

Федеральное государственное образовательное бюджетное
учреждение высшего образования
«Финансовый университет при Правительстве Российской Федерации»
(Финансовый университет)

Факультет информационных технологий и анализа больших данных

Кафедра информационных технологий

Выпускная квалификационная работа
на тему: «Разработка веб-приложения для организации совместной работы над проектами»

Направление подготовки 09.03.03 «Прикладная информатика»,
Профиль «ИТ-сервисы и технологии обработки данных в экономике и финансах»

Выполнил студент группы ПИ21-3
Балашкин Андрей Михайлович _____
Руководитель к.т.н., доцент
Хасанов Ильнур Ильдарович _____

**ВКР соответствует предъявленным
требованиям**

Заведующий кафедрой информационных
технологий
к.т.н., доцент

_____ Д.А. Петросов
« ____ » _____ 20 ____ г.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ.....	3
1. Анализ и оценка существующих решений.....	5
1.1 Существующие решения	5
1.2 Определение потребностей пользователей	13
1.3 Составление требований	19
2. Теоретическая часть.....	26
2.1 Основы проектирования веб приложений.....	26
2.2 Архитектура веб приложений.....	28
2.3 Обоснование выбранных технологий	33
2.4 Обеспечение безопасности данных.....	38
3. Практическая часть	42
3.1 Разработка прототипа	42
3.2 Разработка серверной части	48
3.3 Разработка клиентской части.....	50
3.4 Тестирование приложения	54
ЗАКЛЮЧЕНИЕ	59
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	63
ПРИЛОЖЕНИЕ А	66
ПРИЛОЖЕНИЕ Б.....	67
ПРИЛОЖЕНИЕ В	68
ПРИЛОЖЕНИЕ Г	69
ПРИЛОЖЕНИЕ Д	70

ВВЕДЕНИЕ

В современном мире цифровых технологий коллективная работа над проектами становится неотъемлемой частью рабочих процессов во многих сферах деятельности. С развитием удаленной занятости, распределенных команд и гибких методологий управления проектами возрастает потребность в удобных, функциональных и интуитивно понятных инструментах для совместного планирования, координации задач и обмена информацией.

Существующие решения, такие как Trello, Asana, Jira и другие, предоставляют широкий набор возможностей для управления проектами. Однако многие из них либо сложны в освоении для пользователей без технического опыта, либо имеют ограничения по функционалу в бесплатных версиях. Кроме того, универсальные платформы не всегда учитывают специфические потребности небольших команд и стартапов, которым требуется гибкость в настройке рабочего пространства.

Актуальность исследования обусловлена необходимостью создания удобного и доступного инструмента для командной работы, который обеспечит базовые функции управления проектами: ведение списка задач (todo-лист), работу с канбан-доской, создание и хранение текстовых записей, а также возможность добавления участников с равными правами. Такой инструмент поможет небольшим коллективам эффективно распределять задачи, отслеживать прогресс и централизованно хранить информацию.

Цель работы — разработка веб-приложения для организации совместной работы над проектами, обеспечивающего простоту в управлении задачами и удобство взаимодействия пользователей.

Для достижения этой цели необходимо решить следующие задачи:

- Провести анализ существующих решений и выявить их преимущества и недостатки.
- Определить потребности целевой аудитории и сформировать требования к разрабатываемому приложению.

– Изучить принципы проектирования веб-приложений и обосновать выбор используемых технологий.

– Реализовать серверную и клиентскую части веб-приложения.

– Провести тестирование разработанного решения и оценить его эффективность.

Объект исследования — процессы организации совместной работы над проектами с использованием цифровых инструментов.

Предмет исследования — методы и технологии разработки веб-приложений для управления проектами.

Таким образом, данная работа представляет собой комплексное исследование, включающее анализ предметной области, выбор оптимальных решений и реализацию программного продукта, который может быть использован в практической деятельности небольших команд и проектных групп.

1. Анализ и оценка существующих решений

1.1 Существующие решения

Совместная работа над проектами, как важная часть организационной деятельности, существует с момента возникновения коллективной работы. В течение долгого времени основными средствами для взаимодействия сотрудников являлись традиционные способы коммуникации, такие как бумажные документы, телефонные звонки и встречи. Однако с развитием информационных технологий и массовым распространением интернета возникла необходимость в создании специализированных инструментов для упрощения и улучшения рабочих процессов [1].

Первые инструменты для совместной работы, такие как электронная почта и текстовые редакторы для группового редактирования документов, стали популярными в конце 1990-х и начале 2000-х годов. Эти решения, несмотря на свою полезность, не удовлетворяли потребности в эффективном управлении проектами и задачами. С появлением таких платформ, как Basecamp (2004 год), а затем и Trello, Asana, Jira и других, стало возможным более системно подходить к распределению задач, мониторингу выполнения и координации работы распределенных команд.

В последние годы цифровые инструменты для управления проектами и совместной работы получили широкое распространение. Это связано с активным развитием удаленного формата работы, увеличением количества распределенных команд, а также с ростом популярности гибких методологий управления проектами, таких как Agile, Kanban и Scrum. В условиях динамично меняющейся бизнес-среды компаниям и командам требуются эффективные решения, которые позволяют структурировать рабочие процессы, отслеживать выполнение задач, вести совместные заметки и упрощать взаимодействие между участниками.

Сегодня на рынке представлено множество платформ, ориентированных на коллективную работу. Каждое из этих решений обладает собственным набором функциональных возможностей, направленных на повышение продуктивности и удобства управления проектами. Но не все существующие инструменты одинаково хорошо подходят для небольших команд или пользователей, которым важна простота и интуитивность интерфейса.

Многие современные системы управления проектами содержат широкий спектр функций, среди которых интеграция с другими сервисами, настройка пользовательских ролей, автоматизация процессов, аналитика и многое другое. Несмотря на это, перегруженность интерфейса и сложные настройки могут стать препятствием для быстрого освоения инструмента, особенно если команда нуждается в простом и легком решении. Дополнительно следует учитывать, что ряд популярных платформ в своих бесплатных версиях накладывает ограничения на количество пользователей, проектов или доступных функций, что может создавать определенные неудобства для небольших коллективов.

Для команд, которым важно минимизировать время на освоение системы, упростить управление задачами, организовать совместное ведение заметок и визуализировать процессы с помощью канбан-доски, перегруженные функциональностью решения могут оказаться менее удобными. В таких случаях предпочтение отдается инструментам, обладающим простым и понятным интерфейсом, а также возможностью быстрого доступа к ключевым функциям.

В данном разделе рассмотрены наиболее популярные инструменты, их ключевые возможности, а также их преимущества и недостатки с точки зрения простоты и удобства.

Trello:

Trello — одна из самых популярных и широко используемых платформ для управления задачами, основанная на принципах методологии Kanban. Этот инструмент позволяет визуально структурировать рабочие процессы, облегчая планирование и отслеживание задач в команде. Благодаря своей интуитивно

понятной структуре и гибкости, Trello подходит как для индивидуального использования, так и для работы в командах разного масштаба [2].

В основе Trello лежит концепция доски, которая представляет собой цифровой аналог физической канбан-доски. Доска состоит из списков, в которые добавляются карточки задач. Списки, как правило, отображают различные этапы выполнения работы, например:

- "Запланировано" — для задач, которые только предстоит выполнить.
- "В процессе" — для задач, над которыми идет активная работа.
- "Готово" — для завершенных задач.

Такое визуальное представление позволяет легко отслеживать текущий статус задач и оперативно реагировать на изменения. Карточки можно перемещать между списками, отражая текущий статус работы над задачей. Это делает процесс управления проектами простым и прозрачным.

Одним из ключевых преимуществ Trello является возможность автоматизации процессов. Это реализуется с помощью встроенного инструмента Butler, который позволяет создавать автоматизированные правила и триггеры. Например, можно настроить, чтобы при перемещении карточки в список "Готово" система автоматически отмечала задачу как выполненную и уведомляла ответственного участника.

Преимущества:

- Простота использования и интуитивно понятный интерфейс.
- Поддержка совместной работы с возможностью комментирования и добавления вложений.
- Автоматизация процессов
- Кроссплатформенность
- Бесплатный тариф с основными функциями.

Недостатки:

- Ограниченные возможности в бесплатной версии (например, малое количество автоматизаций).

– Отсутствие встроенных инструментов для текстовых заметок и комплексного управления проектами.

– Сложность масштабирования, так как в крупных проектах с большим количеством задач доска становится перегруженной

Asana:

Asana — это платформа для управления проектами и задачами, которая предлагает пользователям несколько способов организации работы, включая списки, канбан-доски и календарь. Однако, несмотря на широкие функциональные возможности, Asana не всегда оказывается удобным решением, особенно для небольших команд, которым важны простота, легкость и быстрая адаптация к инструменту.

На первый взгляд, Asana предоставляет базовый набор возможностей для управления задачами: пользователи могут создавать списки дел, использовать канбан-доску для визуального представления процессов или отслеживать задачи через календарь. Не смотря на это, при более глубоком использовании становится очевидным, что система перегружена интерфейсными элементами, настройками и дополнительными опциями, которые далеко не всегда действительно необходимы.

Asana стремится охватить сразу все возможные сценарии работы с проектами, из-за чего ее интерфейс наполнен множеством кнопок, вкладок, настроек и переключателей. Для новичков это становится настоящей проблемой. Простая задача, которая могла бы быть создана за пару кликов, в Asana превращается в процесс с обязательным заполнением множества полей и параметров.

Сервис использует фримииум-модель, предлагая базовые функции бесплатно, но при этом активно продвигая платные подписки. В бесплатной версии накладываются жесткие ограничения: нельзя использовать продвинутые отчеты, недоступны некоторые виды автоматизации, а также ограничено количество пользователей и командных функций.

Преимущества:

- Гибкость в настройке задач и проектов.
- Поддержка интеграций с другими сервисами (Google Drive, Slack и др.).
- Возможность автоматизации процессов.

Недостатки:

- Сложность освоения для новых пользователей.
- Ограниченный функционал в бесплатной версии.
- Сложность в обмен на гибкость, которая часто бывает избыточной

Notion:

Notion сочетает в себе инструменты для создания текстовых записей, ведения базы данных, управления задачами и организации информации. В системе можно создавать страницы с текстами, таблицами и списками задач, а также связывать их друг с другом. Notion предлагает возможность совместного редактирования, что делает его удобным для командной работы.

Сервис позволяет создавать страницы, содержащие текст, списки задач, таблицы, вложенные базы данных и даже встраивать элементы других сервисов. Все эти элементы можно связывать между собой, формируя сложную систему взаимодействий. Чтобы грамотно структурировать информацию, пользователю приходится тратить время на освоение системы и создание логики работы внутри сервиса. Простая задача, например, добавление списка дел, может быстро усложниться, если попытаться интегрировать его с базой данных или связать с другими страницами.

В отличие от инструментов, которые позволяют сразу приступить к работе без лишних сложностей, Notion заставляет пользователя разбираться с многочисленными возможностями, многие из которых могут оказаться ненужными.

Хотя Notion поддерживает командное редактирование, оно далеко не всегда реализовано на удобном уровне. Отсутствует гибкая система уведомлений, редактирование в реальном времени иногда вызывает конфликты, история страниц ограничена в бесплатном тарифе.

Как и многие другие платформы, Notion использует модель монетизации фримииум. Если команда хочет получить полный контроль над своими данными, ей придется оформлять платную подписку

Преимущества:

- Универсальность: поддержка todo-листов, таблиц, документов и баз данных.
- Гибкость в настройке интерфейса под нужды пользователя.
- Возможность совместного редактирования контента.

Недостатки:

- Ограниченный функционал в бесплатной версии.
- Возможные проблемы с производительностью при работе с большими объемами данных.

Jira:

Jira — мощный инструмент, ориентированный на IT-команды и разработчиков, использующих методологии Agile и Scrum. Платформа предлагает гибкие настройки рабочих процессов, поддержку Scrum- и Kanban-досок, а также развитую систему отчетности и аналитики. Jira позволяет настраивать различные этапы выполнения задач, устанавливать зависимости между ними и интегрировать их с DevOps-инструментами.

В реальном использовании этот инструмент нередко становится головной болью, особенно для команд, которые не занимаются разработкой программного обеспечения. Jira — это инструмент с крайне сложной архитектурой, предназначенной для крупных команд с четко выстроенными процессами. Настроить рабочую среду под свои задачи без опыта работы с системой крайне сложно. Даже базовые операции, такие как создание задачи или изменение ее статуса, могут требовать множества дополнительных действий и изучения документации.¹

¹ Почему использовать Jira для ведения проектов неэффективно // VC URL: <https://vc.ru/services/54288-pochemu-ispolzovat-jira-dlya-vedeniya-proektov-neeffectivno> (дата обращения: 11.12.2024).

Веб-сервис идеально подходит для разработчиков программного обеспечения, но если вам нужно управлять проектами в других сферах — маркетинг, дизайн, контент, бизнес-аналитика — инструмент становится неудобным и громоздким.

Одним из ключевых преимуществ Jira считается развитая система отчетности, позволяющая анализировать выполнение задач, измерять эффективность работы команды и строить диаграммы загрузки. В небольших коллективах избыточная аналитика становится скорее помехой, чем преимуществом. Большинство команд просто не используют эти отчеты, а их настройка требует знаний и времени. В отличие от легких и интуитивных инструментов, где основные метрики видны сразу, в Jira нужно потратить часы на разбор интерфейса и настройку графиков.

Jira — это инструмент для крупных IT-команд, которым требуется полный контроль над рабочими процессами, глубокая аналитика и интеграция с DevOps-инструментами.

Преимущества:

- Гибкость настройки рабочих процессов.
- Поддержка Scrum- и Kanban-досок.
- Развитые функции отчетности и аналитики.

Недостатки:

- Высокий порог входа и сложность интерфейса.
- Платность большинства функций, необходимых для полноценной работы.

В таблице ниже (Таблица 1) приведен сравнительный анализ основных функций рассмотренных решений. Из нее следует, что наиболее популярны функции: список задач, канбан-доска, текстовые записи. Добавление других элементов сильно усложняет систему и не пользуется спросом для быстрого и простого управления проектом.

Таблица 1 - Сравнительный анализ основных функций существующих решений

Сервис	Основные функции
Trello	Канбан-доска, списки задач
Asana	Списки задач, канбан-доска, календарь
Jira	Scrum и канбан-доски, настройка рабочих процессов, отчетность
Notion	Текстовые записи, базы данных, канбан-доска, задачи

Существующие платформы обладают широкими возможностями, но часто ориентированы на специфические задачи, требуют сложной настройки или имеют ограничения в бесплатных версиях. Анализ показал, что далеко не все из них универсальны и удобны для небольших команд. Среди ключевых недостатков можно выделить перегруженный интерфейс, сложность освоения и ограничения бесплатных версий, что делает их не всегда удобными для широкого круга пользователей. В таблице 2 показаны достоинства и недостатки этих платформ.

