mockup» или просто прямоугольники.

- **Бумага + ручка/маркер** → затем сфотографировать.
- **Balsamiq** (если доступен).

Требования к каждому wireframe:

- Показаны границы экрана (прямоугольник).
- Выделены основные зоны: заголовок, навигация (если есть), контент, кнопки действий.
- Использованы схематические обозначения:
  - Текстовое поле: [_____]
  - Кнопка: [Войти]
  - Список: элемент 1, элемент 2, ...
  - Изображение: прямоугольник с крестиком и надписью [image]
- Подписаны элементы (хотя бы кратко).
- Указан номер экрана (связь с заданием 1).

Совет: Начинайте с главного экрана, затем прорабатывайте экран входа/регистрации, затем формы ввода данных. Не тратьте время на выравнивание — low-fidelity допускает неровности.

Задание 4. Тестирование прототипа (10 минут)

1. Поменяйтесь с другой командой или пригласите одного одногруппника, не знакомого с вашим проектом.
2. Попросите «пользователя» выполнить **один сценарий** (например, «зарегистрироваться и записаться на приём») **без ваших подсказок**, только глядя на wireframes.
3. Наблюдайте: понятно ли, куда нажимать? Находит ли он нужные поля? Не путается ли в навигации?
4. Запишите **2–3 замечания** (что было непонятно, какие элементы отсутствуют, какие лишние).

Задание 5. Доработка по результатам тестирования (10 минут)

Внесите исправления в wireframes на основе обратной связи (хотя бы 2 изменения). Например:

- добавить кнопку «Назад»;
- переименовать непонятную кнопку;
- объединить два экрана или, наоборот, разделить.

Обновите макеты (можно добавить пометки или создать новую версию).

Задание 6. Связь с требованиями (5 минут)

Для каждого экрана укажите, какие **функциональные требования** (из ПР6) и какие **сценарии use case** (из ПР10) он реализует. Заполните таблицу:

Экран	Реализуемое функциональное требование (ID)	Соответствующий use case
...	...	...

Требования к отчёту

Отчёт сдаётся командой в электронном виде (Markdown, PDF или DOCX).

Структура отчёта:

1. Титульный лист.
2. Цель и задачи (переписать из методички).
3. Таблица экранов (задание 1) – минимум 4 строки.
4. Навигационная карта (задание 2) – схема переходов (скриншот или фото).
5. Wireframes (задание 3) – изображения всех экранов (можно вставить в документ по одному). Каждый макет подписан: «Экран 1 — Вход».
6. Результаты тестирования (задание 4) – список замечаний пользователя (2–3 пункта).

7. Исправленные макеты (задание 5) – либо обновлённые изображения, либо описание внесённых изменений.

8. Таблица связей с требованиями (задание 6).

9. Ответы на контрольные вопросы (письменно, кратко).

10. Вывод – что удалось визуализировать, какие проблемы выявило тестирование, как прототипы помогут в дальнейшей разработке.

Технические требования

1. Формат: .md или .pdf.

2. Шрифт: Times New Roman, 14 пт для основного текста, 12 пт для таблиц и подписей.

3. Межстрочный интервал: 1,5.

4. Поля: левое – 30 мм, правое – 15 мм, верх/низ – 20 мм.

5. Абзацный отступ (отступ первой строки) – 1,25 см.

6. Заголовки разделов – полужирные, выравнивание по ширине, абзацный отступ, нумерация (при необходимости).

7. Таблицы и рисунки (если есть) нумеруются и подписываются. Рисунок и подпись к рисунку выравниваются по центру. Подпись к рисунку располагается под ним, к таблице – над.

8. Диаграмма должна быть чёткой, текст на стрелках читаемым. Если использован ручной рисунок – качественное фото с хорошим освещением.

Контрольные вопросы

1. Что такое прототипирование? Какие задачи оно решает?

2. В чём разница между low-fidelity и high-fidelity прототипами?

3. Что такое wireframe? Какие элементы обязательно должны быть на нём?

4. Почему при low-fidelity прототипировании не используют цвета и реальные изображения?

5. Как выделить ключевые экраны для прототипа? На какие артефакты опираться?

6. Какие инструменты для прототипирования вы узнали? Какой использовали в работе?

7. Почему важно тестировать прототип на реальных пользователях (даже на одногруппниках)?

8. Что такое навигационная карта (user flow)? Зачем она нужна перед созданием wireframes?

9. Может ли один экран соответствовать нескольким use case? Приведите пример.

10. Как прототип связан с диаграммой классов? Подсказка: какие данные отображать на экране?

Практическая работа №19

Правовые аспекты в ИТ

Цель: изучить правовые аспекты разработки и использования программного обеспечения: виды лицензий (GPL, MIT, Apache и др.), их особенности и последствия выбора, а также виды ответственности (гражданская, административная, уголовная) за нарушение авторских прав в сфере ИТ. Сформировать понимание правовой грамотности, необходимой для профессиональной деятельности в области программной инженерии.

Задачи:

1. Изучить понятие лицензии на программное обеспечение и её роль в регулировании прав на ПО.
2. Познакомиться с классификацией лицензий (проприетарные, свободные, открытые) и основными типами open source лицензий (GPL, MIT, Apache, BSD).
3. Разобраться в ключевых различиях между пермиссивными (разрешительными) и копилефт-лицензиями.
4. Изучить нормативно-правовую базу защиты авторских прав на ПО в Российской Федерации (Гражданский кодекс РФ, КоАП РФ, УК РФ).
5. Проанализировать виды ответственности (гражданско-правовую, административную, уголовную) за нарушение авторских прав на ПО.
6. Научиться анализировать практические ситуации, связанные с выбором лицензии и нарушением авторских прав.
7. Составить памятку для разработчика по правовым аспектам использования open source ПО.

Теоретическая часть

1. Понятие лицензии на программное обеспечение

Лицензия на программное обеспечение – это юридический документ (договор), который регулирует условия использования, копирования, модификации и распространения программного продукта. Лицензия определяет права и обязанности как правообладателя (лицензиара), так и пользователя (лицензиата).

Программное обеспечение по типу лицензирования делится на две основные категории:

Тип ПО	Характеристика	Примеры
Проприетарное (закрытое)	Исходный код недоступен, использование ограничено условиями лицензионного соглашения (EULA), запрещены модификация и распространение	Microsoft Windows, Adobe Photoshop
Свободное / открытое (Open Source)	Исходный код доступен, разрешены изучение, модификация и распространение при соблюдении условий лицензии	Linux, Firefox, LibreOffice

Важно понимать, что **«открытый исходный код» не означает «бесплатно без условий»**. Любое ПО, даже open source, имеет правообладателя, и его использование должно соответствовать условиям лицензии.

2. Классификация open source лицензий

Все open source лицензии делятся на два основных класса: **пермиссивные (разрешительные)** и **копилефтные**.

2.1. Пермиссивные лицензии

Эти лицензии накладывают минимальные ограничения. Они позволяют использовать, модифицировать и распространять код в любых проектах,

включая закрытые коммерческие. Главное требование — сохранять уведомление об авторских правах и текст лицензии.

Лицензия	Ключевые особенности	Когда выбирать
MIT	Самая простая и «мягкая» лицензия. Разрешает любое использование при условии сохранения уведомления об авторских правах.	Для максимальной свободы использования кода. Популярна среди стартапов и библиотек.
Apache 2.0	Добавляет патентные положения – явное предоставление патентных прав и защита от патентных исков. Требует уведомления об изменениях.	Для проектов, где важна защита от патентных исков.
BSD	Аналогична MIT, но с некоторыми вариациями (2-Clause, 3-Clause).	Для проектов, где нужна совместимость с GPL (в некоторых версиях).

2.2. Копилефт-лицензии

Копилефт-лицензии (copyleft) гарантируют, что производные работы останутся открытыми. Если вы используете код под копилефт-лицензией, весь ваш проект должен распространяться под той же лицензией.

Лицензия	Ключевые особенности	Когда выбирать
GNU GPL	Самая известная копилефт-лицензия. Обладает «вирусным» эффектом: код под GPL требует открытости всего производного проекта. GPLv3 улучшила совместимость с Apache 2.0.	Для проектов, где важно сохранить код свободным и открытым навсегда.
LGPL	Менее строгая версия GPL, предназначенная для библиотек. Позволяет использовать библиотеку в проприетарных программах.	Для библиотек, которые могут использоваться в коммерческих проектах.

3. Сравнительная таблица: MIT, Apache, GPL

Критерий	MIT	Apache 2.0	GPL (GNU General Public License)
Тип	Пермиссивная	Пермиссивная с патентной оговоркой	Копилефт
Разрешает закрытое использование	Да	Да	Нет (весь проект должен быть открыт)
Требование к авторским уведомлениям	Сохранить в коде	Сохранить в коде + уведомление об изменениях	Сохранить в коде + указание изменений
Патентная защита	Нет	Есть	Есть в GPLv3
Совместимость с другими лицензиями	Высокая	Средняя (совместима с GPLv3)	Низкая (требует GPL для производных)
Популярность в коммерческих проектах	Высокая	Средняя	Низкая (коммерсанты избегают)

4. Авторские права на ПО в России

В Российской Федерации правовая охрана программ для ЭВМ регулируется **Частью четвёртой Гражданского кодекса РФ**.

