МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Кафедра «Прикладная математика и информатика»
(наименование)

09.03.03 Прикладная информатика
(код и наименование направления подготовки / специальности)

Разработка программного обеспечения
(направленность (профиль) / специальности)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему Разработка системы управления проектами для компании ООО «ВОРЛДИНТЕРТЕХ РУС»

Обучающийся Д.А. Травин
(Инициалы Фамилия) (личная подпись)

Руководитель Н.Н. Рогова
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Консультант канд. филол. наук, доцент М.В. Дайнеко
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2025

Аннотация

Тема выпускной квалификационной работы: «Разработка системы управления проектами для компании "ВорлдИнтерТех Рус"».

Бакалаврская работа посвящена разработке системы управления проектами, адаптированной под внутренние бизнес-процессы компании «ВорлдИнтерТех Рус». В рамках работы была поставлена задача создания веб-приложения, реализующего функционал управления проектами, задачами и участниками.

Работа состоит из введения, трёх глав, заключения и списка использованной литературы

В первой главе проанализированы текущие инструменты управления проектами, выявлены их ограничения и определены требования к системе.

Во второй главе представлены архитектура и проектирование приложения, модели данных и структуры интерфейса.

Третья глава включает разработка программного обеспечения и комплексное тестирование, охватывающее модульное, функциональное, интеграционное и нагрузочное тестирование системы.

В заключении представлены выводы о соответствии системы заявленным требованиям и её готовности к внедрению в корпоративную среду.

Бакалаврская работа состоит из введения, трёх глав, заключения и списка использованной литературы.

Работа содержит: 80 страниц, 25 рисунков, 12 таблиц, 6 приложений и список из 20 используемых источников

Abstract

The theme of the graduate qualification work: “Development of project management system for the company ‘WorldInterTech Rus’”.

The bachelor's work is devoted to the development of a project management system adapted to the internal business processes of the company “WorldInterTech Rus”. According to the work was set the task of creating a web application that realizes the functionality of project, task and participant management.

The bachelor's thesis consists of an introduction, three chapters, a conclusion and a list of references used.

In the first chapter there is an analysis of current management tools in the project. Also, in this chapter are identified tools limitations and system requirements

In the second chapter architecture and design of the application, data models and interface structures.

The third chapter includes software development and comprehensive testing covering unit, functional, integration and load testing of the system.

In conclusion there is the presentation of the system with the stated requirements and its readiness for implementation in the corporate environment.

The work contains: 80 pages, 25 figures, 12 tables, 6 appendices and a list of 20 sources used

Оглавление

Введение

В условиях цифровизации и роста объёмов информации бизнесу требуются надёжные инструменты для эффективного управления проектной деятельностью. Это особенно актуально для компаний, в которых сотрудники параллельно задействованы в нескольких задачах, требующих координации и контроля. Современные системы управления проектами должны не только упрощать планирование и контроль сроков, но и учитывать специфику бизнес-процессов, автоматизировать рутинные операции и обеспечивать полноценную аналитику.

Компания «ВорлдИнтерТех Рус» на практике использовала популярные решения в управлении проектами. Однако их функциональность не в полной мере соответствовала требованиям организации. Основные проблемы включали слабую интеграцию с внутренними процессами, перегруженность интерфейсов и ограниченные возможности по распределению ролей. Это приводило к усложнению контроля задач и снижению общей эффективности работы команды.

Объектом исследования выступает процесс управления проектами в компании, включая создание проектов, постановку задач, контроль сроков и взаимодействие сотрудников.

Предметом исследования - разработка веб-приложения для управления проектами, ориентированного на потребности компании и интеграцию с существующими бизнес-процессами.

Целью выпускной квалификационной работы является разработка системы управления проектами для компании «ВорлдИнтерТех Рус», обеспечивающей централизованное хранение данных, визуализацию структуры проектов, создания и распределения задач, а также интеграцию с уведомлениями и формирование отчётности.