Таблица 2 - Возможности существующих решений

	Управление задачами	Канбан	Текстовые записки	Простое управление ролями	Бесплатная версия
Trello	+	+	-	+	10 канбан-досок
Asana	+	-	-	-	15 человек
Jira	+	+	-	-	10 человек, ограниченные настройки
Notion	+	-	+	-	10 человек

В рамках данной работы разрабатывается веб-приложение, сочетающее ключевые функции для совместной работы: todo-лист, канбан-доску и текстовые заметки. Основное внимание уделяется простоте интерфейса, удобству использования и доступности. В отличие от существующих решений, создаваемый инструмент не перегружен лишними функциями, а его интуитивный интерфейс позволяет быстро приступить к работе без длительного обучения.

1.2 Определение потребностей пользователей

Совместная работа в цифровом пространстве — это комплекс взаимодействий между людьми, направленных на достижение общих целей, в котором используются различные средства коммуникации и управления. Кооперативная работа над проектами требует четкой организации, эффективного распределения задач и удобных инструментов для коммуникации.

Для классификации инструментов для совместной работы также могут использоваться следующие категории:

- Инструменты для маленьких команд — простые решения, не требующие сложных настроек и высокой степени персонализации (Trello).

- Многофункциональные платформы — комплексные системы, которые включают в себя управление задачами, коммуникацию, аналитические инструменты, интеграцию с внешними сервисами и другие функции (Asana, Jira).

- Инструменты для крупных организаций — решения, ориентированные на автоматизацию и управление крупномасштабными проектами, с продвинутыми возможностями для отчетности, распределения ресурсов и масштабируемости (Microsoft Project).

В последние годы наблюдается тенденция к упрощению процессов управления проектами, особенно в небольших командах, где важнее оперативность и легкость взаимодействия, чем сложные механизмы контроля и разграничения прав.

Популярные платформы предлагают широкий функционал, включая детализированные роли пользователей, настройки доступа и сложные механизмы синхронизации. Однако в небольших группах, стартапах и неформальных командах такие функции часто избыточны и могут усложнять процесс вместо его упрощения.

Разрабатываемое приложение ориентировано на удобство, простоту и интуитивность. Оно должно предоставить базовый, но достаточный набор инструментов для эффективной организации работы без лишней сложности. Для

более глубокого понимания потребностей пользователей рассмотрим ключевые аспекты их деятельности и возникающие проблемы.

Несмотря на широкий выбор инструментов для совместной работы, пользователи часто сталкиваются с рядом проблем [4]:

- Сложность интерфейса. Многие платформы требуют времени на освоение из-за перегруженного интерфейса.

- Лишние функции. Во многих сервисах присутствуют сложные механизмы отчетности, настройки ролей и другие функции, которые не всегда нужны небольшим командам.

- Ограничения бесплатных версий. Популярные сервисы, такие как Trello и Asana, в бесплатных версиях имеют серьезные ограничения.

- Сложность интеграции. Некоторые платформы требуют настройки интеграций с другими сервисами, что усложняет использование.

На основе анализа потребностей пользователей можно сформулировать ключевые требования к разрабатываемому приложению:

1. Возможность совместной работы без распределения ролей.
2. Простая система управления задачами без сложных настроек.
3. Минимальный порог входа.
4. Удобное ведение заметок, связанных с проектом.
5. Доступность на различных устройствах.

Совместная работа без сложных механизмов распределения ролей:

Одной из ключевых потребностей пользователей является удобная и интуитивная система управления задачами. Независимо от типа проекта, его успешное выполнение требует четкого планирования, постановки целей и координации действий команды. Важно, чтобы пользователи могли быстро добавлять и изменять задачи, без необходимости разбираться в сложных настройках или подстраиваться под жесткую систему распределения ролей [5].

В небольших командах и стартапах управление задачами часто осуществляется в динамичном режиме, когда нет строгой иерархии, а участники работают на равных или сразу на нескольких ролях. В таких условиях важна не

сложность механизма, а его удобство и адаптивность. Простые, но эффективные инструменты помогают командам сосредоточиться на работе, а не на освоении системы [6].

Преимущества отказа от сложных настроек доступа:

- Упрощенная работа: пользователи сразу получают полный доступ ко всем возможностям без необходимости настройки прав.

- Гибкость: любой участник может взять на себя любую задачу или внести изменения в проект.

- Отсутствие необходимости администрирования: в небольших командах часто нет отдельного человека, отвечающего за управление правами, поэтому отказ от этих механизмов упрощает процесс.

Управление задачами и контроль выполнения:

Эффективное управление задачами — один из ключевых аспектов организации совместной работы. Это позволяет команде не только распределять рабочую нагрузку, но и четко отслеживать прогресс выполнения проекта. Важно, чтобы система управления задачами была удобной, гибкой и интуитивно понятной, не перегруженной лишними функциями.

В отличие от механизма управления правами и ролями, который регулирует доступ к задачам, сама система управления задачами отвечает за их удобную организацию, отслеживание статусов и поддержку эффективного планирования. Это два независимых аспекта, но оба они важны для комфортной работы команды.

При проектировании удобной системы управления задачами следует учитывать несколько важных аспектов:

- Простота создания и редактирования задач. Пользователи не должны тратить время на сложные настройки — задача должна добавляться в несколько кликов.

- Гибкость организации задач. Разделение на категории, статусы («Ожидание», «В работе», «Завершено») и возможность визуального представления (список или канбан-доска) упрощает контроль за процессом.

– Минимум обязательных полей. В сложных системах пользователи часто вынуждены заполнять множество параметров перед созданием задачи. В простых решениях это не требуется, что делает процесс более быстрым.

Минимальный порог входа и простота освоения:

Чем проще интерфейс, тем быстрее пользователи смогут использовать систему без необходимости долгого изучения инструкций. В современных платформах для совместной работы часто встречаются сложные панели управления, перегруженные настройки и многоуровневые меню. Однако в небольших командах важнее интуитивно понятный интерфейс, который позволяет приступить к работе без предварительного изучения документации.

Простота интерфейса особенно важна в динамичных проектах, где нет времени на обучение и адаптацию. Если пользователь вынужден тратить время на поиск нужных функций или разбираться в сложных настройках, это замедляет процессы и снижает продуктивность команды.

Ключевые принципы простого интерфейса:

– Минимализм. Никаких лишних кнопок, сложных меню и многоуровневых настроек.

– Интуитивность. Все основные функции доступны сразу, без необходимости искать их в скрытых разделах.

– Гибкость интерфейса. Пользователь должен сам выбирать удобный формат работы, например, переключаться между списком задач и канбан-доской.

Ведение текстовых записей и централизованное хранение информации:

В процессе работы над проектами команды сталкиваются с необходимостью фиксировать различную информацию: идеи, обсуждения, технические детали, планы, а также важные уточнения к задачам. Без удобного инструмента для ведения текстовых записей эта информация часто разрозненно хранится в личных заметках участников, в чатах или на сторонних платформах, что приводит к потере данных и усложняет совместную работу.

Централизованное хранение текстовых записей внутри системы совместной работы позволяет всей команде иметь доступ к актуальной информации в одном месте. Это особенно важно в условиях динамичных проектов, где данные могут часто обновляться, а участники должны быть уверены, что работают с последней версией документа [7].

Чтобы инструмент для заметок был действительно полезным, он должен соответствовать нескольким ключевым требованиям:

- Структурированность. Записи должны быть организованы так, чтобы их можно было легко найти.

- Простота редактирования. Встроенный текстовый редактор с базовыми функциями форматирования поможет упорядочить информацию.

- Связь с задачами. Важно, чтобы заметки можно было легко ассоциировать с определенными задачами или этапами проекта.

Хорошо организованная система ведения заметок значительно упрощает работу команды, устраняя хаос в хранении информации. Инструмент должен быть интуитивным, гибким и интегрированным с задачами, чтобы пользователи могли не только фиксировать важные моменты, но и быстро находить нужные данные. Это делает совместную работу более продуктивной и упорядоченной.

Доступность с различных устройств:

Современные пользователи работают не только на компьютерах, но и на мобильных устройствах. Возможность управлять задачами, редактировать заметки и взаимодействовать с коллегами в любое время и в любом месте значительно повышает продуктивность.

Одним из ключевых требований к веб-приложению является его кроссплатформенность, то есть способность адаптироваться под различные экраны и операционные системы без потери функциональности. Пользователь должен иметь одинаково удобный доступ к сервису как с настольного компьютера, так и со смартфона или планшета [8].

Ключевые аспекты кроссплатформенности:

– Адаптивный дизайн - главный принцип кроссплатформенности. Интерфейс должен автоматически подстраиваться под размер экрана, сохраняя при этом удобство использования.

– Упрощенный интерфейс для мобильных пользователей. Некоторые элементы управления могут отличаться на мобильных устройствах, чтобы повысить удобство работы. Если веб-приложение активно использует сложные таблицы или большие списки данных, на мобильных устройствах важно предоставить возможность их удобного просмотра и прокрутки.

– Быстрая загрузка. Это особенно важно в мобильных сетях, где скорость интернета может варьироваться. Легковесный интерфейс, оптимизированные изображения и минимизация фоновых процессов помогут ускорить работу приложения на любых устройствах.

Важно, чтобы мобильная версия не просто сжимала страницу, а предлагала оптимизированный интерфейс. Например, кнопки и поля ввода должны быть достаточно крупными, чтобы ими было удобно пользоваться на сенсорном экране. В мобильной версии может потребоваться скрывать или упрощать некоторые элементы, оставляя только ключевые функции, чтобы избежать перегруженности экрана.

Ключевые особенности системы включают удобное управление задачами, интеграцию с текстовыми заметками и простоту совместной работы без сложных механизмов распределения ролей. Минималистичный и адаптивный интерфейс делает использование приложения комфортным как на компьютерах, так и на мобильных устройствах, обеспечивая доступность в любой ситуации.

Таким образом, разработка данного приложения решает проблему перегруженности функционала, свойственную многим современным платформам, и предлагает компактный, но эффективный инструмент для организации командной работы. Благодаря простоте, гибкости и удобству оно станет надежным помощником для команд, которым важна оперативность, прозрачность процессов и удобный формат взаимодействия.

1.3 Составление требований

Разрабатываемое веб-приложение ориентировано на простоту и удобство использования. Это означает, что требования к нему должны быть тщательно продуманы, чтобы обеспечить баланс между достаточным функционалом и минималистичным интерфейсом. Ключевой задачей является создание интуитивно понятной системы, которая решает основные потребности пользователей без лишней сложности.

Для упорядоченного подхода к проектированию требований их принято разделять на две основные группы:

- Функциональные требования — описывают, что именно должно уметь делать приложение. Они определяют основные возможности системы, такие как работа с задачами, ведение текстовых записей, совместное редактирование и доступ с разных устройств.

- Нефункциональные требования — задают ограничения и характеристики работы системы. Они касаются удобства использования, производительности, безопасности, совместимости и других аспектов, влияющих на пользовательский опыт и надежность приложения.

Определение этих требований на раннем этапе разработки помогает избежать неопределенности, упростить процесс реализации системы и обеспечить соответствие ожиданиям пользователей [9]. Далее будут подробно рассмотрены ключевые функциональные и нефункциональные требования, предъявляемые к веб-приложению.

Функциональные требования:

1. Управление задачами

Одной из ключевых функций приложения является организация задач. Пользователи должны иметь возможность создавать, редактировать и удалять задачи, а также перемещать их между статусами. Приложение должно поддерживать:

- Создание новых задач с возможностью ввода названия и описания. Желательно, чтобы процесс создания занимал минимум времени и не требовал заполнения большого количества полей.

- Редактирование уже существующих задач. Возможность изменения названия, описания и статуса задачи в любое время, чтобы пользователи могли корректировать информацию по мере необходимости.

- Удаление задач. Завершенные или ошибочно созданные задачи должны легко удаляться, обеспечивая порядок в списке.

- Группировка по статусам. Каждая задача должна иметь определенный статус (например, «Ожидание», «В работе», «Завершено»), что позволит пользователям быстро ориентироваться в текущих делах.

- Канбан-доска. Визуальное представление задач в виде карточек, которые можно перетаскивать между статусами, помогает лучше контролировать процесс выполнения работы и планировать дальнейшие действия.

2. Ведение текстовых записей

Помимо задач, пользователи часто нуждаются в возможности фиксировать текстовую информацию, например, идеи, комментарии или планы. В связи с этим в приложении должна быть реализована система текстовых заметок, включающая:

- Возможность создания новых записей.

- Форматирование текста (заголовки, списки, выделение жирным, курсивом).

- Редактирование и удаление записей.

- Связь с задачами. Возможность прикреплять заметки к конкретным задачам или проектам поможет организовать информацию и упростить поиск нужных данных.

- Автоматическое сохранение. Изменения в заметках должны сохраняться в режиме реального времени, чтобы пользователи не теряли данные при закрытии страницы или сбое соединения.

3. Совместный доступ к проекту

Поскольку веб-приложение предназначено для совместной работы, пользователи должны иметь возможность приглашать других участников в проект. Однако в отличие от сложных корпоративных систем с детальной настройкой ролей и прав, здесь реализуется более простой подход: все участники обладают равными возможностями. Это делает взаимодействие в небольших командах более гибким и удобным, исключая необходимость администрирования доступа. Приложение должно позволять:

- Добавлять новых участников в проект. Любой пользователь, имеющий доступ к проекту, должен иметь возможность пригласить других участников по email или через уникальную ссылку.

- Удалять участников. Хотя все пользователи обладают равными правами, администратор проекта (его создатель) должен иметь возможность исключить участников из списка. Это позволит избежать ситуаций, когда доступ остается у неактивных или посторонних пользователей.

- Все пользователи проекта должны иметь равные права на редактирование задач и заметок, что упрощает взаимодействие.

4. Простой механизм авторизации и входа в систему

Для защиты данных пользователей и организации работы с проектами приложение должно поддерживать систему регистрации и авторизации. Однако важно сохранить простоту процесса, чтобы пользователи могли быстро начать работу без необходимости заполнять сложные формы или проходить многоэтапные проверки. Приложение должно поддерживать:

- Регистрацию новых пользователей. При первом входе пользователю необходимо создать учетную запись, указав email и пароль. Опционально можно добавить ввод имени, но основные данные должны оставаться минимальными.

- Авторизацию с проверкой учетных данных. При каждом входе система должна проверять корректность введенного email и пароля, предоставляя доступ к проектам пользователя.

– Восстановление пароля через email. В случае утери пароля должен быть реализован механизм восстановления, при котором пользователю отправляется письмо со ссылкой или кодом для сброса пароля.

5. Просмотр и управление проектами

Каждый пользователь может создавать собственные проекты и управлять ими. Это позволяет группировать задачи и заметки по различным направлениям работы, сохраняя структуру данных. Функции проекта включают:

– Создание нового проекта. Пользователь может создать проект, задав его название и, при необходимости, краткое описание.

– Редактирование названия и описания проекта.

– Удаление проекта. Только создатель проекта должен иметь право удалить его. Это предотвратит случайное удаление данных другими участниками.

– Просмотр списка доступных проектов. Пользователь должен видеть все проекты, в которых он участвует, с возможностью быстрого перехода к нужному.

Нефункциональные требования:

1. Простота и удобство интерфейса

Приложение ориентировано на минимализм и легкость использования. Также приложение ориентировано на небольшие команды, его интерфейс должен быть интуитивно понятным и доступным даже для пользователей без опыта работы с подобными системами. Упрощение интерфейса помогает быстрее освоить функционал и минимизировать время на обучение [10].