Статья 1261 ГК РФ определяет: программы для ЭВМ — это объекты авторских прав, которые охраняются так же, как и литературные произведения. Программа может быть выражена на любом языке и в любой форме, включая исходный текст и объектный код.

Статья 1259 ГК РФ устанавливает, что для возникновения, осуществления и защиты авторских прав не требуется регистрация произведения — охрана возникает с момента создания ПО. Однако **возможна добровольная регистрация программы в Роспатенте**, которая даёт дополнительные преимущества: официальное доказательство авторства, упрощение судебной защиты, возможность получения компенсации до 5 млн рублей.

5. Ответственность за нарушение авторских прав на ПО

Нарушение авторских прав на программное обеспечение влечёт три вида ответственности: **гражданско-правовую**, **административную** и **уголовную**.

a. Гражданско-правовая ответственность (ст. 1252, 1301 ГК РФ)

Правообладатель может потребовать от нарушителя **компенсации** вместо возмещения убытков.

Размер компенсации определяется судом в одном из вариантов:

- от **10 000 до 5 000 000 рублей** (по усмотрению суда);
- в двукратном размере стоимости контрафактных экземпляров;
- в двукратном размере стоимости права использования произведения.

При этом правообладатель **освобождается от доказывания точного размера убытков**.

b. Административная ответственность (ст. 7.12 КоАП РФ)

Наступает за ввоз, продажу, сдачу в прокат или иное незаконное использование экземпляров произведений в целях извлечения дохода.

Размеры штрафов:

- для граждан: от **1 500 до 2 000 рублей**;
- для должностных лиц: от **10 000 до 20 000 рублей**;
- для юридических лиц: от **30 000 до 40 000 рублей**.

Санкция также предусматривает **конфискацию** контрафактных экземпляров, материалов и оборудования.

c. Уголовная ответственность (ст. 146 УК РФ)

Применяется при нарушении авторских прав в **крупном размере** (стоимость прав превышает **100 000 рублей**).

Санкции:

- штраф до 500 000 рублей;
- обязательные работы;
- лишение свободы на срок до **6 лет** (при отягчающих обстоятельствах).

6. Судебная практика: ответственность руководителя за пиратское ПО

В одном из знаковых дел Верховный Суд РФ разъяснил, что **простое наличие нелицензионного ПО на компьютерах предприятия не является безусловным доказательством вины руководителя**. Для привлечения к уголовной ответственности необходимо доказать, что руководитель давал указания на установку нелицензионных программ или был осведомлён об их использовании.

Это дело подчёркивает важность: для предотвращения правовых рисков на предприятии необходимо иметь чёткую политику использования лицензионного ПО и документировать контроль за её соблюдением.

7. Практические рекомендации для разработчика

1. **Перед использованием open source кода** — всегда проверяйте лицензию (файл LICENSE в репозитории).
2. **Не смешивайте несовместимые лицензии** — например, код под GPL нельзя комбинировать с закрытым кодом.
3. **Сохраняйте уведомления об авторских правах** — даже при использовании пермиссивных лицензий.
4. **Для своего проекта выбирайте лицензию осознанно** — учитывайте, как вы хотите, чтобы другие использовали ваш код.
5. **В коммерческих проектах избегайте GPL-зависимостей**, если вы не готовы открыть весь свой код.
6. **Ведите учёт используемых библиотек и их лицензий** — это поможет избежать нарушений и упростит аудит.

Практическая часть

Выполняется **в командах**.

Задание 1. Анализ лицензий

Изучите краткие описания лицензий ниже и заполните таблицу, определив тип каждой лицензии (пермиссивная / копилефт / проприетарная) и основные ограничения.

Лицензия	Краткое описание	Тип	Основные ограничения
MIT	Разрешает использовать, копировать, модифицировать, объединять, публиковать, распространять, сублицензировать и продавать копии ПО.		
GPL v3	Требует, чтобы любые производные работы распространялись под той же лицензией.		
Apache 2.0	Разрешает использование в закрытых продуктах, но требует указания изменений и предоставляет патентную лицензию.		
Проприетарная EULA	Закрытая лицензия, запрещающая модификацию, reverse engineering и распространение.		
BSD 3-Clause	Аналогична MIT, но запрещает использовать имя автора для продвижения без разрешения.		

Задание 2. Кейс «Выбор лицензии для проекта»

Ваша команда разрабатывает библиотеку для работы с базами данных. Ответьте на вопросы, заполнив таблицу для каждого сценария использования:

Сценарий 1: Вы хотите, чтобы библиотекой могли пользоваться любые разработчики, включая коммерческие компании, в своих закрытых проектах. Какую лицензию вы выберете? Почему?

Сценарий 2: Вы хотите гарантировать, что любые улучшения библиотеки будут оставаться открытыми. Какую лицензию вы выберете? Почему?

Сценарий 3: Вы опасаетесь патентных исков от крупных корпораций и хотите защитить пользователей вашей библиотеки. Какую лицензию вы выберете? Почему?

Сценарий	Выбранная лицензия	Обоснование
Сценарий 1		
Сценарий 2		
Сценарий 3		

Задание 3. Кейс «Нарушение авторских прав»

Проанализируйте ситуацию и ответьте на вопросы.

Ситуация: Студент разработал мобильное приложение и использовал в нём библиотеку под лицензией GPL, не открыв исходный код своего приложения. Приложение начало приносить доход от рекламы. Правообладатель библиотеки обратился в суд.

Вопросы для анализа:

1. Какие условия лицензии GPL были нарушены?
2. Какую ответственность (гражданскую, административную, уголовную) может понести студент?
3. Какие действия должен был предпринять студент, чтобы избежать нарушения?
4. Если бы студент использовал библиотеку под лицензией MIT, изменилась бы ситуация? Как?

Запишите ответы развёрнуто.

Требования к отчёту

Отчёт сдаётся командой в электронном виде (Markdown, PDF или DOCX).

Структура отчёта:

11. Титульный лист.

12. Цель и задачи (переписать из методички).

13. Задание 1 — заполненная таблица анализа лицензий.

14. Задание 2 — таблица с выбором лицензии для разных сценариев.

15. Задание 3 — развёрнутые ответы на вопросы кейса о нарушении авторских прав.

16. Ответы на контрольные вопросы (письменно, кратко).

17. Вывод — что нового узнали о правовых аспектах ИТ, какие лицензии показались наиболее интересными, почему правовая грамотность важна для программиста.

Технические требования

1. Формат: .md или .pdf.

2. Шрифт: Times New Roman, 14 пт для основного текста, 12 пт для таблиц и подписей.

3. Межстрочный интервал: 1,5.

4. Поля: левое – 30 мм, правое – 15 мм, верх/низ – 20 мм.

5. Абзацный отступ (отступ первой строки) – 1,25 см.

6. Заголовки разделов – полужирные, выравнивание по ширине, абзацный отступ, нумерация (при необходимости).

7. Таблицы и рисунки (если есть) нумеруются и подписываются. Рисунок и подпись к рисунку выравниваются по центру. Подпись к рисунку располагается под ним, к таблице – над.

8. Диаграмма должна быть чёткой, текст на стрелках читаемым. Если использован ручной рисунок – качественное фото с хорошим освещением.

Контрольные вопросы

1. Что такое лицензия на программное обеспечение? Какие основные права и обязанности она определяет?

2. В чём разница между проприетарным и открытым (open source) ПО?

3. Назовите два основных класса open source лицензий. Чем они отличаются?

4. Каковы ключевые особенности лицензии MIT? Почему она популярна в коммерческих проектах?

5. Каковы ключевые особенности лицензии GPL? Что означает «вирусный эффект» (copyleft)?

6. Чем Apache 2.0 отличается от MIT? Что даёт патентная оговорка?

7. Какие виды ответственности предусмотрены за нарушение авторских прав на ПО в России?

8. Какой закон является основным в сфере авторского права на ПО в РФ?

9. В каком размере может быть назначена компенсация за нарушение авторских прав по ГК РФ?

10. При каком условии наступает уголовная ответственность по ст. 146 УК РФ?

11. Какие меры предосторожности должен соблюдать разработчик при использовании open source библиотек?

12. Какую лицензию вы бы выбрали для своего учебного проекта? Почему?

Практическая работа №20

Экономика проекта

Цель: освоить базовые методы расчёта экономических показателей IT-проекта: оценку стоимости разработки программного продукта, расчёт затрат на оплату труда программиста, затрат машинного времени, а также формирование структуры себестоимости. Сформировать понимание экономической составляющей проектной деятельности.