Для достижения поставленной цели необходимо решить задачи анализа проблемы, определения требований, проектирования архитектуры системы,

разработки компонентов клиентской и серверной частей, а также проведения комплексного тестирования для оценки стабильности и производительности системы.

В первой главе проанализированы проблемы управления проектами в компании, выявлены ограничения текущих решений, сформулированы требования к системе на основе FURPS+ и разработана архитектурная модель компании «ВорлдИнтерТех Рус», интегрирующая бизнес-процессы и информационные системы.

Во второй главе разработана архитектурная модель ПО, включающая клиентскую часть, серверную часть и базу данных. Созданы UML-диаграммы, логическая и физическая модели данных, а также реализована компонентная структура интерфейса, направленная на удобство пользователей.

В третьей главе была разработана система управления проектами и проведено её комплексное тестирование, включающее модульное, функциональное, интеграционное и нагрузочное тестирование. Проверены ключевые пользовательские сценарии, взаимодействие компонентов и устойчивость системы при высокой нагрузке, что подтвердило её стабильность и соответствие требованиям.

Глава 1 Постановка задачи на разработку системы управления проектами

1.1 Значения программного обеспечения для системы управления проектами

Программное обеспечение (ПО) - это набор программ, предназначенный для решения прикладных задач на вычислительных устройствах. В управлении проектами ПО позволяет автоматизировать планирование, контроль, распределение ресурсов и координацию действий сотрудников.

Разработка специализированного программного обеспечения для управления проектами позволит компании устранить текущие недостатки в организации рабочих процессов: недостаточную эффективность распределения ресурсов, слабую координацию действий сотрудников, недостаточную прозрачность и контроль на всех этапах выполнения проектов.

«Диаграмма причины-следствия Исикавы (Cause-and-Effect-Diagram, «рыбья кость», «рыбий скелет») – это графический метод анализа и формирования причинно-следственных связей, инструментальное средство в форме рыбьей кости для систематического определения причин проблемы и последующего графического представления» [4]. Представлена на рисунке 1

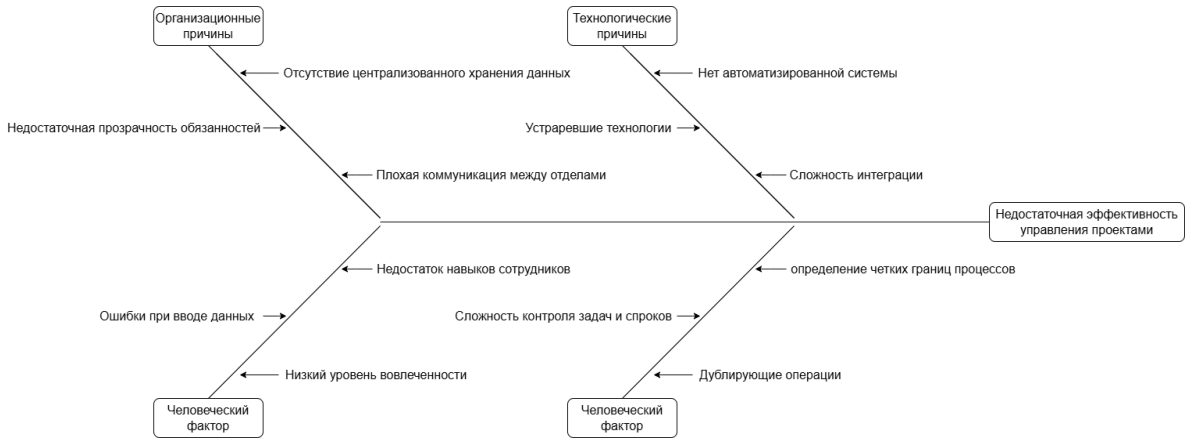

Рисунок 1 – Диаграмма причина-следствие

Представленная диаграмма отражает проблему «Недостаточная эффективность управления проектами». Причины проблемы разделены на четыре основные категории: организационные, технологические, человеческий фактор и процессные причины.