Основные требования к интерфейсу:

– Минимальное количество действий для выполнения основных операций. Создание задач, заметок и управление ими должны осуществляться в один-два клика.

– Минимум обязательных полей. Пользователь должен иметь возможность добавлять задачи и заметки без необходимости заполнять множество параметров. Например, для задачи необязательно заполнять срок

выполнения, ответственного, описание и другие поля. Чтобы зафиксировать цель нужно только название, отражающее ее суть.

- Отсутствие сложных настроек, которые могут затруднить работу пользователя. Приложение должно быть готово к использованию сразу после регистрации, без необходимости дополнительной конфигурации.

- Четкая визуальная структура. Важные элементы интерфейса (список задач, канбан-доска, текстовые записи) должны быть логично организованы, а второстепенные детали не должны отвлекать от работы.

2. Производительность и быстродействие

Для комфортной работы пользователей система должна быть отзывчивой и обеспечивать быструю загрузку интерфейса. Скорость работы приложения напрямую влияет на его удобство. Долгая загрузка страниц и задержки при обновлении задач могут негативно сказаться на продуктивности пользователей. Поэтому необходимо обеспечить стабильную и быструю работу системы. Основные требования:

- Загрузка страниц и данных должна происходить менее чем за 2 секунды (время рендера страницы вместе с загрузкой необходимых ресурсов со средней скоростью интернета).

- Взаимодействие с задачами (создание, редактирование, перемещение) должно выполняться без значительных задержек.

- Оптимизированное потребление ресурсов. Приложение не должно чрезмерно загружать процессор и оперативную память, особенно при работе в браузере на мобильных устройствах.

3. Кроссплатформенность

Пользователи могут работать с приложением как на настольных компьютерах, так и на мобильных устройствах, поэтому важно обеспечить корректное отображение интерфейса на разных экранах. Для этого требуется:

- Адаптивный дизайн. Интерфейс должен динамически подстраиваться под размеры экрана, чтобы пользователям было удобно работать с любого устройства.

- Оптимизация под мобильные браузеры. На сенсорных экранах элементы управления должны быть удобны для нажатия, а текст и кнопки — хорошо читаемы.

- Корректное отображение всех функций. Вне зависимости от устройства пользователь должен иметь доступ ко всему функционалу приложения без ограничений.

4. Надежность и отказоустойчивость

Стабильная работа системы — важный аспект, особенно в условиях командной работы. Потеря данных из-за сбоев или ошибок может значительно снизить доверие к приложению. Требования к надежности:

- Автоматическое сохранение данных. Все изменения в задачах и заметках должны сохраняться в реальном времени, чтобы предотвратить потерю информации.

- Обработка ошибок с информативными сообщениями. При возникновении проблемы пользователь должен получать понятное уведомление с рекомендациями по ее устранению.

- Минимизация простоев. Важно, чтобы приложение продолжало работать даже при высокой нагрузке или кратковременных сбоях сервера.

5. Безопасность данных

Хотя приложение не использует сложные механизмы прав доступа, оно должно обеспечивать базовые меры защиты данных. Основные меры безопасности:

- Шифрование передаваемых данных. Вся информация должна передаваться через защищенное соединение (HTTPS).

- Хранение паролей в зашифрованном виде. Должны использоваться современные методы хеширования. В приоритете механизм сохранения хеша, а не самого пароля. Это обеспечивает максимальную защиту [11].

- Защита от подбора паролей. Ограничение количества попыток входа предотвратит атаки методом перебора.

– Резервное копирование данных. Регулярное создание резервных копий поможет восстановить информацию в случае сбоя.

При разработке программного продукта важно четко определить его функциональные и нефункциональные требования. Они задают границы системы, помогают структурировать процесс разработки и служат ориентиром для оценки качества конечного продукта. Без четко сформулированных требований существует риск отклонения от целей проекта, усложнения интерфейса и внедрения избыточных функций, которые могут усложнить работу пользователей.

Разрабатываемое веб-приложение должно предоставлять пользователям удобные инструменты для управления задачами и заметками, сохраняя при этом простоту интерфейса и легкость освоения.

Функциональные требования определяют, какие именно возможности будут доступны пользователям: работа с задачами, текстовыми записями, совместное редактирование и управление проектами. Нефункциональные требования задают параметры работы системы, такие как удобство интерфейса, производительность, безопасность и отказоустойчивость.

Сбалансированное сочетание этих требований обеспечит удобство работы пользователей и позволит создать легкое, но функциональное решение для совместной работы над проектами.

Проблема организации совместной работы над проектами рассмотрена в текущей главе. Но для реализации задумки требуется провести анализ существующих практик и технологий реализации веб-приложений. А для подтверждения востребованности разрабатываемого программного продукта важно изучить реальный пример командной разработки. Это позволит выявить закономерности, тенденции развития и оценить эффективность применения подобного сервиса для повышения эффективности и удобства работы в небольших группах.

2. Теоретическая часть

2.1 Основы проектирования веб приложений

Проектирование веб-приложений — это один из важнейших этапов создания программных решений, включающий определение объёмов, методов и средств реализации задачи, а также описание выбранных методов сбора и обработки данных. Оно охватывает как технические, так и организационные аспекты, обеспечивая соответствие будущей информационной системы требованиям пользователей, возможностям инфраструктуры и стандартам качества.

Первым шагом проектирования является анализ предметной области и структурно-функциональный анализ процессов, которые необходимо автоматизировать. В рамках разрабатываемого веб-приложения для совместной работы над проектами были выявлены ключевые бизнес-процессы: планирование задач, организация командной работы, отслеживание выполнения и ведение рабочих записей. Эти процессы формируют основу логики приложения, обеспечивая создание todo-листов, kanban-досок и текстовых заметок с равными правами пользователей.

Следующим важным шагом является структурно-функциональный анализ. Он позволяет наглядно представить, какие данные циркулируют в системе, как они связаны между собой и какие операции с ними выполняются. В рамках этого анализа проектируются основные компоненты информационной системы, их функции, а также схемы обмена данными. На этом этапе формируются требования к объёмам хранимой информации, скорости обработки запросов и стабильности работы системы.

Структурно-функциональное проектирование веб-приложений основывается на нескольких ключевых принципах:

- Модульность — разделение приложения на независимые компоненты, которые могут быть разработаны, протестированы и обновлены отдельно.

– Масштабируемость – возможность системы эффективно работать при увеличении нагрузки.

– Производительность – оптимизация скорости загрузки страниц и обработки данных.

– Юзабилити (удобство использования) – интуитивно понятный интерфейс и доступность функционала.

– Безопасность – защита данных пользователей и предотвращение атак.

– Кроссплатформенность – корректная работа приложения на различных устройствах и браузерах.

Процесс проектирования веб-приложения можно разделить на несколько этапов [12]:

– Анализ требований – определение целей приложения, изучение целевой аудитории и исследование конкурентов.

– Разработка архитектуры – выбор клиент-серверной модели, распределение функций между фронтендом и бэкендом.

– Проектирование интерфейса – создание макетов и прототипов, определение логики взаимодействия пользователей с системой.

– Выбор технологий – подбор стеков технологий для серверной и клиентской частей.

– Разработка и тестирование – реализация функционала, тестирование и отладка.

– Развертывание и поддержка – публикация веб-приложения в продакшен-среде и обеспечение его стабильной работы.

На этапе проектирования проводится формирование структуры будущей системы и выбор подходящей архитектуры. Современные веб-приложения строятся по клиент-серверной модели, которая позволяет разделить обязанности между серверной частью (backend) и клиентской (frontend). Такая архитектура повышает гибкость, производительность и масштабируемость системы.

При проектировании веб-приложений применяются различные архитектурные шаблоны:

– Монолитная архитектура – весь код приложения объединен в один проект.

– Микросервисная архитектура – приложение разделено на независимые сервисы, взаимодействующие через API.

– MVC (Model-View-Controller) – разделение данных, представления и логики управления.

– SPA (Single Page Application) – одностраничное приложение, где взаимодействие с сервером минимизировано.

Особое внимание уделяется обеспечению отказоустойчивости, надёжности и расширяемости системы. При проектировании закладываются возможности для масштабирования и модернизации, что позволяет адаптировать приложение к растущим нагрузкам или новым требованиям без полной переработки архитектуры.

Также в рамках проектирования важно определить методы сбора и обработки информации. Для этого планируются механизмы регистрации, хранения и передачи данных между различными компонентами системы. Разрабатываются стандарты и форматы данных, обеспечивающие корректную работу всех модулей приложения.

Проектирование веб-приложения требует комплексного подхода, включающего анализ требований, выбор архитектуры, проработку интерфейса и обеспечение безопасности. Следование принципам и этапам разработки позволяет создать удобное, надежное и эффективное веб-приложение.

2.2 Архитектура веб приложений

Архитектура веб-приложений — это совокупность структурных решений, определяющих взаимодействие между различными компонентами системы, распределение логики и данных, а также способы обмена информацией между клиентом и сервером. Грамотно выбранная архитектура играет ключевую роль в стабильности, масштабируемости, безопасности и удобстве поддержки приложения.

Современные веб-приложения, как правило, строятся по клиент-серверной модели (Рисунок 1), где клиент отправляет запросы к серверу, а сервер обрабатывает эти запросы, взаимодействует с базой данных и возвращает ответы[13]. Основу такой архитектуры составляют три основных слоя: клиентский интерфейс (frontend), серверная логика (backend) и хранилище данных (база данных).

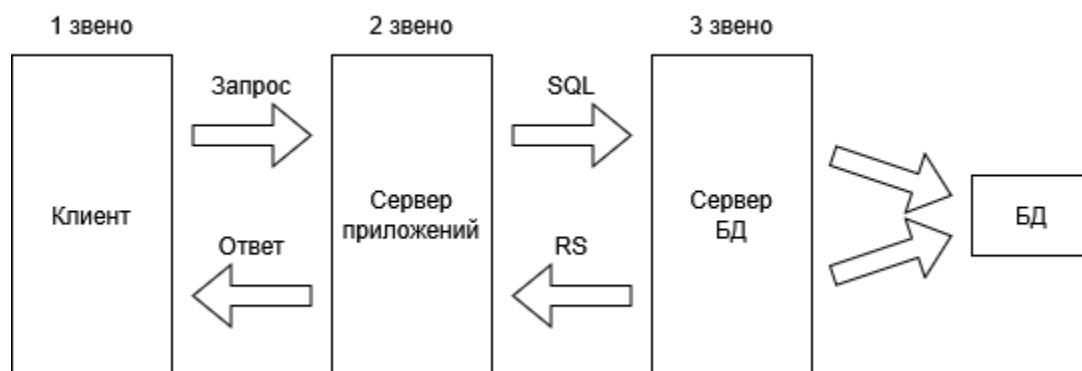


Рисунок 1 - Клиент-серверная модель

На клиентском уровне реализуется интерфейс взаимодействия пользователя с приложением. Этот уровень отвечает за отображение информации, сбор пользовательского ввода и отправку данных на сервер. Для создания интерфейса чаще всего используются HTML, CSS и JavaScript, а также популярные фреймворки, такие как React, Angular или Vue.js. Современные интерфейсы поддерживают динамическое обновление данных и взаимодействие с сервером через API без полной перезагрузки страницы.

Серверная часть отвечает за обработку логики приложения: получение запросов от клиента, выполнение бизнес-операций, обращение к базе данных и формирование ответа. Сервер может быть реализован на различных языках программирования и платформах, таких как Python (Django, Flask), JavaScript (Node.js, Express), Java (Spring), PHP и других. Сервер также обеспечивает

маршрутизацию запросов, управление сессиями пользователей и реализацию механизмов авторизации и аутентификации².

База данных служит для хранения всей информации, необходимой для работы приложения. Это могут быть как реляционные СУБД (PostgreSQL, MySQL), так и нереляционные (MongoDB, Redis). Выбор конкретной системы зависит от требований к структуре данных, скорости обработки и масштаба приложения.

Современная модель построения веб приложений выделяет 2 сервера: приложение и база данных. В свою очередь приложение может быть разделено на 2 отдельных: клиентское и серверное. Обычно выбирается REST архитектура для взаимодействия всех приложений. Такая схема может быть сложной в настройке и обеспечения безопасности, так как сервера должны посылать защищенные запросы на всех этапах работы. Одно из решений этой проблемы – контейнеризация Docker. Виртуализация позволяет запускать изолированные песочницы с настроенными приложениями. Все приложения работают на одном сервере и используют локальную сеть для общения³.

Среди архитектурных подходов к построению веб-приложений можно выделить следующие:

- Монолитная архитектура — всё приложение реализовано как единое целое. Такой подход прост в разработке и развёртывании, но с ростом функциональности усложняется масштабирование и сопровождение.

- Микросервисная архитектура — приложение состоит из множества независимых сервисов, каждый из которых отвечает за определённую функциональность. Это повышает гибкость и масштабируемость, но требует более сложной инфраструктуры и управления.

² Введение в серверную часть // MDN WEB DOCS URL: https://developer.mozilla.org/ru/docs/Learn_web_development/Extensions/Server-side/First_steps/Introduction (дата обращения: 23.01.2025)

³ Контейнерные микрослужбы // Microsoft URL: <https://learn.microsoft.com/ru-ru/dotnet/architecture/maui/micro-services> (дата обращения: 15.01.2025)

– Многоуровневая архитектура — приложение делится на уровни (например, представление, бизнес-логика, доступ к данным), что повышает модульность и упрощает поддержку.

Было принято решение об использовании микросервисной архитектуры, а не монолитной. У такого подхода есть множество преимуществ, среди которых независимость компонентов друг от друга, масштабируемость и легкость разработки, так как нет одного большого и сложного приложения [16]. На схеме (Рисунок 2) наглядно показано, как можно разделить различные слои сервиса. К тому же масштабирование микросервисного решения легче в реализации и поддержке. Многоуровневая архитектура – не лучшая архитектура для этого проекта, так как не предоставляет полной модульности и сложнее в разработке.