Задачи:

1. Изучить основные статьи затрат при разработке ПО (трудовые, материальные, накладные).
2. Освоить методы расчёта трудоёмкости и стоимости разработки на основе оценок трудозатрат.
3. Научиться рассчитывать затраты машинного времени (аренда серверов, лицензии, электроэнергия).
4. Рассчитать фонд оплаты труда (ФОТ) программиста с учётом налогов и отчислений.
5. Оценить себестоимость разработки MVP своего проекта.
6. Составить калькуляцию и определить ориентировочную цену продажи продукта.

Теоретическая часть

1. Экономика IT-проекта: основные понятия

Экономика проекта – это раздел проектного менеджмента, оценивающий затраты, выгоды и риски, связанные с реализацией проекта. Для IT-проекта ключевыми экономическими показателями являются:

- **Себестоимость разработки** — сумма всех затрат на создание продукта.
- **Стоимость разработки** — цена, по которой разработчик продаёт продукт заказчику (включает себестоимость + прибыль).

- **Трудоёмкость** — количество человеко-часов (или человеко-дней), необходимых для выполнения работ.
- **Окупаемость** — время, за которое доходы от продукта покроют затраты на его разработку.

В учебном проекте мы оцениваем **себестоимость и стоимость разработки MVP**.

2. Структура затрат на разработку ПО

Все затраты можно разделить на три категории:

Категория	Состав	Примеры
Прямые трудовые затраты	Оплата труда разработчиков, аналитиков, тестировщиков, менеджера	Зарплата, премии, отчисления в фонды
Материальные и технические затраты	Оборудование, ПО, лицензии, серверная инфраструктура	Компьютеры, IDE, облачные сервера, электроэнергия
Накладные расходы	Затраты, не связанные напрямую с разработкой, но необходимые для функционирования компании	Аренда офиса, интернет, канцтовары, административные расходы

В учебном проекте основное внимание уделяется **трудовым затратам**, так как они составляют 70–90% себестоимости разработки ПО.

3. Оценка трудоёмкости и расчёт зарплаты

На основе WBS (декомпозиции работ из ПР17) мы имеем оценки в человеко-часах для каждой задачи. Для расчёта затрат на оплату труда (ФОТ) необходимо:

1. **Определить ставку оплаты труда** (например, для учебного проекта можно использовать условные значения: стажёр — 300 руб./ч, младший разработчик — 500 руб./ч, опытный разработчик — 1000 руб./ч).
2. **Рассчитать основную зарплату** = Сумма (оценка задачи × часовая ставка исполнителя).

3. **Добавить отчисления во внебюджетные фонды** (в РФ — 30,2% от ФОТ: 22% ПФР, 5,1% ФОМС, 2,9% ФСС, 0,2% травматизм — усреднённо).

Формула:

ФОТ = Основная зарплата

Общие затраты на оплату труда = ФОТ + (ФОТ × 0,302)

4. Затраты машинного времени

В затраты машинного времени включаются расходы на:

- **Амортизацию оборудования** (компьютеров, серверов) — стоимость оборудования, делённая на срок службы.
- **Облачные ресурсы** (аренда VPS, базы данных, CDN) — оплата по факту использования.
- **Лицензии на программное обеспечение** (IDE, операционные системы, инструменты).
- **Электроэнергию** (потребление оборудования в кВт·ч).

Упрощённый расчёт для учебного проекта:

- Стоимость машинного часа работы компьютера: 10–20 руб./ч (включая амортизацию и электричество).
- Затраты машинного времени = (Суммарные человеко-часы работы) × (Стоимость маш.-часа) × (Коэффициент одновременности работы, например 1 компьютер на человека).

Для серверной части: аренда облачного сервера (например, 1000 руб./мес). Если проект разрабатывается 2 месяца, то 2000 руб.

5. Накладные расходы

Обычно накладные расходы оцениваются как **процент от прямых затрат** (от 20% до 50% в зависимости от организации). Для учебного проекта можно принять **30%** от суммы прямых затрат (трудовые + машинное время).

6. Себестоимость и цена разработки

Себестоимость разработки = Прямые трудовые затраты + Материальные/технические затраты + Накладные расходы.

Цена разработки (для заказчика) = Себестоимость + Прибыль (обычно 20–50% от себестоимости).

Формулы:

Себестоимость = ФОТ_с_отчислениями + Затраты_маш_время + Накладные

Накладные = (ФОТ_с_отчислениями + Затраты_маш_время) × 0,30

Прибыль = Себестоимость × 0,30 (например)

Цена = Себестоимость + Прибыль

7. Пример расчёта для учебного проекта

Допустим, суммарная трудоёмкость MVP — 200 человеко-часов. Команда: 2 разработчика со ставкой 500 руб./ч, 1 тестировщик 400 руб./ч, 1 аналитик 600 руб./ч.

Роль	Часы	Ставка (руб./ч)	Сумма (руб.)
Разработчик 1	80	500	40 000
Разработчик 2	80	500	40 000
Тестировщик	25	400	10 000
Аналитик	15	600	9 000
Основная зарплата	200 ч		99 000 руб.

Отчисления (30,2%): 99 000 × 0,302 = 29 898 руб.

ФОТ с отчислениями = 128 898 руб.

Затраты машинного времени: 200 ч × 15 руб./ч = 3 000 руб.

Серверная аренда: 2 000 руб. (за 2 месяца).

Итого мат. затраты = 5 000 руб.

Накладные (30% от (128 898 + 5 000)) = 40 169 руб.

Себестоимость = 128 898 + 5 000 + 40 169 = **174 067 руб.**

Прибыль (30%): 174 067 × 0,30 = 52 220 руб.

Цена для заказчика = 174 067 + 52 220 = **226 287 руб.**

8. Экономические показатели для стартапа

Кроме себестоимости разработки, для коммерческого проекта важны:

- **Точка безубыточности** — количество проданных копий/подписок, при котором доходы сравняются с затратами.
- **ROI (Return on Investment)** — отношение чистой прибыли к инвестициям.
- **NPV (Net Present Value)** — чистая приведённая стоимость.

В учебном проекте достаточно ограничиться расчётом себестоимости и цены.

Практическая часть

Выполняется **в командах** на основе WBS из ПР17 (оценки трудозатрат).

Задание 1. Сбор исходных данных

Из ПР17 возьмите:

- Суммарную трудоёмкость MVP (в человеко-часах) и разбивку по ролям.
- Длительность проекта (количество недель/месяцев).

Дополнительно определите:

- Часовая ставка для каждой роли (согласно вашему региону или условно).

Используйте ориентиры:

  - Аналитик / PM – 600–800 руб./ч
  - Разработчик (Junior) – 400–500 руб./ч
  - Разработчик (Middle) – 800–1200 руб./ч
  - Тестировщик – 350–500 руб./ч
  - Дизайнер – 500–700 руб./ч

Для учебного проекта можно принять средние ставки.

Задание 2. Расчёт фонда оплаты труда

Заполните таблицу на основе вашей WBS:

Роль	Часы (из WBS)	Ставка (руб./ч)	Основная зарплата (руб.)
...	...	...	...
Итого	**∑ часов**		**∑ основная зарплата**

Рассчитайте:

- Отчисления во внебюджетные фонды (30,2% от основной зарплаты).
- **Общий ФОТ с отчислениями**.

Задание 3. Расчёт затрат машинного времени

Рассчитайте:

1. **Затраты на компьютеры/рабочие станции**:
    - Количество человек в команде = N.
    - Стоимость маш.-часа (примите 15 руб./ч, включая амортизацию и электричество).
    - Затраты = (Общее количество человеко-часов) × (стоимость маш.-часа) × (коэффициент 0,8–1,0, если не все работают одновременно).
2. **Затраты на серверную инфраструктуру** (если проект требует сервера):
    - Оцените стоимость аренды облачного сервера (например, 1000–3000 руб./мес).
    - Умножьте на длительность проекта (в месяцах).
3. **Прочие затраты** (лицензии на ПО, если используете платные инструменты) – по желанию.

Суммируйте все машинные затраты.

Задание 4. Расчёт накладных расходов

Накладные расходы примите как **30% от суммы прямых затрат** (ФОТ с отчислениями + затраты машинного времени).

Задание 5. Расчёт себестоимости и цены

Рассчитайте:

- **Себестоимость** = ФОТ с отчислениями + Затраты машинного времени + Накладные.
- **Прибыль** = Себестоимость × 0,30 (или другой процент, обоснуйте).
- **Цена для заказчика** = Себестоимость + Прибыль.

Результаты оформите в виде итоговой таблицы:

Статья затрат	Сумма (руб.)
Основная зарплата	
Отчисления (30,2%)	
Итого ФОТ с отчислениями	
Затраты машинного времени	
Накладные (30%)	
Себестоимость	
Прибыль (__%)	
Цена для заказчика	

Задание 6. Анализ точки безубыточности

Предположим, ваш продукт будет продаваться в виде подписки или копий. Придумайте модель монетизации (например, подписка 500 руб./мес или разовая продажа 3000 руб.). Рассчитайте:

- Сколько копий (или подписок на год) нужно продать, чтобы окупить разработку (точка безубыточности).
- При каком объёме продаж вы получите прибыль 100 000 руб.?

Формулы:

Точка безубыточности (шт) = Себестоимость / Цена_продажи_единицы

Прибыль = (Цена_единицы × Количество) - Себестоимость

Задание 7. Экономическое обоснование проекта

Напишите краткое экономическое обоснование (5–7 предложений) для вашего проекта, включив:

- Суммарные затраты на разработку.
- Потенциальный рынок (кто будет покупать).

- Прогноз продаж / окупаемости.
- Вывод о целесообразности разработки.

Требования к отчёту

Отчёт сдаётся командой в электронном виде (Markdown, PDF или DOCX).

Структура отчёта:

18. Титульный лист.

19. Цель и задачи (переписать из методички).

20. Исходные данные (суммарная трудоёмкость, ставки, длительность проекта).

21. Таблица расчёта ФОТ (задание 2).

22. Расчёт затрат машинного времени (задание 3).

23. Расчёт накладных расходов (задание 4).

24. Итоговая калькуляция (таблица себестоимости и цены, задание 5).

25. Расчёт точки безубыточности (задание 6) — таблица или расчёт.

26. Экономическое обоснование (задание 7).

27. Ответы на контрольные вопросы.

28. Вывод — что узнали об экономике проекта, какой вклад в стоимость вносит каждая статья, насколько реалистичны полученные цифры.

Технические требования

1. Формат: .md или .pdf.

2. Шрифт: Times New Roman, 14 пт для основного текста, 12 пт для таблиц и подписей.

3. Межстрочный интервал: 1,5.

4. Поля: левое – 30 мм, правое – 15 мм, верх/низ – 20 мм.

5. Абзацный отступ (отступ первой строки) – 1,25 см.

6. Заголовки разделов – полужирные, выравнивание по ширине, абзацный отступ, нумерация (при необходимости).

7. Таблицы и рисунки (если есть) нумеруются и подписываются. Рисунок и подпись к рисунку выравниваются по центру. Подпись к рисунку располагается под ним, к таблице – над.

8. Диаграмма должна быть чёткой, текст на стрелках читаемым. Если использован ручной рисунок – качественное фото с хорошим освещением.

Контрольные вопросы

1. Что входит в себестоимость разработки ПО? Перечислите основные статьи затрат.

2. Как рассчитать затраты на оплату труда с учётом налогов?

3. Что такое накладные расходы? Какой процент обычно закладывают в IT-проектах?

4. Как оценить затраты машинного времени? Приведите примеры.

5. Чем отличается себестоимость от цены разработки?

6. Что такое точка безубыточности? Как она рассчитывается?

7. Какие отчисления с заработной платы существуют в РФ? Назовите примерный суммарный процент.

8. Почему в IT-проектах трудоёмкость (человеко-часы) – ключевой фактор стоимости?

9. Как влияет выбор облачной инфраструктуры на себестоимость проекта?

10. Может ли проект иметь отрицательную прибыль? Когда это оправдано?

Практическая работа №21

Презентация проекта и переговоры

Цель: сформировать умения подготовки эффективной презентации проекта (питч-дек) и ведения переговоров с заказчиком, инвесторами или стейкхолдерами. Освоить структуру питч-дека, принципы убедительной коммуникации и правила публичного выступления.

Задачи:

1. Изучить структуру и содержание питч-дека для IT-проекта (презентации для инвесторов, заказчиков или защиты проекта).
2. Освоить правила визуального оформления слайдов (лаконичность, единый стиль, читаемость).
3. Познакомиться с техниками публичного выступления: удержание внимания, работа с голосом, тайминг, ответы на вопросы.
4. Научиться адаптировать презентацию под разные аудитории (технические специалисты, менеджмент, инвесторы).
5. Подготовить и провести мини-питч (5–7 минут) своего проекта перед группой или на камеру.
6. Освоить базовые навыки ведения переговоров: подготовка аргументов, работа с возражениями, достижение договорённостей.

Теоретическая часть

1. Питч-дек: что это и зачем нужен

Питч-дек (pitch deck) – это краткая презентация проекта (обычно 10–15 слайдов), предназначенная для привлечения внимания инвесторов, заказчиков или руководства. Главная цель – за короткое время (3–10 минут) донести ключевую ценность проекта, его уникальность и выгоды для аудитории.

В учебном проекте питч-дек используется для защиты курсовой работы или демонстрации результатов перед комиссией.

2. Структура классического питч-дека (12 слайдов)

№ слайда	Название	Содержание
1	Титульный слайд	Название проекта, логотип, авторы (команда), дата
2	Проблема	Какая проблема существует у целевой аудитории? Почему это важно? (цифры, факты)
3	Решение	Ваш продукт / сервис — как он решает проблему? Кратко и понятно
4	Целевая аудитория	Кто пользователи? Сегментация, портрет клиента (возраст, потребности)
5	Уникальное ценностное предложение (UVP)	Чем вы отличаетесь от конкурентов? (1–3 ключевых отличия)
6	Как это работает	Схема работы продукта (можно показать скриншоты прототипа или диаграмму)
7	Рынок и конкуренты	Объём рынка (TAM, SAM, SOM), основные конкуренты и ваши преимущества
8	Бизнес-модель	Как вы зарабатываете? (подписка, лицензии, freemium, реклама и т.д.)
9	Технология и архитектура	Кратко: используемые технологии, стек, почему такой выбор (если аудитория техническая)
10	План развития	Дорожная карта (MVP → версия 1.0 → новые функции)
11	Команда	Кто делает проект? Опыт, роли, почему эта команда справится
12	Призыв к действию (CTA)	Что вы хотите от аудитории? (финансирование, пилотный проект, обратная связь)

Для учебной защиты можно сократить до 8–10 слайдов, убрав рыночные и бизнес-блоки, если проект некоммерческий.

3. Правила оформления презентации

Правило	Пояснение
Минимум текста	Не более 5–7 строк на слайде, ключевые слова и тезисы. Подробности — в устной речи.
Крупный шрифт	Заголовки — 32–40 pt, основной текст — 24–28 pt.
Один слайд – одна мысль	Не смешивайте проблему и решение на одном слайде.
Визуализация	Используйте схемы, иконки, диаграммы, скриншоты. Избегайте сложных таблиц.
Единый стиль	Одинаковые шрифты, цветовая гамма (2–3 цвета), расположение логотипа.
Контрастность	Тёмный текст на светлом фоне (или наоборот, но без пестроты).
Номера слайдов	Облегчают навигацию и вопросы («вернитесь на слайд 5»).

4. Правила публичного выступления (для питча)

Подготовка:

- Выучите первые 2–3 предложения наизусть (чтобы уверенно начать).
- Отрепетируйте тайминг (обычно 5–7 минут на 10 слайдов).
- Продумайте возможные вопросы и заготовьте ответы.

Во время выступления:

- **Контакт с аудиторией** — смотрите на людей, не читайте со слайдов.
- **Голос** — говорите громко, чётко, с паузами. Избегайте монотонности.
- **Язык тела** — стойте прямо, не скрещивайте руки, жестикулируйте умеренно.
- **Не извиняйтесь** («я плохо подготовился», «извините за качество»).
- **Управляйте временем** — если видите, что не успеваете, сократите подробности, но не тараторьте.

Ответы на вопросы:

- Выслушайте вопрос до конца, не перебивайте.
- Если не знаете ответа — честно скажите: «Я не готов ответить сейчас, но уточню и вернусь».

- На сложный вопрос можно ответить кратко и предложить обсудить детали после выступления.

5. Адаптация презентации под аудиторию

Тип аудитории	Акцент в презентации	Что опустить
Инвесторы	Бизнес-модель, рынок, потенциальная прибыль, уникальность	Технические детали, архитектура
Заказчик (бизнес)	Решение его проблем, экономия времени/денег, сроки, стоимость	Внутреннее устройство, выбор языка
Технические специалисты	Архитектура, технологии, масштабируемость, API	Подробности о бизнес-плане
Учёные / комиссия	Научная новизна, методология, корректность расчётов	Маркетинговые уловки

6. Навыки переговоров при защите проекта

Переговоры — это двусторонняя коммуникация, направленная на достижение взаимовыгодного соглашения. При защите проекта элементы переговоров возникают при обсуждении требований, сроков, бюджета или при ответах на возражения.

Этапы переговоров (упрощённо):

1. **Подготовка** — определить свою цель, минимально приемлемый результат (МТО), изучить интересы другой стороны.
2. **Открытие** — приветствие, обозначение повестки, установление контакта.
3. **Обсуждение** — аргументация своей позиции, активное слушание, вопросы.
4. **Работа с возражениями** — не отрицать, а признать и перевести в конструктив («Да, это сложно, но мы предлагаем…»).
5. **Закрытие** — фиксация договорённостей, благодарность.