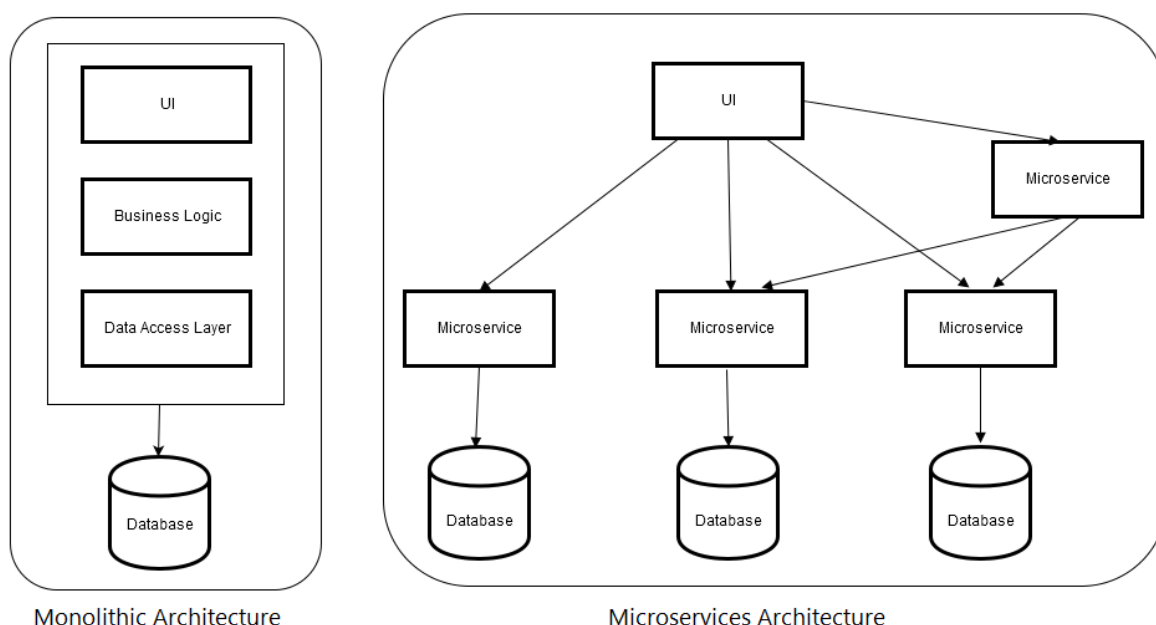


Рисунок 2 - Монолитная и микросервисная архитектуры

Также следует упомянуть шаблоны взаимодействия клиента и сервера:

– Server-side rendering (SSR) — вся логика обработки запроса и генерации HTML происходит на сервере. Такой подход обеспечивает быструю загрузку и хорошую индексацию поисковыми системами.

– Client-side rendering (CSR) — логика отображения интерфейса полностью реализована на клиенте. Используется для динамичных приложений, снижает нагрузку на сервер.

– Isomorphic/Universal rendering — совмещает оба подхода: начальная отрисовка происходит на сервере, а последующая работа — на клиенте. Часто используется в SPA для улучшения производительности и SEO.

Для разгрузки сервера и упрощения разработки выбран CSR. Этот подход не требует предварительной генерации страниц HTML. Вся работа по отрисовке и обработке информации происходит в браузере на клиентском устройстве. При правильном написании кода это не вызывает проблем с производительностью ни на сервере, ни у пользователя. В будущем возможен переход на изоморфный рендеринг для улучшения SEO, так как поисковые роботы при заходе на страницы не отрисовывают их. Для работы CSR необходимо выполнить JS код. У роботов по умолчанию он выключен для безопасности. При генерации страниц на сервере пользователь получает готовый или шаблонный документ, который можно индексировать[17].

В современных веб-приложениях часто применяется REST или GraphQL API для взаимодействия между клиентом и сервером. Это позволяет отделить логику отображения от логики обработки данных, облегчая масштабирование и повторное использование компонентов.

Выбранная архитектура представлена на рисунке 3. Каждое приложение обернуто в контейнер Docker, что позволяет легко масштабировать решение.

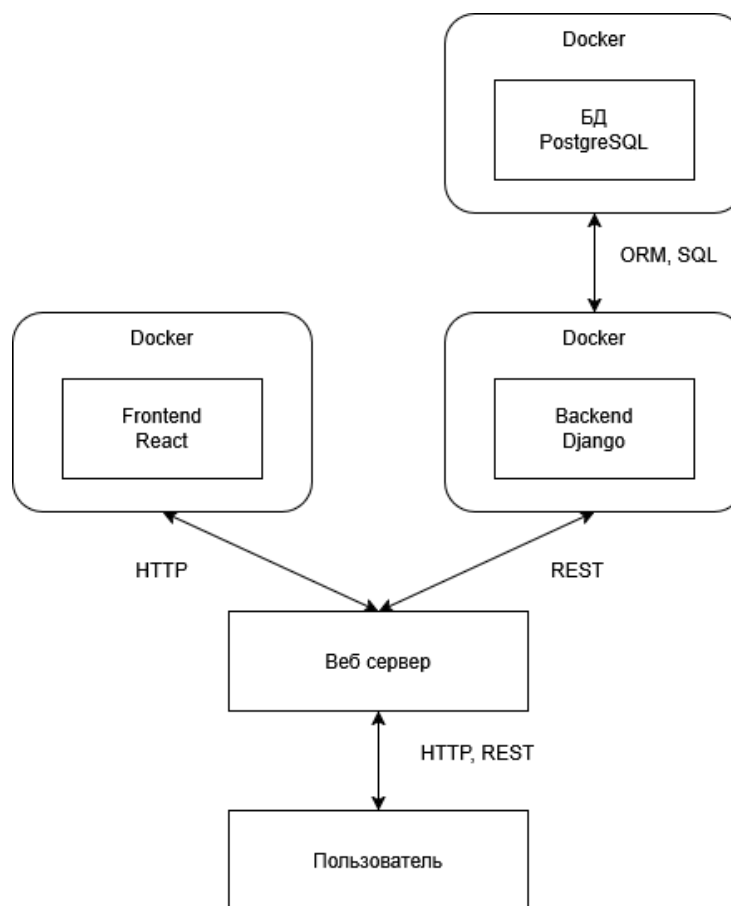


Рисунок 3 – Выбранная архитектура

Архитектура веб-приложения должна быть выбрана исходя из конкретных требований проекта: планируемой нагрузки, количества пользователей, частоты обновлений данных, особенностей бизнес-логики и предпочтений команды разработки. Правильно спроектированная архитектура — залог успешной реализации и долгосрочной поддержки веб-приложения.

2.3 Обоснование выбранных технологий

При разработке веб-приложения для организации совместной работы над проектами был произведён тщательный выбор стека технологий, способного обеспечить надёжность, масштабируемость, производительность и простоту сопровождения. В результате были выбраны следующие компоненты: серверный фреймворк Django, клиентская библиотека React, реляционная система управления базами данных PostgreSQL и система контейнеризации Docker. В совокупности они позволяют построить современное, отказоустойчивое и масштабируемое веб-приложение.

В таблице 3 показано сравнение фреймворков для разработки серверной части. Из нее можно сделать вывод, что Django больше всего подходит реализации.

Таблица 3 – Сравнение серверных фреймворков

	Django	Spring	Express	Flask
Язык	Python	Java	JavaScript	Python
Концепция	MVT (MVC)	MVC	MVC	MVT (MVC)
Совместимость с СУБД	PostgreSQL MariaDB MySQL Oracle SQLite	PostgreSQL MySQL Oracle SQL Server MongoDB Couchbase Redis	PostgreSQL MySQL SQLite SQL Server Oracle	PostgreSQL MySQL SQLite Oracle Redis
Тип	Веб-приложение	Веб-приложение Микросервисы	Веб-приложение API	Веб-приложение API
Сложность	Средне	Сложно	Легко	Легко
Особенности	ORM Админ-панель Миграции Аутентификация Безопасность	Аутентификация Аннотации	Настройка вручную Зависимости	Базовые шаблоны

Серверная часть: Django

Django — это мощный и зрелый фреймворк для языка программирования Python, который идеально подходит для разработки сложных веб-приложений. Одним из его ключевых преимуществ является наличие встроенного ORM (Object-Relational Mapping), позволяющего разрабатывать логику взаимодействия с базой данных на уровне объектов, без написания SQL-запросов. Это ускоряет разработку и снижает вероятность ошибок.

Фреймворк следует архитектуре MTV (Model-Template-View), обеспечивая разделение ответственности между моделями, шаблонами и представлениями. В рамках данной работы шаблоны Django не используются.

напрямую, так как интерфейс реализован с помощью React, однако вся бизнес-логика и обработка запросов реализованы средствами Django.

Кроме того, Django предоставляет ряд встроенных механизмов безопасности: защиту от подделки межсайтовых запросов (CSRF), межсайтовых скриптовых атак (XSS), SQL-инъекций, и другие. Это особенно важно для многопользовательских приложений, где безопасность данных имеет первостепенное значение⁴.

В таблице 4 представлено сравнение фреймворков для разработки клиентской части. Был сделан выбор в пользу React.

Таблица 4 – Сравнение клиентских фреймворков

	Vue	React	Angular
Производительность	Высокая	Высокая	Высокая
Гибкость	Высокая	Высокая	Низкая
Поддержка	Сообщество	Facebook	Google
Рендеринг	VDOM	VDOM	RDOM
Масштабируемость	Для небольших проектов	Для средних и больших проектов	Для крупных проектов

Клиентская часть: React

React — это популярная JavaScript-библиотека, разработанная компанией Facebook, и предназначенная для создания быстрых и интерактивных пользовательских интерфейсов. Принцип построения интерфейса в React основан на компонентном подходе, при котором каждый элемент интерфейса является изолированным компонентом, обладающим собственным состоянием и жизненным циклом⁵.

Использование React позволяет реализовать SPA (Single Page Application) — одностраничное приложение, в котором навигация и взаимодействие с сервером происходят без перезагрузки страницы. Это делает интерфейс более

⁴ Django documentation // Django URL: <https://www.djangoproject.com/> (дата обращения: 20.11.2024).

⁵ React documentation // React URL: <https://react.dev/learn> (дата обращения: 27.11.2024).

отзывчивым и приближённым к ощущениям от работы с нативным приложением.

React активно поддерживается сообществом, имеет множество дополнительных библиотек и инструментов, упрощающих разработку и тестирование компонентов. Благодаря модульности и широкому распространению, React является одним из самых эффективных решений для реализации современного клиентского интерфейса.

База данных: PostgreSQL

В качестве хранилища данных используется PostgreSQL — мощная и надёжная объектно-реляционная СУБД с открытым исходным кодом. Одним из её преимуществ является поддержка сложных типов данных, транзакций, триггеров, полнотекстового поиска, индексов и расширений, таких как PostGIS⁶. Все эти возможности позволяют гибко управлять структурой данных и обеспечивать их целостность.

PostgreSQL широко применяется в производственных системах и считается одной из наиболее надёжных и масштабируемых СУБД. Её использование оправдано в проектах, где важна стабильность, соответствие ACID-принципам, надёжность при больших нагрузках и открытая архитектура.

Контейнеризация: Docker

Docker используется для изоляции и управления средами разработки и продакшена. С его помощью каждая часть приложения (бэкенд, фронтенд, база данных) запускается в собственном контейнере, что позволяет избежать проблем совместимости и облегчает перенос приложения между различными средами.

Система контейнеров Docker позволяет создать воспроизводимую инфраструктуру, что особенно ценно при командной разработке и развертывании на серверах. Все зависимости, версии библиотек и параметры

⁶ Documentation // PostgreSQL: Documentation URL: <https://www.postgresql.org/docs/> (дата обращения: 06.12.2024).

конфигурации задаются в Dockerfile и docker-compose.yml, что гарантирует одинаковое поведение приложения в любой среде.

Дополнительно Docker упрощает масштабирование приложения. При необходимости можно быстро поднять дополнительные контейнеры, например, при увеличении количества пользователей, или разделить сервисы на микросервисы в будущем⁷.

Взаимодействие компонентов

Фронтенд и бэкенд приложения взаимодействуют через REST API, который реализован средствами Django REST Framework. Этот подход позволяет надёжно обмениваться данными между клиентом и сервером в формате JSON. Благодаря этому интерфейс может быть полностью независим от серверной реализации, а сервер может быть повторно использован, например, в мобильном приложении. Безопасность обеспечивается использованием защищенного протокола HTTPS [22].

Контейнеризация позволяет организовать сетевое взаимодействие между компонентами приложения в изолированной среде, при этом сохраняя гибкость настройки и лёгкость масштабирования. Один раз настроенный контейнер может использоваться на любом устройстве с любой операционной системой. Приложения в них прослушивают порты для обработки запросов. Docker позволяет настроить их проксирование в глобальную систему на локальный хост или публичный ip адрес. Это позволяет настроить общение контейнеров между собой, не выходя в общий интернет.

Основные преимущества выбранных технологий и архитектурных решений перед остальными:

1. Клиент-серверная модель – разделение логики
2. CSR (клиентский рендеринг) – позволяет разгрузить сервер и сделать динамическое приложение. Страницы с сервера загружаются быстрее

⁷ Guides // Docker Docs URL: <https://docs.docker.com/guides/> (дата обращения: 30.12.2024).

3. SPA (одностраничное приложение) – позволяет работать с одной страницей, не теряя время на переходы по ним. Также браузеру клиента не нужно отрисовывать всю страницу заново, а только некоторые элементы

4. Микросервисная архитектура – разделение приложения на независимые сервисы, общающиеся между собой. Простая поддержка и масштабируемость

5. Использование REST API – единый и легкий формат обмена сообщениями между сервисами.

6. Контейнеризация – изоляция приложений для безопасности и возможность масштабирования

Выбранный стек технологий обеспечивает надёжность, гибкость и масштабируемость разрабатываемого приложения. Django отвечает за обработку данных, бизнес-логику и безопасность, React обеспечивает удобный и динамичный пользовательский интерфейс, PostgreSQL — надёжное хранение данных, а Docker — удобное развертывание и сопровождение. Совместно они образуют устойчивую архитектуру, способную удовлетворить потребности пользователей в рамках проекта по организации совместной работы.

2.4 Обеспечение безопасности данных

В современном веб-приложении безопасность данных является критически важным аспектом, особенно в условиях постоянных угроз со стороны злоумышленников. Надёжная защита информации включает как организационные, так и технические меры, направленные на предотвращение утечки, потери или несанкционированного доступа к данным пользователей. В процессе разработки веб-приложения для совместной работы над проектами были реализованы основные принципы информационной безопасности на нескольких уровнях: сетевом, серверном и клиентском.

С точки зрения определения объёмов, методов и средств обеспечения безопасности, проект охватывает все ключевые бизнес-процессы, подлежащие автоматизации, такие как регистрация пользователей, авторизация, передача и хранение конфиденциальных данных, а также организация взаимодействия

между клиентом и сервером. Были выбраны наиболее надёжные и проверенные технологии и подходы, обеспечивающие необходимый уровень защиты в условиях современных угроз

Передача данных по HTTPS

Для обеспечения безопасности передаваемой информации между клиентом и сервером используется защищённый протокол HTTPS. Он основан на протоколе TLS и обеспечивает шифрование всех запросов и ответов, тем самым предотвращая возможность перехвата данных третьими лицами (например, паролей или содержимого запросов). Для реализации HTTPS был использован бесплатный сертификат от организации Let's Encrypt, который автоматически обновляется с помощью инструмента Certbot.

Использование HTTPS гарантирует:

- конфиденциальность данных;
- защиту от атак типа «Man-in-the-Middle»;
- доверие со стороны браузеров и пользователей (отсутствие предупреждений о небезопасном соединении).

Аутентификация и хранение пользовательских данных

Для управления пользователями и их авторизацией используется встроенная система аутентификации Django. Один из её важных элементов — безопасное хранение паролей. Пароли пользователей никогда не сохраняются в базе данных в открытом виде. Вместо этого они проходят через криптографическую хеш-функцию, с применением солью и алгоритма PBKDF2 (по умолчанию), что делает невозможным восстановление пароля даже при компрометации базы данных [23].