Приёмы работы с возражениями (для защиты проекта):

- **Присоединение** — «Я понимаю ваше беспокойство…», «Это важный вопрос…»
- **Аргументация** — факты, цифры, ссылки на успешные аналоги.
- **Переформулирование** — «Если я правильно понял, вы хотите сказать, что…»
- **Отложенный ответ** — «Давайте я отвечу на этот вопрос после презентации, когда покажу все возможности».

Практическая часть

Выполняется **в командах**. Каждая команда готовит питч-дек своего проекта (на основе всех предыдущих работ) и проводит мини-презентацию.

Задание 1. Сбор материалов для питч-дека

Используя артефакты вашего проекта из предыдущих работ (ПР5–ПР18), соберите информацию для слайдов:

Слайд	Источник информации
Проблема	ТЗ, раздел «Назначение разработки»
Решение	Функциональные требования (Must have)
Целевая аудитория	Сценарии use case (актёры)
Уникальность	Анализ конкурентов (ПР4, если делали)
Как это работает	Диаграмма use case (ПР9) + wireframes (ПР18)
Технологии	Требования к программной совместимости (ПР7)
План развития	WBS и итерации (ПР17)
Команда	Распределение ролей (ПР2)

Задание 2. Создание питч-дека

Создайте презентацию из **8–10 слайдов** в любом инструменте (Google Slides, PowerPoint, Canva, Figma). Используйте правила оформления (минимум текста, крупные заголовки, визуализация). **Каждый слайд должен содержать не более 5–7 строк текста (или 3–4 ключевых пункта).**

Обязательные слайды:

1. Титул (название, команда)
2. Проблема

3. Решение (ваш продукт)
4. Уникальность / преимущества
5. Демонстрация (скриншоты wireframes или прототипа)
6. Технологии и архитектура (кратко)
7. План развития (дорожная карта)
8. Команда
9. Призыв к действию / контакты

Для некоммерческого учебного проекта слайды о рынке и бизнес-модели можно опустить или объединить.

Задание 3. Подготовка устного выступления

Напишите **сценарий выступления** на 5–7 минут. Распределите, кто из команды говорит какой слайд. Один человек не должен говорить более 2 минут подряд. Потренируйтесь внутри команды.

Структура устной части:

- Приветствие (15 сек)
- Проблема и решение (1,5 мин)
- Уникальность и демонстрация (2 мин)
- Технологии и план (1,5 мин)
- Команда и призыв (1 мин)

Задание 4. Проведение мини-питча

Каждая команда (или по одному представителю от команды) выступает перед группой (или записывает видео на телефон). Время выступления — **строго 5–7 минут**. Преподаватель засекает время. Остальные студенты играют роль заказчиков / инвесторов.

Задание 5. Вопросы и работа с возражениями (5 минут на команду)

После выступления аудитория задаёт 1–2 вопроса (в том числе провокационных). Команда должна ответить, используя техники работы с возражениями. Примеры вопросов:

- «Почему вы выбрали эту технологию, а не более современную?»
- «Что будет, если ваш основной разработчик уволится?»
- «Почему пользователи должны платить, если есть бесплатные аналоги?»
- «Вы учли юридические риски (персональные данные)?»

Требования к отчёту

Отчёт сдаётся командой в электронном виде (Markdown, PDF или DOCX).

Структура отчёта:

1. Титульный лист.
2. Цель и задачи (переписать из методички).
3. Ссылка на презентацию (Google Slides, PDF, PowerPoint файл).
4. Скриншоты всех слайдов (можно по 2–3 на странице) с краткими комментариями.
5. Сценарий выступления (что говорит каждый член команды, тайминг).
6. Чек-лист самооценки (заполненный каждым участником или общий).
7. Отзывы от других команд (2–3 отзыва).
8. Ответы на контрольные вопросы.
9. Вывод – что удалось при подготовке презентации, какие сложности возникли, какие приёмы выступлений оказались полезными.

Технические требования

1. Формат: Презентация в формате .pptx, .pdf или ссылка на Google Slides (с открытым доступом).
2. Отчёт – .pdf или .md.

Контрольные вопросы

11. Что такое питч-дек? Сколько слайдов рекомендуется для классической презентации проектов?
12. Какие три слайда являются обязательными в любом питч-деке?
13. Назовите 5 правил оформления презентации.
14. Как адаптировать презентацию под техническую аудиторию? А под бизнес-аудиторию?
15. Что делать, если вы не знаете ответ на вопрос во время презентации?
16. Какой тайминг считается оптимальным для питча на 10 слайдов?
17. Почему на слайдах нельзя писать много текста?
18. Какие техники работы с возражениями вы знаете?
19. Что такое «призыв к действию» (CTA) в презентации? Приведите пример.
20. Почему важно репетировать выступление перед реальной презентацией?

Практическая работа №22

Презентация проекта и переговоры

Цель: систематизировать и оформить в единый структурированный документ все артефакты проекта, созданные в ходе выполнения практических работ (техническое задание, диаграммы, макеты, API-документация, экономические расчёты и др.). Итоговый отчёт должен быть представлен в формате Markdown и отражать полноту проектной документации.

Задачи:

1. Собрать и систематизировать все материалы проекта по разделам.
2. Оформить итоговый отчёт в формате Markdown с использованием заголовков, таблиц, списков, встраиванием изображений и кода.
3. Проверить целостность и согласованность документов (например, чтобы функциональные требования соответствовали диаграммам use case).
4. Подготовить навигацию по документу (оглавление, ссылки на разделы).
5. Экспортировать отчёт в PDF (или оставить в .md для сдачи).
6. Сдать итоговый отчёт как финальный артефакт проекта.

Теоретическая часть

1. Зачем нужен итоговый отчёт

Итоговый отчёт по проекту — это единый документ, который:

- **Фиксирует результаты** всех этапов проектирования и разработки.
- **Обеспечивает целостность** — все артефакты собраны в одном месте, видна их связь.
- **Упрощает проверку** преподавателем или заказчиком (не нужно открывать 20 отдельных файлов).
- **Служит основой для защиты** проекта (курсовой работы, диплома).

В учебном проекте итоговый отчёт объединяет:

- Техническое задание (основание, назначение, функциональные и нефункциональные требования).
- Модели (диаграммы UML: use case, классов).
- Прототипы интерфейсов (wireframes).
- Документацию API.
- Экономические расчёты (себестоимость, цена).
- Управленческие артефакты (WBS, оценки, Kanban).
- Презентационные материалы (ссылка на питч-дек).

2. Структура итогового отчёта (рекомендуемая)

Ниже приведена универсальная структура, которую можно адаптировать под учебный проект.

№	Раздел	Содержание
1	Введение	Краткое описание проекта, цель и задачи, актуальность.
2	Техническое задание (ТЗ)	Основание, назначение, функциональные требования (MoSCoW), нефункциональные требования (надежность, эргономика, совместимость), требования к техническим средствам.
3	Анализ предметной области	Глоссарий, описание бизнес-процессов (IDEF0 – контекстная диаграмма), актёры и сценарии use case (диаграмма + детальные сценарии).
4	Архитектура и моделирование	Диаграмма классов (сущности, атрибуты, методы, ассоциации с множественностью).
5	Проектирование интерфейса	Wireframes (low-fidelity макеты экранов), навигационная карта.
6	API / Интерфейсы	Описание функций системы (методы, параметры, ответы, примеры).
7	Управление проектом	WBS (декомпозиция), оценки трудозатрат, план итераций, Kanban-доска (скриншот).
8	Экономика проекта	Расчёт себестоимости разработки (ФОТ, машинное время, накладные), цена для заказчика, точка безубыточности.
9	Правовые аспекты	Выбранная лицензия на ПО, обоснование.
10	Заключение	Достигнутые результаты, перспективы развития, выводы.
11	Список источников	Ссылки на ГОСТ, учебные материалы, аналоги.

12	Приложение	Питч-дек (ссылка или встроенные слайды), ссылка на репозиторий/Yougile.

3. Оформление в Markdown

Почему Markdown? — лёгкий язык разметки, поддерживается на GitHub, легко конвертируется в PDF, позволяет вставлять код, таблицы, изображения, математические формулы.

Базовые элементы для отчёта:

- Заголовки #, ##, ### для структуры.
- Списки - или 1. для перечислений.
- Таблицы через | и |---|.
- Вставка изображений: ![alt](путь/к/файлу.png).
- Блоки кода (например, для JSON-примеров) с указанием языка.
- Ссылки: [текст](URL).
- Автоматическое оглавление (в некоторых парсерах — [TOC]).

Рекомендации по вёрстке:

- Используйте переносы строк для разделения абзацев.
- Для сложных диаграмм (UML) вставляйте скриншоты, т.к. Mermaid может не отображаться в PDF.
- Нумеруйте таблицы и рисунки (например, «Таблица 1 — Функциональные требования»).
- Добавьте титульный лист (можно в виде текста или отдельной страницы при экспорте).

4. Процесс сборки отчёта

1. **Инвентаризация** – выпишите все файлы, созданные в ПР1–ПР21.
2. **Проверка актуальности** – убедитесь, что правки после рецензий внесены.
3. **Структурирование** – распределите материалы по разделам.
4. **Написание связующего текста** – между разделами должны быть краткие пояснения.

5. **Вёрстка в Markdown** – один .md-файл (или несколько, если большой).
6. **Экспорт в PDF** – через Pandoc, Markdown Preview Enhanced (VS Code), Typora или онлайн-конвертеры.
7. **Финальная проверка** – читаемость, наличие всех изображений, корректность ссылок.

5. Типичные ошибки при оформлении отчёта

Ошибка	Как избежать	Ошибка
Разрозненные файлы (ТЗ в Word, диаграммы в PNG)	Всё собрать в один .md (диаграммы — как встроенные изображения).	Разрозненные файлы (ТЗ в Word, диаграммы в PNG)
Отсутствие навигации	Добавить оглавление и ссылки на разделы.	Отсутствие навигации
Нечитаемые изображения	Использовать достаточное разрешение (не менее 1000 px по ширине), подписывать.	Нечитаемые изображения
Противоречия между разделами	Проверить, что требования в ТЗ соответствуют use case и диаграмме классов.	Противоречия между разделами
Отсутствие выводов	В конце каждого раздела (или в заключении) подвести итог.	Отсутствие выводов

Практическая часть

Выполняется **в командах**. Студенты собирают итоговый отчёт из всех ранее созданных артефактов.

Задание 1. Инвентаризация артефактов

Составьте список всех файлов/документов, которые должны войти в итоговый отчёт. Используйте таблицу:

Практическая работа	Созданные артефакты	Где хранятся (ссылка / папка)
ПР1	...	...
ПР2	...	...
...	...	...
ПР21	...	...

Задание 2. Создание файловой структуры

Создайте папку project_report и внутри:

- README.md – основной файл отчёта.
- Папку images – для всех скриншотов и диаграмм.
- Папку diagrams – для исходных файлов диаграмм (Draw.io, Mermaid), если нужно.
- (Опционально) assets для дополнительных материалов.

Задание 3. Написание введения

Напишите введение (1–2 абзаца):

- Название проекта.
- Проблема, которую решает продукт.
- Цель работы (разработать документацию ...).
- Перечень основных разделов отчёта.

Задание 4. Сборка разделов отчёта (30 минут)

Последовательно заполните каждый раздел отчёта (см. п. 3.2), вставляя:

- Текст (цели, задачи, обоснования) из ранее написанных работ.
- Таблицы (требования, WBS, экономические расчёты).
- Изображения (контекстная диаграмма IDEF0, диаграммы use case и классов, wireframes, скриншот Kanban-доски, скриншот Wiki/Issues).
- Код (примеры JSON для API).

Важно: не копируйте механически всю методичку, а переносите только результаты вашей команды.

Задание 5. Добавление заключения и списка источников (5 минут)

Напишите заключение (1–2 абзаца):

- Что удалось выполнить.
- Какие трудности возникли.
- Какие перспективы развития проекта.

Добавьте список использованных источников (ГОСТ, учебные пособия, ссылки на аналоги).

Задание 6. Оформление оглавления и навигации (5 минут)

В начале файла добавьте оглавление вручную или используйте автоматическое (например, [TOC], если поддерживается). Проверьте, что все внутренние ссылки работают.

Задание 7. Экспорт в PDF (5 минут)

Экспортируйте .md-файл в PDF:

- **VS Code** – расширение «Markdown PDF» (Ctrl+Shift+P → Markdown PDF: Export (pdf)).
- **Typora** – Файл → Экспорт → PDF.
- **Онлайн** – https://www.markdowntopdf.com/ (но будьте осторожны с изображениями).
- **Pandoc** – pandoc report.md -o report.pdf.

Задание 8. Финальная проверка (5 минут)

Проверьте:

- Все изображения отображаются, нет битых ссылок.
- Таблицы не выходят за границы страницы в PDF.
- Номера рисунков и таблиц соответствуют тексту.
- Отсутствуют орфографические ошибки (можно прогнать через проверку в Word или Google Docs).

Требования к отчёту

Отчёт сдаётся командой в электронном виде (Markdown, PDF или DOCX).

Структура отчёта:

Отчёт должен соответствовать структуре из п. 3.2 (введение, ТЗ, анализ, архитектура, интерфейсы, управление, экономика, правовые аспекты, заключение, приложения). Обязательные элементы:

- Титульная информация (название проекта, команда, дата).
- Оглавление.
- Все таблицы и рисунки подписаны и пронумерованы.
- Ссылки на репозиторий / Yougile / питч-дек в приложении.

Технические требования

1. Формат: .md + экспортированный .pdf (оба файла сдаются).
2. Шрифт в PDF – на усмотрение конвертера, но должен быть читаемым.
3. Изображения в формате PNG или JPG, разрешение не менее 72 DPI, текст на изображениях читаем.
4. Размер файла PDF не более 20 МБ (при большом количестве скриншотов можно сжать).

Контрольные вопросы

1. Зачем нужен итоговый отчёт по проекту?
2. Какие разделы обязательно должны быть в отчёте? Назовите не менее 6.
3. Почему для оформления отчёта рекомендуется использовать Markdown?
4. Как в Markdown вставить изображение? Приведите пример синтаксиса.
5. Как создать таблицу в Markdown? Приведите пример.

6. Что делать, если диаграмма не отображается при экспорте в PDF?

7. Как оформить автоматическое оглавление в Markdown?

8. Какие инструменты можно использовать для конвертации Markdown в PDF?

9. Почему важно проверять согласованность артефактов (например, ТЗ и диаграммы)?

10. Что нужно указать в разделе «Заключение»?

Практическая работа №23

Предзащита проектов

Цель: провести пробную защиту проектов перед итоговой презентацией, получить обратную связь от других команд и преподавателя, выявить слабые места в выступлении и документации. Освоить навыки кросс-рецензирования и конструктивной критики.

Задачи:

1. Провести пробное выступление каждой команды в формате, приближённом к итоговой защите.
2. Продемонстрировать ключевые артефакты проекта: техническое задание, диаграммы, макеты, питч-дек, итоговый отчёт (если готов).
3. Осуществить кросс-рецензирование: одна команда оценивает выступление другой по установленным критериям.
4. Собрать и зафиксировать замечания и рекомендации для доработки.
5. Составить план доработки проекта и презентации перед итоговой защитой.

Теоретическая часть

1. Цели и формат предзащиты

Предзащита – это репетиция итоговой защиты проекта, которая позволяет:

- Проверить готовность команды к публичному выступлению.
- Выявить пробелы в содержании и оформлении материалов.
- Получить внешнюю оценку от коллег и преподавателя.
- Скорректировать тайминг и структуру выступления.

Формат предзащиты:

- Каждая команда выступает **7–10 минут** (как на итоговой защите).
- После выступления – вопросы от аудитории (3–5 минут).

- Другая команда (или несколько) заполняет оценочный лист (рецензию).
- Преподаватель даёт общие рекомендации.

2. Структура выступления на предзащите

Часть	Длительность	Содержание
Вступление	30 сек	Название проекта, команда, проблема
Основная часть	6–8 мин	Решение, уникальность, демонстрация (wireframes/прототип), технологии, план, экономика (кратко)
Заключение	30 сек	Итоги, перспективы, призыв к действию
Ответы на вопросы	3–5 мин	Реакция на замечания, уточнения

3. Критерии оценки выступления (для рецензентов)

Критерий	Описание	Оценка (1–5)
Содержание	Полнота раскрытия темы, чёткость формулировок	
Структура	Логичность переходов, наличие вступления, основной части, заключения	
Визуализация	Качество слайдов, читаемость, уместность схем и скриншотов	
Презентационные навыки	Громкость, темп речи, контакт с аудиторией, жестикуляция	
Тайминг	Укладывание в регламент (7–10 мин)	
Ответы на вопросы	Аргументированность, честность, знание материала	
Командная работа	Распределение ролей, синхронность, смена выступающих	

4. Правила кросс-рецензирования

Рецензент (оценивающая команда) должен:

- Внимательно слушать, не перебивать.
- Заполнять оценочный лист объективно.
- Фиксировать конкретные замечания (не «плохо», а «слайд 3 перегружен текстом»).
- После выступления задать 1–2 уточняющих вопроса.

- Предложить конструктивные рекомендации для улучшения.

Выступающая команда должна:

- Воспринимать критику как возможность улучшить проект.
- Не оправдываться, а благодарить за обратную связь.
- Записывать замечания для доработки.