Дополнительные меры защиты включают:

- обязательную проверку сложности пароля;
- возможность включения механизма ограничения количества неудачных попыток входа (через сторонние библиотеки, например, django-axes);
- использование безопасных токенов для восстановления пароля и подтверждения регистрации.

Эти методы обеспечивают надёжную защиту учётных данных пользователей и предотвращают их компрометацию.

Защита от атак на уровне веб-приложения

Django предоставляет встроенные механизмы защиты от наиболее распространённых уязвимостей веб-приложений:

- CSRF (Cross-Site Request Forgery) — защита от подделки межсайтовых запросов реализуется с помощью специальных токенов, которые автоматически встраиваются в формы и проверяются на сервере.

- XSS (Cross-Site Scripting) — при отображении данных, введённых пользователем, Django автоматически экранирует HTML-символы, что предотвращает внедрение вредоносного кода.

- SQL-инъекции — ORM Django изначально защищает от данной уязвимости, так как все SQL-запросы формируются безопасным образом, без прямой вставки пользовательского ввода.

Эти механизмы минимизируют риски уязвимостей в веб-интерфейсе и серверной части приложения.

Изоляция и безопасность окружения (Docker)

Контейнеризация с использованием Docker дополнительно усиливает безопасность системы за счёт изоляции компонентов приложения. Каждое приложение — будь то веб-сервер, база данных или frontend — работает в своём собственном контейнере с ограниченными правами доступа. Это снижает риски распространения атак между сервисами при компрометации одного из них⁸.

Также в Docker предусмотрены механизмы ограничения ресурсов, контроль сетевого взаимодействия между контейнерами и использование образов только из доверенных источников. Все Docker-образы приложения настраиваются с минимально необходимыми зависимостями и правами, что уменьшает площадь возможных атак.

⁸ Securing Docker: Best practices for robust container security // SolDevelo URL: <https://soldevelo.com/blog/securing-docker-best-practices-for-robust-container-security/> (дата обращения: 15.02.2025).

Для минимизации площади атаки используются максимально облегчённые образы с минимально необходимыми зависимостями.

Регулярные обновления и аудит безопасности

Безопасность — это не разовая мера, а непрерывный процесс. В рамках разработки и поддержки приложения важно регулярно обновлять зависимости, в том числе библиотеки Python, JavaScript и образы Docker, чтобы исключить уязвимости, обнаруженные в новых версиях.

Также следует периодически проводить аудит конфигураций и логов, использовать средства статического анализа кода и при необходимости проводить тестирование на проникновение (penetration testing).

Экономическая, техническая и социальная эффективность

С точки зрения экономической эффективности, использование бесплатных решений (Let's Encrypt, open-source библиотек и Docker) позволяет минимизировать затраты на обеспечение безопасности при сохранении высокого уровня защиты.

В техническом плане внедрение данных мер повышает надёжность работы приложения, снижает риски простоев и утечек информации. Изоляция компонентов упрощает администрирование и обновление системы.

Социальная эффективность выражается в росте доверия пользователей к приложению благодаря прозрачной политике безопасности, использованию современных технологий защиты данных и соблюдению актуальных стандартов информационной безопасности.

Обеспечение безопасности в веб-приложении достигается за счёт использования HTTPS, безопасной аутентификации и хранения паролей, встроенных средств защиты Django, а также изоляции компонентов с помощью Docker. Эти меры в совокупности позволяют защитить данные пользователей от несанкционированного доступа и сохранить целостность системы при работе в сети Интернет.

3. Практическая часть

3.1 Разработка прототипа

Разработка прототипа веб-приложения для совместной работы над проектами стала подготовительным этапом, позволившим смоделировать пользовательский интерфейс, проверить сценарии взаимодействия и определить требования к будущей системе. Прототип не рассматривался как самоцель дипломной работы, а выступал рабочим инструментом, помогающим выстроить архитектуру приложения и снизить риски, связанные с разработкой.

В процессе создания прототипа применялись итерационные методы проектирования: первоначально разрабатывались наброски интерфейса, затем они постепенно уточнялись на основе анализа бизнес-процессов и предполагаемых действий пользователя. Были охвачены основные функции: todo-лист, канбан-доска, текстовые заметки, а также навигация между разделами.

Особое внимание уделялось удобству использования интерфейса: анализировалась структура экранов, логика переходов, минимизация числа действий для выполнения типичных задач. Были созданы черновые макеты, отражающие расположение элементов, цветовые акценты, типографику и базовые сценарии.

Макеты создавались в бесплатном сервисе Pixso. При разработке клиентской части настройки стилей для элементов брались именно из этой программы. Pixso — это облачный сервис для совместной работы над дизайном интерфейсов, который позволяет создавать макеты, прототипы и графику в режиме реального времени. Он поддерживает командную работу, комментарии, обмен ресурсами и интеграции, обеспечивая удобное взаимодействие дизайнеров, разработчиков и заказчиков на всех этапах проекта.

Форма авторизации (Рисунок 4) содержит минимальное необходимое количество элементов и трансформаций. Всего реализовано 3 формы: авторизация/регистрация, подтверждение кодом из почты и восстановление

пароля. Для входа в свой аккаунт и регистрации на сервисе используется комбинированная форма, чтобы уменьшить путь пользователя до главной страницы сайта. Программная реализация представлена в приложении А.

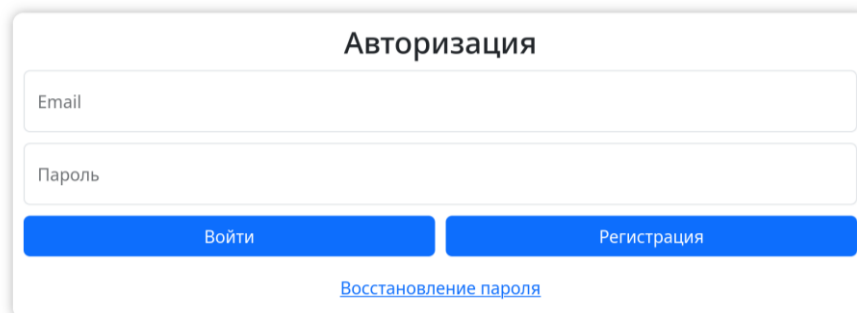
The image shows a login and registration form titled "Авторизация" (Authorization). It contains two input fields: "Email" and "Пароль" (Password). Below the fields are two blue buttons: "Войти" (Login) and "Регистрация" (Registration). At the bottom, there is a blue link labeled "Восстановление пароля" (Reset password).

Рисунок 4 - Форма авторизации/регистрации

Главная страница сайта (Рисунок 5) состоит из двух элементов: навигации и рабочего пространства. Первый постоянен, а второй зависит от текущего состояния. Все части сервиса, такие как канбан-доска или записки, открываются по клику в навигации и отображаются во втором окне.

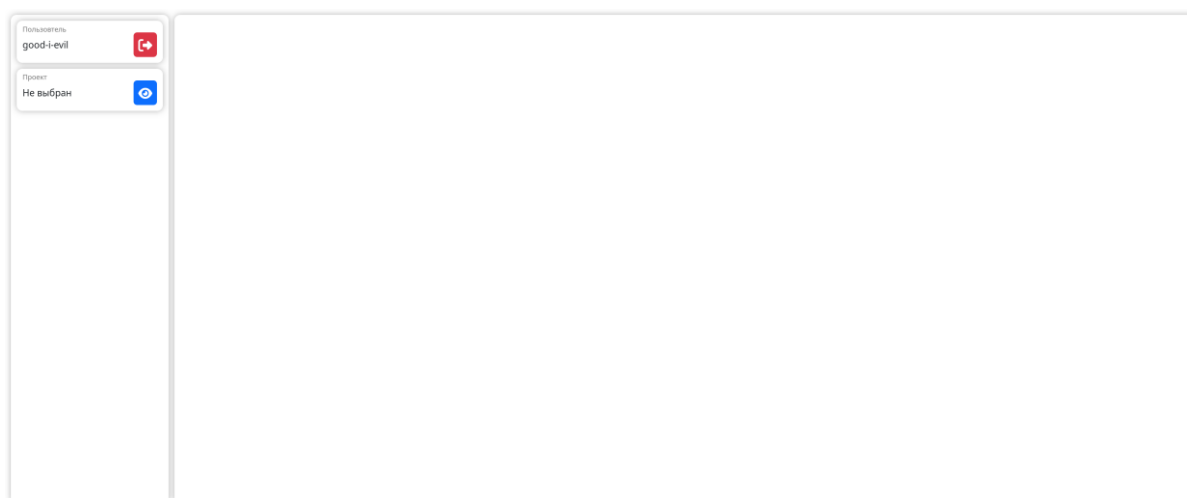


Рисунок 5 - Главная страница сайта

Элементы навигации выполнены в виде блоков, состоящих всего из 2 элементов: название и кнопка открытия/закрытия (Рисунок 6). При открытии не должно происходить переходов на другие страницы и невозможен сброс текущего состояния. Так пользователь не сможет потеряться. Пример данного компонента представлен в приложении Б.



Рисунок 6 - Элемент навигации

Далее будут показаны блоки для решения основных бизнес требований. Пользователь может менять проект, над которым будет работать. Это делается в окне выбора проекта (Рисунок 7). В нем видно название и описание проекта, кнопки управления. Удаление и смена работают без дополнительных действий, а для редактирование открывается следующее окно с атрибутами объекта (Рисунок 8).

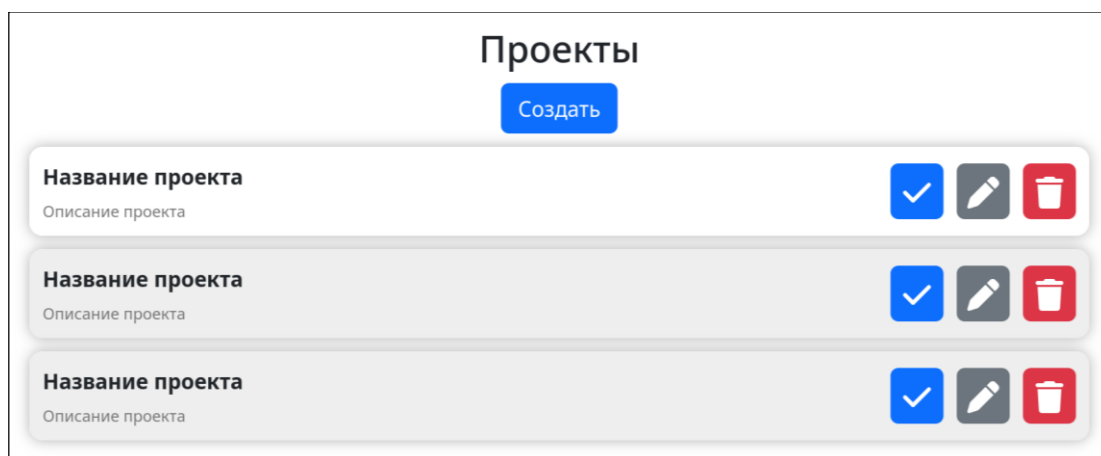


Рисунок 7 - Окно выбора активного проекта

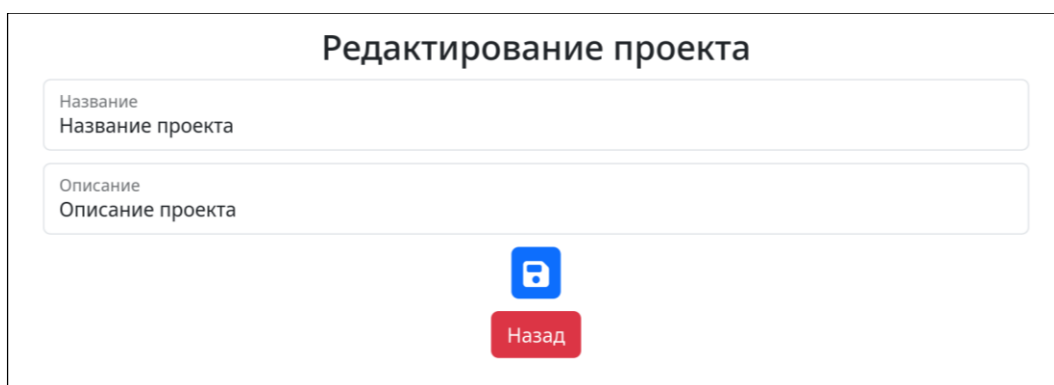


Рисунок 8 - Окно редактирования проекта

Пользователи могут создавать задачи с разными статусами, они отображаются в списке задач (Рисунок 9). Чтобы не занимать место, прогресс отрисовывается цветом, а не текстом. Для новых красный, в прогрессе – желтый,

выполненных – зеленый. Любую задачу можно скрыть с канбан-доски или вернуть обратно.

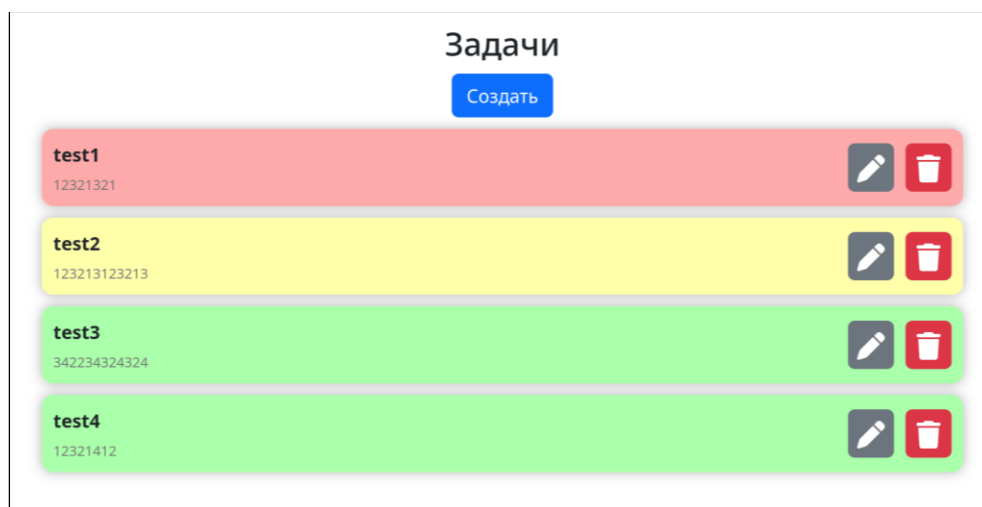


Рисунок 9 - Окно просмотра задач

На основе объекта «Задача» строится канбан-доска (Рисунок 10). Цвета элементов соответствуют статусам, описанным ранее. К каждой задаче есть только самые нужные элементы управления, а именно перемещение и скрывание.

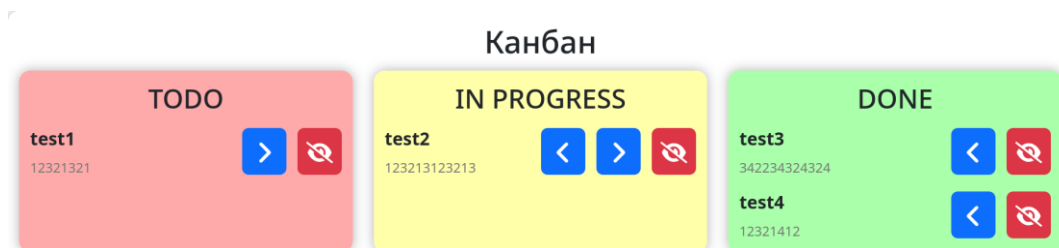


Рисунок 10 - Канбан-доска

Управление командой происходит в соответствующем окне (Рисунок 11). Несмотря на то, что в сервисе не предусмотрено управление ролями, пользователи в проекте идентифицируются как создатель и администратор. Разница прав только в одной функции: удаление проекта целиком.