5. Типичные ошибки на защите

Ошибка	Как исправить
Чтение со слайдов или бумажки	Выучить ключевые тезисы, слайды – только подсказки
Превышение времени	Отрепетировать с секундомером, сократить лишнее
Монотонная речь	Добавить паузы, менять интонацию, выделять главное
Отсутствие демонстрации	Показать скриншоты, wireframes, прототип
Неподготовленные ответы на вопросы	Продумать возможные вопросы заранее
Слишком много текста на слайдах	Оставить только ключевые слова, подробности – устно

Практическая часть

Выполняется **в командах**. В аудитории присутствуют все команды. Преподаватель назначает порядок выступлений и пары для кросс-рецензирования.

Задание 1. Подготовка к выступлению

Каждая команда за 10 минут до своего выступления:

- Проверяет работоспособность презентации (на проекторе или в Zoom).
- Распределяет роли, кто какие слайды ведёт.
- Проверяет тайминг (можно засечь репетицию).

Задание 2. Выступление команд (по 10–12 минут на команду)

Команда представляет проект по следующему плану:

1. Название, проблема, целевая аудитория (1 мин).
2. Решение, уникальность (1,5 мин).

3. Демонстрация wireframes / прототипа (2 мин).
4. Технологии и архитектура (1 мин).
5. План разработки (WBS, итерации) (1 мин).
6. Экономика (себестоимость, цена) – кратко (1 мин).
7. Команда, выводы, призыв (1 мин).
8. Ответы на вопросы (3–5 мин).

Задание 4. Вопросы и обсуждение (3–5 минут после каждого выступления)

После выступления преподаватель и другие команды задают вопросы. Рецензент задаёт **один обязательный вопрос**. Вопросы могут касаться:

- Обоснования выбора технологии.
- Реализации сложных функций.
- Рисков и их смягчения.
- Экономических расчётов.

Задание 5. Сбор обратной связи и составление плана доработки

Каждая команда после всех выступлений:

- Собирает заполненные оценочные листы от рецензента и других желающих.
- Фиксирует **3–5 конкретных улучшений** для презентации и/или документации.
- Записывает план действий: что исправить, к какому сроку, кто ответственный.

Пример плана доработки:

Что улучшить	Действие	Срок	Ответственный
Слайд 3 перегружен текстом	Сократить до 5 строк, добавить иконку	Завтра	Аналитик
Не хватает демонстрации прототипа	Добавить 2 скриншота wireframes	Завтра	Дизайнер

Превысили тайминг на 1,5 мин	Убрать подробности экономики из устной речи	Сегодня	Капитан

Задание 6. Рефлексия

Каждый участник команды письменно (в тетради или в общем документе) отвечает на вопросы:

- Что получилось лучше всего в нашем выступлении?
- Что нужно срочно исправить?
- Какие вопросы аудитории были неожиданными?
- Как я могу помочь команде подготовиться к итоговой защите?

Требования к отчёту

Отчёт сдаётся каждой командой после предзащиты.

Структура отчёта:

1. Титульный лист (название команды, участники, дата предзащиты).
2. Оценочные листы от рецензента (скан/фото или таблица).
3. Сводная таблица замечаний (из всех полученных рецензий).
4. План доработки (задание 5) с указанием сроков и ответственных.
5. Рефлексия команды (общий вывод: что удалось, что нет, как готовиться к итоговой защите).
6. Ссылка на обновлённую презентацию (после доработок – можно позже).

Технические требования

1. Формат: .pdf или .md.

Контрольные вопросы

1. Почему предзащита важна перед итоговой защитой проекта?
2. Какие три основных критерия оценки выступления вы считаете самыми важными?
3. Что делать, если выступающий забыл текст или сбился?
4. Как правильно задавать вопросы выступающим, чтобы не быть агрессивным?
5. Что такое «кросс-рецензирование»? Какую пользу оно приносит обеим сторонам?
6. Как реагировать на критику, с которой вы не согласны?
7. Почему важно соблюдать тайминг? Какие последствия превышения времени?
8. Какой вопрос на защите самый сложный и почему?
9. Что важнее: красивые слайды или уверенная речь?
10. Как распределить роли в команде, чтобы выступление выглядело единым?

Практическая работа №24

Итоговая защита проектов

Цель: продемонстрировать результаты проектной деятельности: представить разработанный проект, обосновать принятые решения (архитектурные, функциональные, экономические), ответить на вопросы комиссии и подтвердить уровень сформированных компетенций в области программной инженерии.

Задачи:

1. Представить итоговый проект в форме публичной презентации (10–12 минут).
2. Продемонстрировать понимание Use Case диаграмм, Class Diagram и их связи с требованиями.
3. Обосновать выбор технологий, архитектурных решений и модели жизненного цикла.
4. Показать результаты экономического обоснования (себестоимость, цена, окупаемость – при наличии).
5. Ответить на вопросы комиссии (преподавателя и сокурсников).

Теоретическая часть

1. Формат итоговой защиты

Итоговая защита проектов является завершающим этапом курса. Каждая команда (или индивидуальный студент) представляет результаты своей работы перед комиссией (преподаватель, возможно, приглашённые эксперты). Формат защиты включает:

- **Выступление** – 10–12 минут (регламент жёсткий).
- **Демонстрацию** – слайды питч-дека, скриншоты прототипов, фрагменты документации.
- **Вопросы** – от комиссии и аудитории (5–10 минут).

- **Оценку** – на основе заранее объявленных критериев.

2. Структура выступления на защите

Часть	Время	Содержание
Приветствие и название	0:30	Название проекта, команда, проблема (одно предложение)
Актуальность и проблема	1:00	Почему это важно? Цифры, факты, примеры
Решение и MVP	1:30	Какие функции реализованы (Must have), уникальное ценностное предложение
Use Case и архитектура	2:00	Диаграмма вариантов использования (актёры, прецеденты), диаграмма классов (ключевые сущности)
Демонстрация интерфейса	1:30	Wireframes / прототип (скриншоты, можно анимацию)
Технологии и управление	1:30	Стек, WBS, оценки трудозатрат, Kanban (как работали)
Экономика (если есть)	1:00	Себестоимость, цена, окупаемость
Заключение и перспективы	0:30	Что сделано, что планируется, благодарность

Важно: Время жёстко контролируется. За 1 минуту до окончания подаётся сигнал.

3. Ключевые демонстрационные артефакты

Артефакт	Где взять	Как показать
Use Case Diagram	ПР9	Вставить на слайд, кратко пояснить актёров и 2-3 главных прецедента
Class Diagram	ПР12-13	Показать основные классы-сущности и связи (множественность)
Wireframes	ПР18	Скриншоты 2-3 ключевых экранов (логин, главный, форма записи)
Требования (MoSCoW)	ПР6	Таблицей на слайде – что сделано (Must have)
WBS и оценки	ПР17	Диаграмма или таблица трудоёмкости
Kanban	ПР17	Скриншот доски Yougile/GitHub Projects
Экономика	ПР20	Себестоимость, цена (кратко)

4. Типичные вопросы комиссии и как на них отвечать

Вопрос	Ожидаемый ответ
Почему вы выбрали именно эту архитектуру / технологию?	Обосновать: требования к производительности, опыту команды, совместимости, срокам.
Какой use case самый важный? Покажите на диаграмме.	Назвать и показать на слайде (например, «Записаться на приём»).
Как связаны классы X и Y? Какая множественность?	Объяснить бизнес-логику: «один заказ содержит много позиций, поэтому 1..*».
Почему это требование – Must have, а не Should have?	«Без этой функции продукт теряет ценность для пользователя».
Как вы оценивали трудоёмкость? Какой коэффициент использовали?	«Экспертно, с учётом неопытности – коэффициент 1,5–2».
Что было самым сложным в проекте?	Честно назвать (например, согласование требований, интеграция API).
Как вы собираетесь развивать продукт после MVP?	Перечислить 2-3 функции из Should/Could have.

Общие правила ответов:

- Не говорите «не знаю» – скажите «не готов ответить сейчас, но мы это проработаем».
- Не спорьте с комиссией, а благодарите за вопрос и уточняйте.
- Если вопрос не понятен – попросите переформулировать.

5. Критерии оценки защиты (для преподавателя)

Критерий	Вес	Описание
Полнота представления проекта	25%	Раскрыты все разделы (проблема, решение, архитектура, интерфейс, управление)
Понимание Use Case и Class Diagram	20%	Корректное объяснение актёров, прецедентов, связей, множественности
Качество презентации	15%	Дизайн слайдов, читаемость, визуализация
Презентационные навыки	15%	Громкость, темп, контакт с аудиторией, тайминг
Ответы на вопросы	15%	Аргументированность, честность, знание деталей

Командная работа	10%	Распределение ролей, синхронность, помощь друг другу

6. Подготовка к защите: чек-лист

За 2–3 дня до защиты:

- Питч-дек доработан с учётом замечаний предзащиты (ПР22).
- Все изображения (диаграммы, скриншоты) чёткие, подписи читаемы.
- Репетиция с секундомером – 2-3 раза.
- Продуманы ответы на возможные вопросы (список из 5–7 вопросов).
- Распределены роли: кто начинает, кто показывает диаграммы, кто отвечает за технические вопросы, кто за экономику.
- Файл презентации сохранён на флешку/облако (запасная копия).