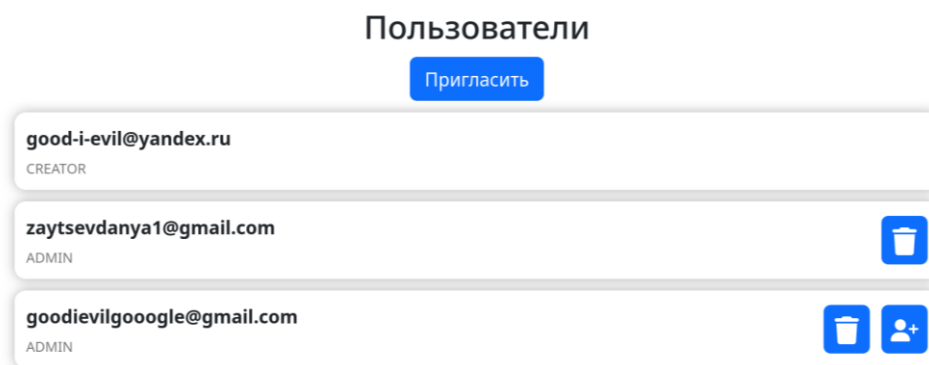


Рисунок 11 - Окно управления командой

И последний основной элемент – редактирование записок (Рисунок 12). Так как используется блочная система, на экране пользователь видит кнопки добавления разных типов содержимого и изменение контента для каждого контейнера.

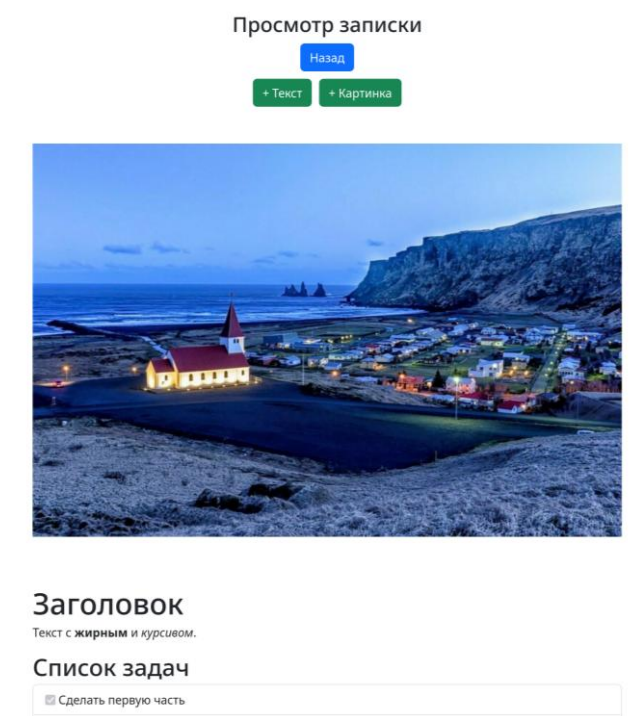


Рисунок 12 - Окно редактирования записок

Взаимодействие пользователя с системой показано на диаграмме (Рисунок 13).

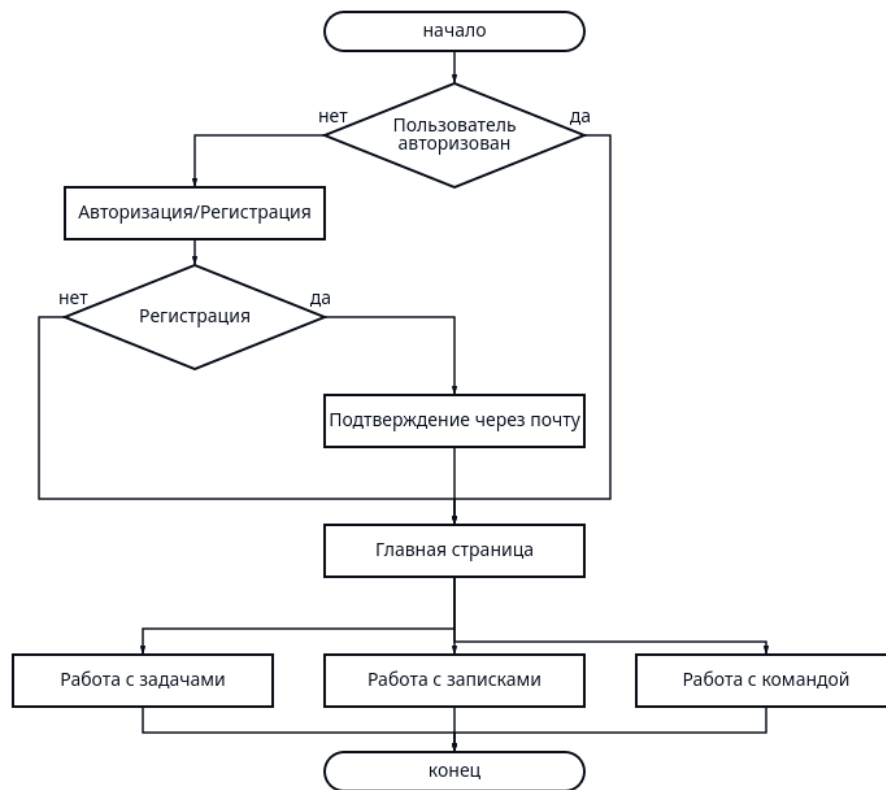


Рисунок 13 - Взаимодействие пользователя с системой

В прототипе были реализованы ключевые экраны: обзор проектов, доска задач, интерфейс создания и редактирования задач, редактор заметок. Визуально проработаны кнопки, карточки задач и текстовые блоки. Для демонстрации основных элементов использовались упрощённые скетчи (например, кнопка добавления задачи, карточка задачи, поле редактирования заметки).

Для лучшего понимания взаимодействия пользователей с системой была подготовлена диаграмма, отражающая основные шаги: авторизация, выбор проекта, создание задач, редактирование, добавление комментариев и заметок, выход из системы. Эта схема помогла выявить потенциальные узкие места и повысить общее удобство использования приложения.

Прототип позволил оценить масштаб системы и подготовить основу для дальнейшей разработки, а также выявить технические и пользовательские риски. Он стал инструментом раннего тестирования концепции, помог в подготовке программных и аппаратных требований, а также дал возможность спрогнозировать экономические и социальные эффекты от внедрения: экономию

времени команды, повышение прозрачности работы и улучшение коммуникации внутри проекта.

Таким образом, этап разработки прототипа стал важным связующим звеном между аналитической частью исследования и практической реализацией веб-приложения.

3.2 Разработка серверной части

На этапе проектирования веб-приложения для совместной работы над проектами был проведён всесторонний анализ современных серверных и клиентских технологий. Целью этого анализа стало определение наиболее подходящих инструментов, обеспечивающих высокую производительность, удобство разработки, масштабируемость и безопасность системы.

Для серверной части рассматривались такие фреймворки, как Django, Spring, Express и Flask. Все они поддерживают работу с основными реляционными и нереляционными СУБД, но Django оказался наиболее сбалансированным решением. Django имеет встроенный ORM для работы с базами данных, мощную административную панель и богатую экосистему библиотек, включая средства для создания REST API. По сравнению со Spring, требующим более глубоких знаний и больших ресурсов, Django проще в освоении, а в отличие от Express и Flask, он поставляется с широким набором готовых решений, что ускоряет разработку

Выбор библиотек

Для обеспечения быстрой, качественной и надежной реализации поставленных задач были выбраны следующие библиотеки Python, предоставляющие готовый функционал и механизмы, необходимые для разработки серверной части приложения. Выбор библиотек обусловлен их стабильностью, широкой распространённостью в профессиональной среде, наличием качественной документации, а также их полной совместимостью с используемым стеком технологий. Полный список:

– `django-rest-framework` - библиотека, предназначенная для создания REST API. Она обеспечивает удобные инструменты для сериализации данных, построения API-эндпоинтов, обработки запросов и ответов, а также включает множество встроенных функций, значительно упрощающих разработку.

– `django-rest-framework-simplejwt` - библиотека, обеспечивающая реализацию авторизации с использованием JWT-токенов. Она позволяет гибко настраивать процесс аутентификации пользователей, реализовывать механизмы обновления токенов и ограничивать доступ к определённым ресурсам.

– `django-cors-headers` - библиотека для настройки обработки CORS-заголовков. Поскольку серверная и клиентская части приложения разворачиваются как отдельные приложения, возникает необходимость в корректной настройке CORS (Cross-Origin Resource Sharing), чтобы обеспечить безопасный обмен данными между ними.

– `psycopg2` - библиотека для подключения и взаимодействия с базой данных PostgreSQL. Она обеспечивает стабильное соединение, выполнение SQL-запросов, управление транзакциями и обработку ошибок на стороне базы данных.

– `pillow` - библиотека для работы с изображениями. В проекте используется для обработки и сохранения изображений, прикрепляемых к записям (например, изменение размера, сжатие, сохранение в различных форматах).

БД

В качестве системы управления базами данных выбрана PostgreSQL. Она обладает высокой надёжностью, поддержкой ACID-транзакций, расширяемостью и активным сообществом. PostgreSQL отлично интегрируется с Django через ORM и драйвер `psycopg2`. Её возможности, включая работу с JSON, полнотекстовый поиск и поддержку сложных запросов, позволяют гибко обрабатывать данные и обеспечивать высокую производительность даже при масштабировании приложения.

Docker

Для разработанного приложения на базе Django была подготовлена конфигурация Docker-контейнера, приведенная в приложении В. Использование Docker позволяет упростить и ускорить развертывание серверной части приложения на любой целевой системе, независимо от её операционной системы и установленного программного обеспечения.

Для работы с Postgres были написаны модели – классы Python, описывающие таблицы БД. Программная реализация модели объекта «проект» представлена в приложении Г.

Обработка клиентских запросов к API управляется функциями. Инструменты библиотеки позволяют создавать такие обработчики с помощью декораторов. Поэтому код не сильно увеличивается. В приложении Д содержатся две реализации: обновление и удаление задачи.

3.3 Разработка клиентской части

Клиентская часть приложения разработана с использованием библиотеки React, основанной на компонентном подходе. Для реализации системы маршрутизации, панели навигации и рабочей области были разработаны собственные компоненты, что позволило обеспечить гибкость и модульность интерфейса.

В современных React-приложениях для организации обмена данными между компонентами нередко применяются контексты или сторонние библиотеки для работы с глобальным хранилищем состояния. Однако подобные решения требуют тщательной архитектурной проработки, так как ошибки на этапе проектирования могут привести к снижению производительности приложения и избыточным перерисовкам интерфейса. В рамках данного проекта было принято решение отказаться от использования глобального хранилища в пользу передачи данных через свойства компонентов (props). Такой подход позволяет контролировать поток данных и минимизировать количество перерисовок компонентов, поскольку неизменяемые данные не вызывают обновления тех частей интерфейса, которые от них не зависят.

Для авторизации и аутентификации пользователей используются JWT-токены, которые хранятся в локальном хранилище браузера. Эти токены автоматически прикрепляются к каждому запросу, за исключением запросов, связанных с процессом авторизации. С целью повышения уровня безопасности токены обновляются с установленным интервалом — один раз в час. При выполнении запросов, связанных с управлением проектами, система дополнительно проверяет, состоит ли текущий пользователь в команде соответствующего проекта. Таким образом, обеспечивается защита от несанкционированных изменений данных других проектов.

Клиентское приложение, как и серверная часть, упаковано в Docker-контейнер. Все необходимые зависимости устанавливаются автоматически при сборке образа, что обеспечивает воспроизводимость и удобство развертывания. Поддерживается два режима работы: режим разработки и продакшн-режим. В режиме разработки обеспечена горячая перезагрузка и удобная отладка, тогда как в продакшн-режиме приложение собирается и запускается в оптимизированной версии для обеспечения максимальной производительности и стабильности.

На основе макетов разработанного прототипа были разработаны компоненты, из которых строятся страницы. Благодаря сервису Pixso получилось сделать элементы, которые максимально похожи на задумку. Чтобы не повторяться с пунктом 3.1, далее будут скриншоты мобильной версии

Страница авторизации (Рисунок 14) была сделана с учетом адаптивности, поэтому она выглядит также, как и в обычной версии.

The image shows a web form titled "Авторизация" (Authorization). It contains two input fields: "Email" and "Пароль" (Password). Below the fields are two blue buttons: "Войти" (Login) and "Регистрация" (Registration). At the bottom of the form is a blue underlined link: "Восстановление пароля" (Password recovery).

Рисунок 14 - Страница авторизации

После авторизации пользователь попадает на главную страницу (Рисунок 15).

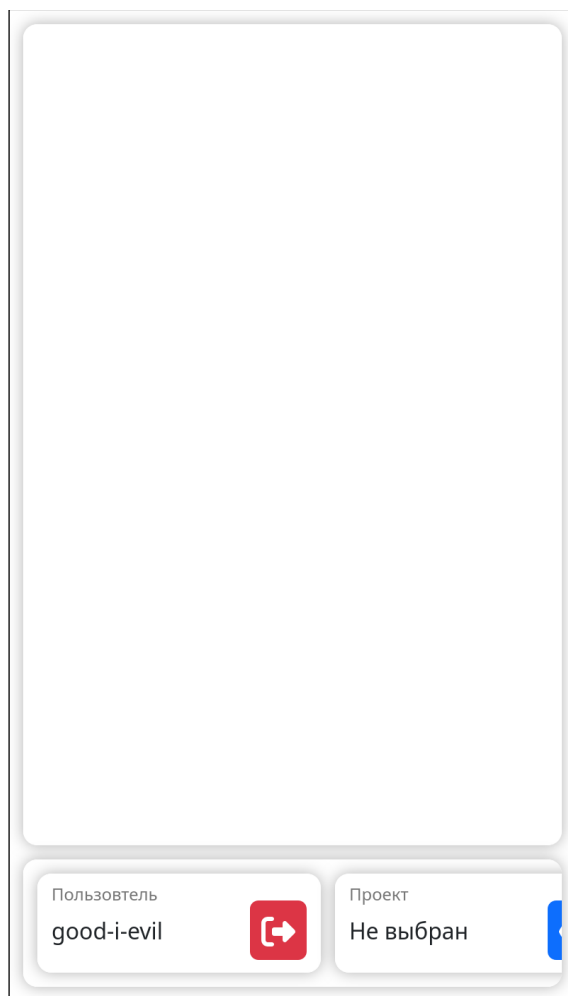


Рисунок 15 - Главная страница

Все страницы и элементы адаптированы для мобильной версии с учетом удобства пользователя. Например, панель навигации располагается снизу, чтобы не тянуться пальцем к верхней части устройства. Компоненты перерисовываются так, чтобы не уменьшать элементы управления и текст. Это хорошо видно на канбан-доске (Рисунок 16).

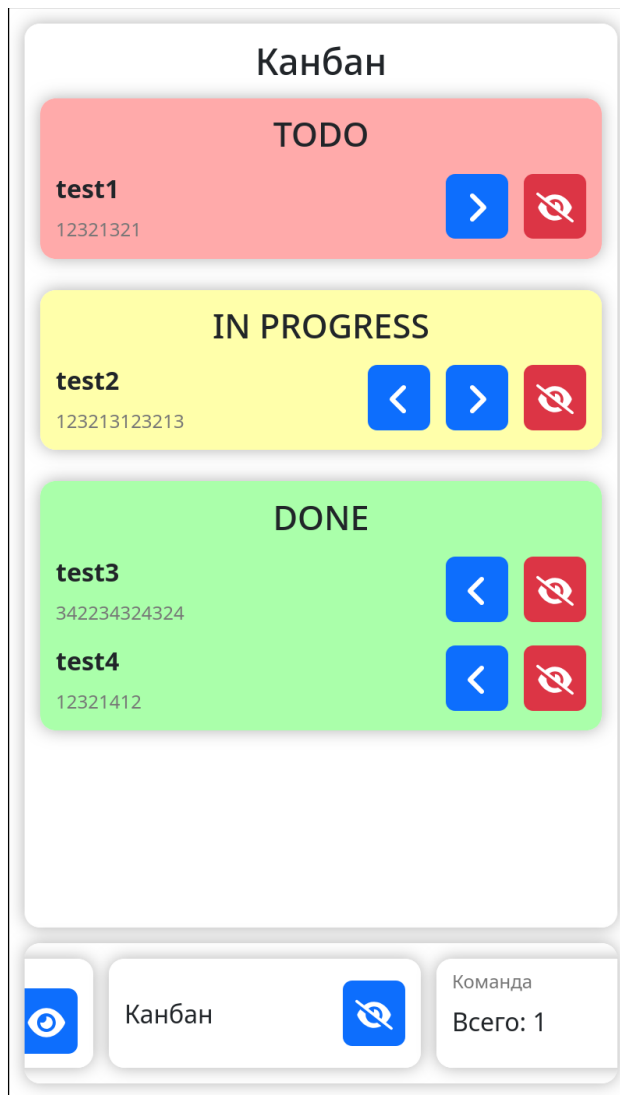


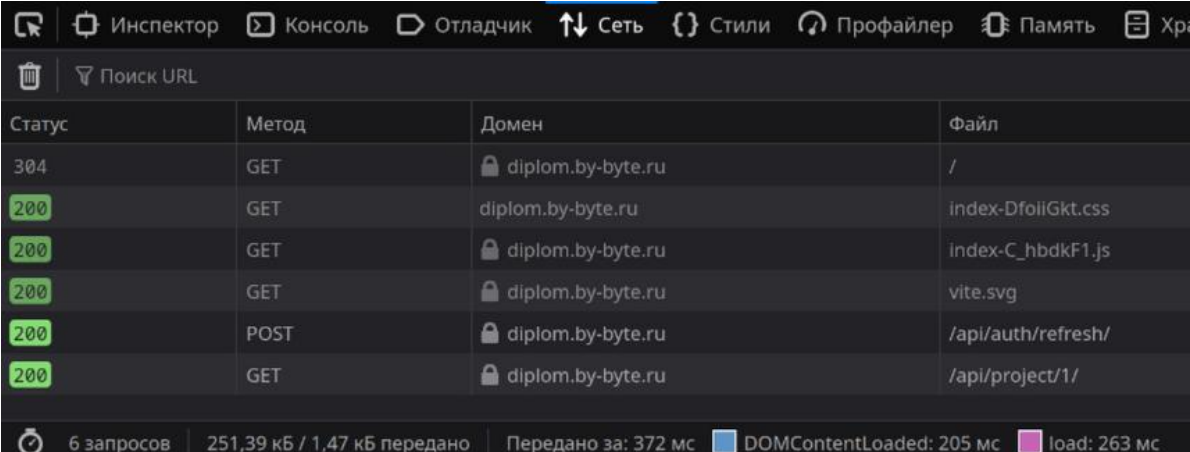
Рисунок 16 - Канбан-доска в мобильной версии

Полученное приложение не использует сложные структуры данных и используют все преимущества React, такие как реактивность и иммутабельность, что позволило создать легкое решение.

3.4 Тестирование приложения

Приложение было протестировано по нескольким аспектам. Учитывались не только отзывы пользователей, но и скорость работы, нагрузка на сервер, безопасность.

Скорость работы — одно из преимуществ данного решения. Без использования кеша или при первой загрузке главная страница загружается за 200 мс., а контент генерируется еще за 400 мс. При повторной загрузке общее время составляет около 500 мс. Результаты показаны ниже (Рисунок 17).



Статус	Метод	Домен	Файл
304	GET	diplom.by-byte.ru	/
200	GET	diplom.by-byte.ru	index-DfoilGkt.css
200	GET	diplom.by-byte.ru	index-C_hbdkF1.js
200	GET	diplom.by-byte.ru	vite.svg
200	POST	diplom.by-byte.ru	/api/auth/refresh/
200	GET	diplom.by-byte.ru	/api/project/1/

6 запросов | 251,39 кБ / 1,47 кБ передано | Передано за: 372 мс | DOMContentLoaded: 205 мс | load: 263 мс

Рисунок 17 - Тестирование скорости загрузки страницы

Приложение также не использует много ресурсов для своей работы. Далее представлено сравнение сервиса с сайтом Github (Рисунок 18). Разница по памяти в 2 раза и использование процессора гораздо ниже.



Firefox (2259)	358МБ	12 %
https://by-byte.ru (3544)	91МБ	0,091 %
Вкладка: Vite + React		
https://github.com (3475)	245МБ	7,4 %
Вкладка: diplom/code/backend/requirements.txt at main · C...		

Рисунок 18 - Используемые ресурсы приложения

Таким образом, полученное решение действительно легкое и может работать на любом устройстве, включая мобильные.

Был проведен опрос конечных пользователей через Google Forms. Все вопросы направлены на получение оценки по 10 бальной шкале. Всего приняло участие 49 пользователей, из которых 38 с мобильной версией сайта. Такая пропорция выбрана специально, чтобы лучше оценить адаптацию интерфейса и удобство для телефонов.

Все пользователи прошли регистрацию и авторизацию на сервисе. Средняя оценка 8.76 (Рисунок 19). Можно сделать вывод, что процесс авторизации соответствует ожиданиям.

Удобство авторизации
49 ответов

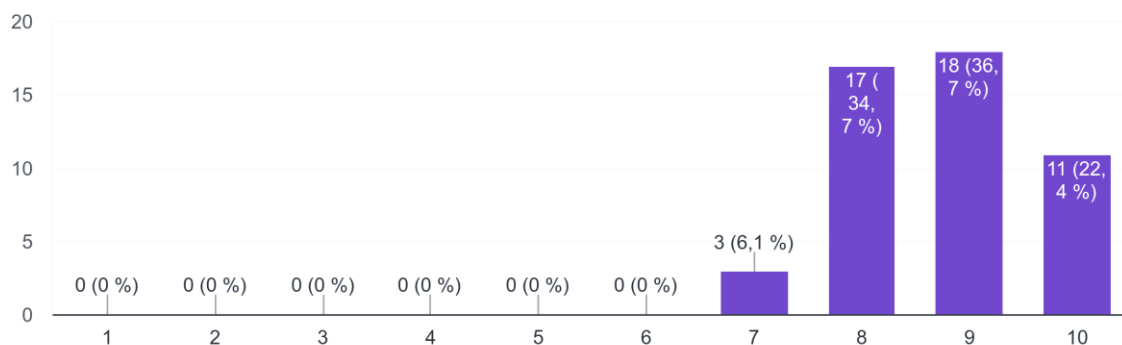


Рисунок 19 - Оценка удобства авторизации

Пользователи были приглашены в тестовый проект для демонстрации возможностей. После этого они попробовали все возможности сервиса и оценили насколько просто это можно сделать (Рисунок 20). Средняя оценка 8.59, что говорит об успешном выполнении работы.

Простота выполнения повседневных задач
49 ответов

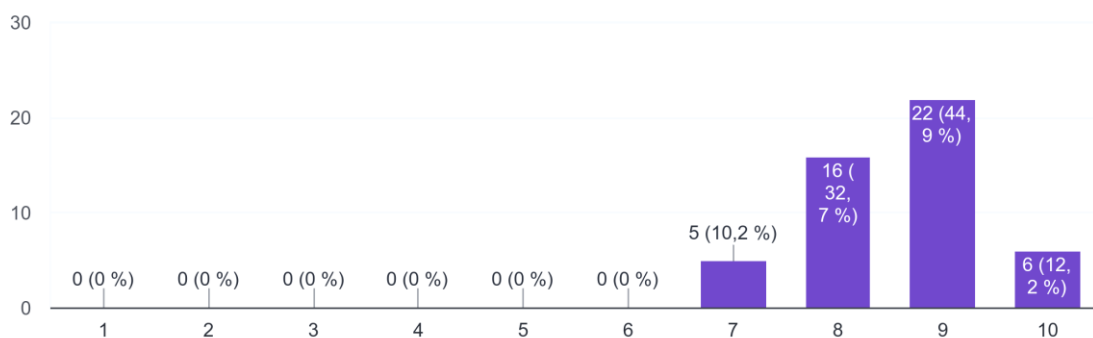


Рисунок 20 - Оценка простоты выполнения повседневных задач

Пользователям понравился интерфейс системы (Рисунок 21). Использование простых и понятных блоков позволило получить достаточно высокую оценку. Почти все смогли разобраться без дополнительных подсказок. Все же есть негативные отзывы, поэтому в интерфейс необходимо добавить подписи к некоторым кнопкам и блокам. Средняя оценка 8.53.

Удобство интерфейса

49 ответов

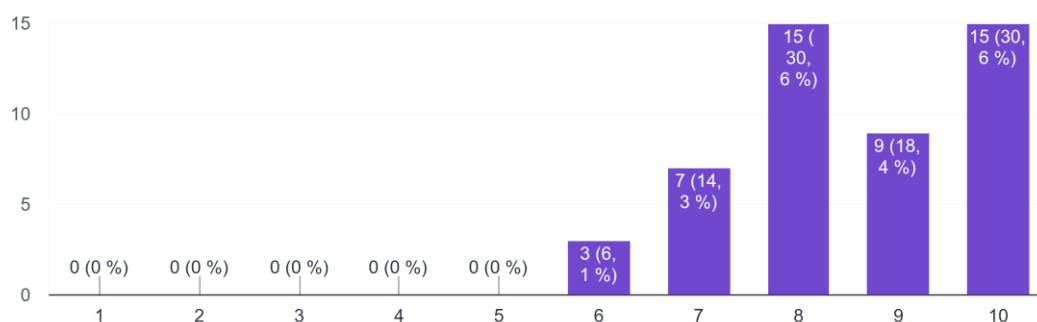


Рисунок 21 - Оценка удобства интерфейса

Невозможно было бы не спросить конечного пользователя насколько разработанное решение соответствует его ожиданиям (Рисунок 22). Средняя оценка 8.33 показывает, что в основном сервис удовлетворяет потребностям пользователей. Были замечания по ограничению прав пользователей в команде, добавлению даты и других атрибутов задачам, а также больше блоков в записках, например таблицы. Несмотря на это, реализованных функций хватает, чтобы пользоваться приложением.

Насколько решение покрывает Ваши потребности

49 ответов

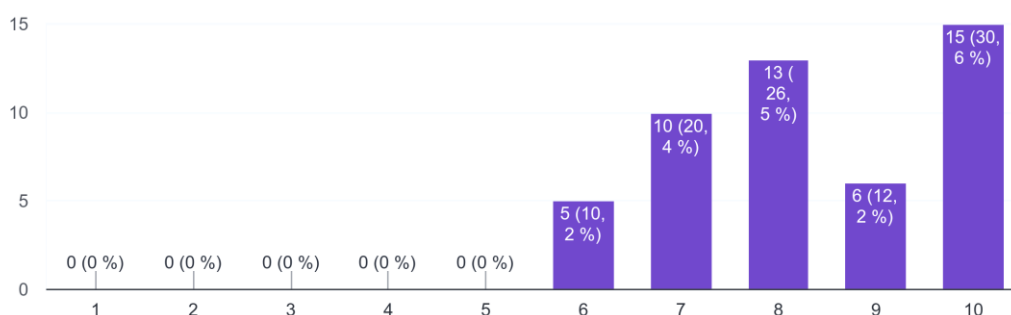


Рисунок 22 - Оценка соответствия ожиданий и результата

В целом получены хорошие оценки (Рисунок 23). Средняя оценка 8.88. Это более чем успешный результат. Быстрый, отзывчивый и функциональный сервис позволяет организовать удобное ведение проектов.

Общее впечатление от сервиса

49 ответов

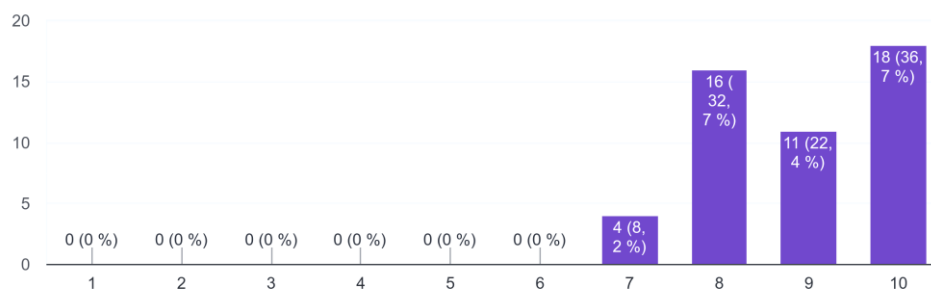


Рисунок 23 - Оценка сервиса в целом

Проведённое тестирование показало, что разработанное приложение соответствует основным требованиям, предъявляемым к современным сервисам для совместной работы. Оно демонстрирует высокую скорость работы, низкое потребление ресурсов и хорошую масштабируемость, что позволяет использовать его даже на мобильных устройствах. Оценки пользователей подтверждают удобство интерфейса, простоту выполнения повседневных задач и высокий уровень удовлетворённости функциональностью. Несмотря на отдельные замечания и предложения по улучшению, средние оценки выше 8 баллов из 10 свидетельствуют об успешной реализации поставленных задач и значительном потенциале приложения для дальнейшего развития.

ЗАКЛЮЧЕНИЕ

Выпускная квалификационная работа на тему «Разработка веб-приложения для организации совместной работы над проектами» была направлена на решение актуальной задачи — создание удобного и функционального программного продукта, который позволит небольшим командам эффективно управлять задачами, координировать действия участников и организовывать совместное хранение и использование информации.