Практическая часть

Выполняется **в командах** в аудитории в присутствии комиссии. Время на защиту – до 20 минут на команду (включая вопросы).

Задание 1. Финальная проверка материалов

Убедитесь, что:

- Презентация открывается на компьютере комиссии.
- Все ссылки на скриншоты работают (если встроены).
- Выступающие готовы, у каждого есть тезисы (можно карточки).

Задание 2. Выступление команды (10–12 минут)

Выступление строго по регламенту. Один из членов команды следит за временем (показывает табличку «5 минут», «1 минута», «стоп»). Если время вышло – выступающий должен завершить текущую мысль и передать слово комиссии.

Дополнительное требование: обязательно продемонстрировать **понимание Use Case и Class Diagram** – т.е. не просто показать картинку, а

кратко пояснить: кто актёры, какие прецеденты ключевые, какие классы и как связаны.

Задание 3. Ответы на вопросы (5–10 минут)

Комиссия (преподаватель и, возможно, другие студенты) задаёт вопросы. Каждый член команды должен быть готов ответить на вопрос по своей части. Нельзя отвечать одному и тому же человеку более чем на два вопроса подряд.

Задание 4. Подведение итогов (после всех выступлений)

Преподаватель оглашает предварительные оценки (или объявляет результаты после проверки). Студенты могут задать вопросы по оценке.

Требования к отчёту

После защиты команда сдаёт финальный пакет документов (если не сдали ранее):

Состав финального пакета:

1. Итоговый отчёт (ПР23) в формате PDF (и исходный .md).
2. Презентация (питч-дек) в формате PDF или PowerPoint.
3. Ссылка на репозиторий (GitHub/GitLab) со всеми артефактами (Wiki, диаграммы, прототипы).
4. Ссылка на Kanban-доску (Yougile или GitHub Projects) с историей задач.

Технические требования

1. Все файлы должны быть собраны в одном архиве **Фамилия_команды_защита.zip** или предоставлены ссылками на облачные сервисы (с открытым доступом).
2. Итоговый отчёт – в PDF, все изображения встроены.
3. Презентация – в PDF (для гарантии совместимости).

Контрольные вопросы

1. Какова основная проблема, которую решает ваш проект?
2. Назовите 3 ключевых функциональных требования (Must have).
3. Покажите на диаграмме use case: кто главный актёр и какой прецедент самый важный?
4. Как связаны классы [класс А] и [класс Б] на вашей диаграмме? Какая множественность?
5. Почему вы выбрали именно этот стек технологий?
6. Как вы оценивали трудоёмкость? Сколько человеко-часов ушло на MVP?
7. Какова себестоимость разработки вашего продукта? Цена для заказчика?
8. Что бы вы сделали по-другому, если бы начинали проект заново?
9. Какие риски вы предусмотрели и как их смягчали?
10. Какие перспективы развития у вашего проекта после MVP?

Список литературы

1. Нормативные документы и стандарты

1. ГОСТ 34.602-89. Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы. – М.: Стандартинформ, 2009.

2. ГОСТ Р 54869-2011. Проектный менеджмент. Требования к управлению проектом. – М.: Стандартинформ, 2012.

3. ISO/IEC 12207:2008. Systems and software engineering – Software life cycle processes. (Аналог ГОСТ Р ИСО/МЭК 12207-2010).

4. ISO/IEC 25010:2011. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models.

5. ГОСТ Р ИСО/МЭК 20926-2014. Информационная технология. Оценка программного продукта. Измерение функционального размера. Метод COSMIC.

2. Учебники и учебные пособия

6. Соммервилл, И. Инженерия программного обеспечения = Software Engineering / И. Соммервилл. – 10-е изд. – М.: Вильямс, 2016. – 960 с.

7. Коберн, А. Современные методы описания функциональных требований к системам / А. Коберн. – М.: Лори, 2019. – 340 с. (Use case modeling).

8. Фаулер, М. UML. Основы = UML Distilled: A Brief Guide to the Standard Object Modeling Language / М. Фаулер. – 3-е изд. – СПб.: Символ-Плюс, 2014. – 192 с.

9. Вендров, А.М. Проектирование программного обеспечения / А.М. Вендров. – М.: Финансы и статистика, 2019. – 560 с.

10. Орлов, С.А. Технологии разработки программного обеспечения / С.А. Орлов, Б.Я. Цилькер. – 5-е изд. – СПб.: Питер, 2018. – 704 с.

11. Брукс, Ф. Мифический человеко-месяц / Ф. Брукс. – СПб.: Символ-Плюс, 2010. – 304 с. (управление проектами, оценки).

12. Гласс, Р. Фактор качества. Практика высококачественного программирования / Р. Гласс. – М.: Лори, 2014. – 310 с.

3. Управление проектами, методологии, экономика

13. PMI. Руководство к Своду знаний по управлению проектами (PMBOK Guide). – 7-е изд. – Project Management Institute, 2021. (русское издание: PMBOK® Guide, 2021).

14. Швабер, К. Scrum. Революционный метод управления проектами / К. Швабер, Д. Сазерленд. – М.: Эксмо, 2016. – 288 с.

15. Андерсон, Д. Канбан. Альтернативный путь в Agile / Д. Андерсон. – М.: Манн, Иванов и Фербер, 2017. – 336 с.

16. Мадера, А.Г. Риски и шансы: неопределенность, прогнозирование и оценка / А.Г. Мадера. – М.: Едиториал УРСС, 2019. (для экономических оценок).

17. Одинцова, Т.М. Экономика информационных систем / Т.М. Одинцова. – М.: Финансы и статистика, 2018. – 288 с.

4. Сбор требований, анализ, моделирование бизнес-процессов

18. Вигерс, К. Разработка требований к программному обеспечению / К. Вигерс, Дж. Битти. – 3-е изд. – М.: Русская редакция, 2019. – 736 с.

19. Робертсон, С. Mastering the Requirements Process: Getting Requirements Right / С. Робертсон, Дж. Робертсон. – 4th ed. – Addison-Wesley, 2012.

20. Калянов, Г.Н. Моделирование, анализ, реорганизация и автоматизация бизнес-процессов / Г.Н. Калянов. – М.: Финансы и статистика, 2017. – 240 с. (IDEF0).

5. Инструментарий и практики разработки

21. Чакон, С. Pro Git / С. Чакон, Б. Штрауб. – 2-е изд. – Apress, 2014. (русский перевод: Про Git, доступен онлайн на git-scm.com/book/ru).

22. Макконнелл, С. Совершенный код / С. Макконнелл. – 2-е изд. – СПб.: Питер, 2018. – 896 с. (главы о документировании и оценках).

23. Толыпин, С.Ю. Markdown для технических писателей / С.Ю. Толыпин // Техническая документация в IT. – 2020. – №2. – С. 15-22. (доступно в интернете).

24. Галкин, В.А. Прототипирование интерфейсов: инструменты и методы / В.А. Галкин. – М.: ДМК Пресс, 2019. – 180 с.

6. Правовые аспекты в ИТ

25. Гражданский кодекс Российской Федерации. Часть четвертая (ст. 1255–1302) – авторское право.

26. Кодекс Российской Федерации об административных правонарушениях (КоАП РФ). Ст. 7.12.

27. Уголовный кодекс Российской Федерации (УК РФ). Ст. 146.

28. Кузьмина, О.А. Правовая охрана программ для ЭВМ и баз данных / О.А. Кузьмина. – М.: Юрайт, 2020. – 220 с.

29. Open Source Initiative (OSI). Список одобренных лицензий: https://opensource.org/licenses (дата обращения: 2026).

7. Интернет-ресурсы и документация

30. Agile Manifesto (Манифест гибкой разработки) – http://agilemanifesto.org/ (русская версия).

31. GitHub Docs – https://docs.github.com/ru (Wiki, Issues, Projects).

32. GitLab Documentation – https://docs.gitlab.com/ (Wiki, Issues, Boards).

33. Mermaid.js Documentation – https://mermaid.js.org/ (синтаксис диаграмм).

34. Yougile – справка – https://yougile.com/help (канбан-доска, задачи).

35. Draw.io Guides – https://www.drawio.com/guides (создание диаграмм UML, IDEF0).

36. Markdown Guide – https://www.markdownguide.org/ (полное описание синтаксиса).

37. Balsamiq Wireframes – документация – https://balsamiq.com/learn/ (прототипирование).

38. Choose a License – https://choosealicense.com/ (помощник по выбору лицензии).