Актуальность выбранной темы определяется растущей потребностью в инструментах для коллективной работы, особенно в условиях цифровизации, распространения удалённого формата и гибких методологий управления проектами. Анализ существующих решений, таких как Trello, Asana, Jira и Notion, показал, что, несмотря на широкий функционал, многие из них оказываются сложными для освоения, перегруженными лишними функциями и имеют ограничения в бесплатных версиях. Это создаёт неудобства для небольших коллективов, которые нуждаются в простых и доступных инструментах, позволяющих сосредоточиться на выполнении задач, а не на освоении системы. Таким образом, проведённый анализ позволил сформировать чёткое понимание потребностей целевой аудитории и подтвердить актуальность разработки.

Анализ существующих решений показал, что важнейшими функциями являются: ведение списка задач (todo-лист), использование канбан-доски, возможность оставлять текстовые заметки и комментарии, а также простой и интуитивный интерфейс. Особое внимание было уделено проблемам, с которыми сталкиваются небольшие команды: сложность интерфейсов, лишние функции, ограничения бесплатных версий и высокая зависимость от интеграций. Эти выводы стали основой для постановки задач дипломной работы.

Рассматривались различные архитектурные подходы, принципы разработки и обеспечения безопасности. Выбор таких технологий, как Django для серверной части, React для клиентского интерфейса, PostgreSQL для

хранения данных и Docker для контейнеризации, обеспечил соответствие приложения современным стандартам. Кроме того, значительное внимание уделялось вопросам безопасности: обеспечена защита передаваемых данных через протокол HTTPS, применены механизмы хеширования паролей, защита от CSRF- и XSS-атак, что существенно повышает доверие пользователей и снижает риски эксплуатации уязвимостей.

Разработка прототипа приложения позволила смоделировать интерфейсы, проверить пользовательские сценарии и выявить потенциальные проблемы. Далее были реализованы серверная и клиентская части. На сервере с помощью Django и Django REST Framework были созданы API для управления задачами, проектами и пользователями, реализованы механизмы аутентификации с использованием JWT-токенов. Клиентская часть на базе React обеспечила отзывчивый и удобный интерфейс с возможностью работы в режиме SPA (Single Page Application), что позволило минимизировать время загрузки страниц и повысить комфорт работы. Важным достижением стала адаптация интерфейса под мобильные устройства, что делает приложение доступным в любом месте и с любого устройства. На завершающем этапе было проведено тестирование, включавшее проверку функциональности, производительности и безопасности. Результаты тестирования показали, что приложение демонстрирует высокую скорость отклика, устойчивость к нагрузкам и корректную работу всех функций.

В результате в ходе выполнения работы были решены все поставленные задачи:

- проведён анализ существующих решений и определены основные потребности целевой аудитории;
- сформулированы и обоснованы функциональные и нефункциональные требования к приложению;
- выбраны оптимальные архитектурные и технологические решения;
- реализованы серверная и клиентская части приложения;
- проведено тестирование готового продукта и дана оценка его эффективности.

Полученное веб-приложение позволяет пользователям:

- создавать и управлять задачами с разделением по статусам;
- использовать канбан-доску для наглядного отображения рабочего процесса;
- вести текстовые заметки и комментарии;
- приглашать участников для совместной работы без сложной настройки ролей;
- работать с разных устройств благодаря адаптивному интерфейсу.

В практическом плане разработанное приложение имеет высокую значимость. Оно может быть внедрено в образовательных организациях, малом бизнесе, стартапах и неформальных командах. Минимальный порог входа и простота интерфейса позволяют начать работу практически без обучения, а низкие требования к инфраструктуре делают продукт доступным даже для небольших организаций с ограниченными ресурсами. В экономическом аспекте использование open-source инструментов и Docker-контейнеров позволяет минимизировать затраты на развертывание и поддержку приложения. Социальная эффективность проекта проявляется в улучшении коммуникации между участниками, повышении прозрачности процессов, сокращении времени на организацию работы и распределение задач.

Перспективы дальнейшего развития приложения включают:

- расширение функционала (например, добавление календарей, напоминаний, интеграций с внешними сервисами);
- внедрение аналитических инструментов для мониторинга эффективности командной работы;
- разработку мобильных приложений для Android и iOS;
- усовершенствование системы уведомлений и взаимодействия между участниками.

Таким образом, выполненная работа имеет как теоретическое, так и прикладное значение. Она не только демонстрирует возможности современных технологий веб-разработки, но и предлагает конкретное решение реальной

проблемы — организации совместной работы небольших команд. Полученные результаты соответствуют поставленной цели и задачам, а разработанное приложение обладает высоким потенциалом для практического применения и дальнейшего развития.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Чекмарев А. В. Управление цифровыми проектами и процессами. - 2 изд. - М.: Юрайт, 2025. - 424 с.
2. Фаулер С. Почему они не работают? Новый взгляд на мотивацию сотрудников. - М.: Альпина Паблишер, 2020. - 192 с.
3. Почему использовать Jira для ведения проектов неэффективно // VC URL: <https://vc.ru/services/54288-pochemu-ispolzovat-jira-dlya-vedeniya-proektov-neeffectivno> (дата обращения: 11.12.2024).
4. Schwaber, K., Beedle, M. Agile Software Development with Scrum. - Upper Saddle River: Prentice Hall, 2001. - 176 с.
5. Акмалов, О. И. Критерии выбора информационной системы управления проектами // Молодой ученый. - 2023. - №22. - С. 159-161.
6. Селиховкин И. Управление ИТ-проектом Эффективная система «с нуля» в любой организации. - СПб.: РМ, 2010. - 89 с.
7. Коул Р., Скотчер Э. Блистательный Agile. Гибкое управление проектами с помощью Agile, Scrum и Kanban. - СПб.: Питер, 2019. - 304 с.
8. Сазерленд Д. Scrum. Революционный метод управления проектами. - М.: МИФ, 2025. - 272 с.
9. Тестина Я.С., Чумаков В.Н. Управление проектами: учебное пособие для вузов. - Гатчина: ГИЭФПТ, 2023. - 69 с.
10. Осипов Д.В. Управление проектами: Учебное пособие для магистров направления «Менеджмент». - М.: РУТ (МИИТ), 2017. - 170 с.
11. Эдмондсон Э. Взаимодействие в команде. - Эксмо, 2016. - 320 с.
12. Гринберг М. Разработка веб-приложений с использованием Flask на языке Python. - М.: ДМК Пресс, 2017. - 272 с.
13. Berson A. Client-server architecture. - New York: McGraw-Hill, 1992. - 486 с.
14. Введение в серверную часть // MDN WEB DOCS URL: https://developer.mozilla.org/ru/docs/Learn_web_development/Extensions/Server-side/First_steps/Introduction (дата обращения: 23.01.2025)

15. Контейнерные микрослужбы // Microsoft URL: <https://learn.microsoft.com/ru-ru/dotnet/architecture/maui/micro-services> (дата обращения: 15.01.2025).
16. Митра Р., Надареишвили И. Микросервисы. От архитектуры до релиза. - СПб.: Питер, 2025. - 336 с.
17. Браун И. Веб-разработка с применением Node и Express. - СПб.: Питер, 2021. - 336 с.
18. Django documentation // Django URL: <https://www.djangoproject.com/> (дата обращения: 20.11.2024).
19. React documentation // React URL: <https://react.dev/learn> (дата обращения: 27.11.2024).
20. Documentation // PostgreSQL: Documentation URL: <https://www.postgresql.org/docs/> (дата обращения: 06.12.2024).
21. Guides // Docker Docs URL: <https://docs.docker.com/guides/> (дата обращения: 30.12.2024).
22. Lazzaro L. L. Ultimate Web API Development with Django REST Framework. - Delhi: Orange Education Pvt Ltd, AVA, 2025. - 265 с.
23. Бирн Д. Безопасность веб-приложений на PYTHON. - М.: ДМК Пресс, 2023. - 334 с.
24. Securing Docker: Best practices for robust container security // SolDevelo URL: <https://soldevelo.com/blog/securing-docker-best-practices-for-robust-container-security/> (дата обращения: 15.02.2025).
25. Фаулер М. Архитектура корпоративных программных приложений. - М.: Издательский дом "Вильямс", 2006. - 541 с.
26. Соммервилл И. Инженерия программного обеспечен. - 6 изд. - М.: Издательский дом "Вильямс", 2002. - 618 с.
27. Чиннатамби К. Изучаем React. - 2 изд. - М.: Бомбора, 2022. - 368 с.
28. Capture, organize, and tackle your to-dos from anywhere // Trello URL: <https://trello.com/> (дата обращения: 17.12.2024).

29. Manage your team's work, projects, & tasks online // Asana URL: <https://asana.com/ru> (дата обращения: 18.12.2024).
30. Jira | Issue & Project Tracking Software // Jira URL: <https://www.atlassian.com/software/jira> (дата обращения: 19.12.2024).
31. Your connected workspace for wiki, docs & projects // Notion URL: <https://www.notion.com/> (дата обращения: 20.12.2024).
32. A Free Online UI/UX Design Tool // Pixso URL: <https://pixso.net/> (дата обращения: 19.01.2025).
33. Gene K., Humble J., Debois P., Willis J. The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations. - Portland: IT Revolution Press, LLC, 2016. - 644 с.
34. Humble J., Molesky, J., O'Reilly B. Lean Enterprise: How High Performance Organizations Innovate at Scale. - Sebastopol: O'Reilly Media, Incorporated, 2020. - 317 с.
35. Форсгрэн Н., Хамбл Д. Ускоряйся! Наука DevOps : Как создавать и масштабировать высокопроизводительные цифровые организации. - М.: Альпина PRO, 2022. - 220 с.
36. Кочер П. С. Микросервисы и контейнеры Docker. - М.: ДМК Пресс, 2019. - 240 с.
37. Ной, Г., Кеннеди Б. Python и DevOps: Ключ к автоматизации Linux. - СПб.: Питер, 2022. - 544 с.
38. Сейерс, Э. Х. Docker на практике. - М.: ДМК Пресс, 2020. - 516 с.
39. Хоффман Э. Безопасность веб-приложений. Разведка, защита, нападение. - 2 изд. - Астана: Sprint Book, 2025. - 432 с.
40. Теджас К. React. К вершинам мастерства. - Астана: АЛИСТ, 2025. - 368 с.

Программная реализация страницы авторизации

```
[STEP_LOGIN_REGISTER]: (
  <div className='form'>
    <h3 className='text-center'>Авторизация</h3>

    {error && <p className='text-danger text-center'>{error}</p>}

    <div className='form-floating'>
      <input
        key='login_register_email'
        type='email'
        className='form-control'
        placeholder='Enter Email'
        value={authData.email ?? ''}
        onChange={(e) =>
          setAuthData((old) => ({ ...old, email: e.target.value }))
        }
      />
      <Label>Email</Label>
    </div>

    <div className='form-floating'>
      <input
        key='login_register_password'
        type='password'
        className='form-control'
        placeholder='Enter password'
        value={authData.password ?? ''}
        onChange={(e) =>
          setAuthData((old) => ({ ...old, password: e.target.value }))
        }
      />
      <Label>Пароль</Label>
    </div>

    <div className='form-buttons'>
      <button
        type='button'
        className='btn btn-primary'
        onClick={() => {
          login(authData).then((response) => {
            if (response.ok) {
              token.setTokens(response.data.tokens)
              token.setUserInfo(response.data.userInfo)

              if (response.data.tokens.access) navigation('/')
            } else setError(() => response.detail)
          })
        }}>
        Войти
      </button>

      <button
        type='button'
        className='btn btn-primary'
        onClick={() => {
          register(authData).then((response) => {
            if (response.ok) {
              token.setUserInfo(response.data.userInfo)
              if (response.data.userInfo)
                setActiveStep(() => STEP_CONFIRM_REGISTER)
            } else setError(() => response.detail)
          })
        }}>
        Регистрация
      </button>
    </div>

    <button
      type='button'
      className='btn btn-link'
      onClick={() => {
        setActiveStep(() => STEP_RECOVERY)
      }}>
      Восстановление пароля
    </button>
  </div>
),
```

Рисунок 1.1 – Фрагмент кода формы регистрации

Программная реализация элемента навигации

```

<div className='project-block block'>
  <div className='block-content'>
    <p className='block-label'>Проект</p>
    <p className='block-content'>{activeProject.name ?? 'Не выбран'}</p>
  </div>

  <div className='block-action'>
    <button
      className='btn btn-primary'
      onClick={() => {
        setActiveWindow((old) =>
          [WINDOW_PROJECT, WINDOW_PROJECT_EDIT].includes(old)
            ? WINDOW_CLEAR
            : WINDOW_PROJECT
        )
        setWindowData(() =>
          [WINDOW_PROJECT, WINDOW_PROJECT_EDIT].includes(activeWindow)
            ? {}
            : { activeProject, setActiveProject, updateActiveProject }
        )
      }}>
    <img
      src={
        [WINDOW_PROJECT, WINDOW_PROJECT_EDIT].includes(activeWindow)
          ? EyeSlash
          : Eye
      }
    />
  </button>
</div>
</div>

```

Рисунок 2.1 – Фрагмент кода элемента навигации (выбор проекта)

Конфигурация Docker контейнера серверного приложения

```
FROM python:3.13.3-slim-bullseye

RUN apt update && apt install -y netcat

WORKDIR /app

RUN pip install -U pip

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

ENTRYPOINT [ "/app/entrypoint.sh" ]
```

Рисунок 3.1 – Конфигурация Docker контейнера Django приложения

Программная реализация модели объекта «проект»

```
class Project(models.Model):
    name = models.CharField(
        max_length=100,
        unique=True,
        verbose_name="Название",
    )
    description = models.TextField(
        null=True,
        blank=True,
        verbose_name="Описание",
    )

    created_at = models.DateTimeField(
        auto_now_add=True,
        verbose_name="Дата создания",
    )
    updated_at = models.DateTimeField(
        auto_now=True,
        verbose_name="Дата обновления",
    )

    project_users: models.QuerySet["ProjectUser"]
    tasks: models.QuerySet["Task"]
    notes: models.QuerySet["Note"]

    def __str__(self) → str:
        return f"{self.name}"

class Meta:
    verbose_name = "Проект"
    verbose_name_plural = "Проекты"
    db_table = "project"
```

Рисунок 4.1 – Полная реализация модели (таблицы) объекта «проект»

Программная реализация обработчиков API

```

@api_view(["POST"])
@permission_classes([IsAuthenticated])
@validate_project_user
def update_task(request, project_id: int, task_id: int) → Response:
    task: Task | None = Task.objects.filter(id=task_id).first()
    if task is None:
        return Response(
            {"detail": "Task not found"},
            status=status.HTTP_400_BAD_REQUEST,
        )

    name = request.data.get("name")
    description = request.data.get("description")
    stage = request.data.get("stage")
    is_show_in_kanban = request.data.get("is_show_in_kanban")

    if name:
        task.name = name
    if description:
        task.description = description
    if stage:
        task.stage = stage
    if is_show_in_kanban is not None:
        task.is_show_in_kanban = is_show_in_kanban

    task.save()

    result = dict(TaskSerializer(task).data)

    return Response(
        {
            "task": result,
        }
    )

@api_view()
@permission_classes([IsAuthenticated])
@validate_project_user
def delete_task(request, project_id: int, task_id: int) → Response:
    Task.objects.filter(id=task_id).delete()
    return Response({})

```

Рисунок 5.1 – Полная реализация функции обработки запроса пользователя на обновление информации о задаче