Таким образом, диаграмма четко показывает взаимосвязь различных причин и факторов с ключевой проблемой, что позволит более эффективно разрабатывать меры по её устранению и улучшению текущих процессов управления проектами в компании.

Для демонстрации эффективности системы управления проектами на рисунке 2 представлена сравнительная диаграмма, отражающая изменение ключевых показателей до и после внедрения в реальной компании. Пример иллюстрирует, как эффективное управление проектами напрямую влияет на рост производительности, улучшение бизнес-процессов и повышение удовлетворённости сотрудников.

Рисунок 2 – Сравнительная диаграмма ключевых показателей

Анализируя данные диаграммы, можно отметить, что после внедрения системы управления проектами наблюдается значительный рост всех ключевых показателей. Производительность выросла с 60% до 90%, эффективность работы увеличилась на 60%, а удовлетворенность пользователей повысилась с 50% до 95%. Также существенно ускорилась обработка задач и улучшилась прозрачность процессов. Это подтверждает высокую эффективность и практическую ценность разработанного программного обеспечения.

Таким образом, внедрение СУП оказывает позитивное влияние на работу компании, обеспечивая заметное улучшение основных бизнес-показателей.

1.2 Постановка задачи на разработку программного обеспечения

1.2.1 Текущая архитектура программного обеспечения компании

Проведен анализ текущей архитектуры информационных технологий компании «ВорлдИнтерТех Рус». Определены оптимальные подходы интеграции нового программного обеспечения в уже существующую информационную инфраструктуру компании.

На рисунке 3 представлена схема элементов архитектуры компании, демонстрирующая взаимосвязь информационных технологий, информационных систем и бизнес-архитектуры организации.

Рисунок 3 – Элементы архитектуры компании

На представленной схеме изображены основные элементы архитектуры компании и их взаимосвязь. Архитектура компании объединяет три ключевых компонента: информационные технологии, информационные системы и бизнес-архитектуру.

Информационные технологии (ИТ) – это совокупность программных и аппаратных средств, обеспечивающих стабильную работу корпоративных систем и поддерживающих бизнес-процессы компании. Информационные системы (ИС) автоматизируют ключевые процессы и служат связующим звеном между ИТ-инфраструктурой и целями бизнеса. Бизнес-архитектура (БА) задаёт стратегические ориентиры, определяя требования к ИС и ИТ-решениям. Все элементы архитектуры тесно взаимосвязаны и формируют единую систему, обеспечивающую надёжную и эффективную работу компании [7].

В организации применяется многоуровневая архитектура, включающая несколько слоёв. Инфраструктурный слой представлен серверами и сетевым оборудованием, обеспечивающими отказоустойчивость и стабильность. Слой данных реализован на базе централизованного хранилища с использованием нереляционной базы данных, что обеспечивает гибкость и высокую скорость обработки разнотипных данных. Прикладной слой включает используемые ИС, такие как Trello и корпоративная почта. Trello обеспечивает базовое управление задачами, но ограничена в аналитике, масштабируемости и интеграции. Почта служит для коммуникации, но не решает задачи проектного управления.

Анализ показал, что Trello не удовлетворяет требованиям компании из-за слабой структуризации задач, отсутствия развитой отчётности и недостаточной интеграции с внутренними системами. Это стало основанием для создания новой специализированной системы управления проектами, которая будет размещена на прикладном уровне архитектуры.

Разрабатываемая система будет построена на современных веб-технологиях и интегрирована с существующими сервисами через REST API.

Использование нереляционной базы данных обеспечит гибкость модели хранения, высокую производительность и возможность масштабирования в условиях роста компании.

На рисунке 4 продемонстрирована взаимосвязь между корпоративной ИТ-архитектурой, бизнес-архитектурой и разрабатываемой СУП.

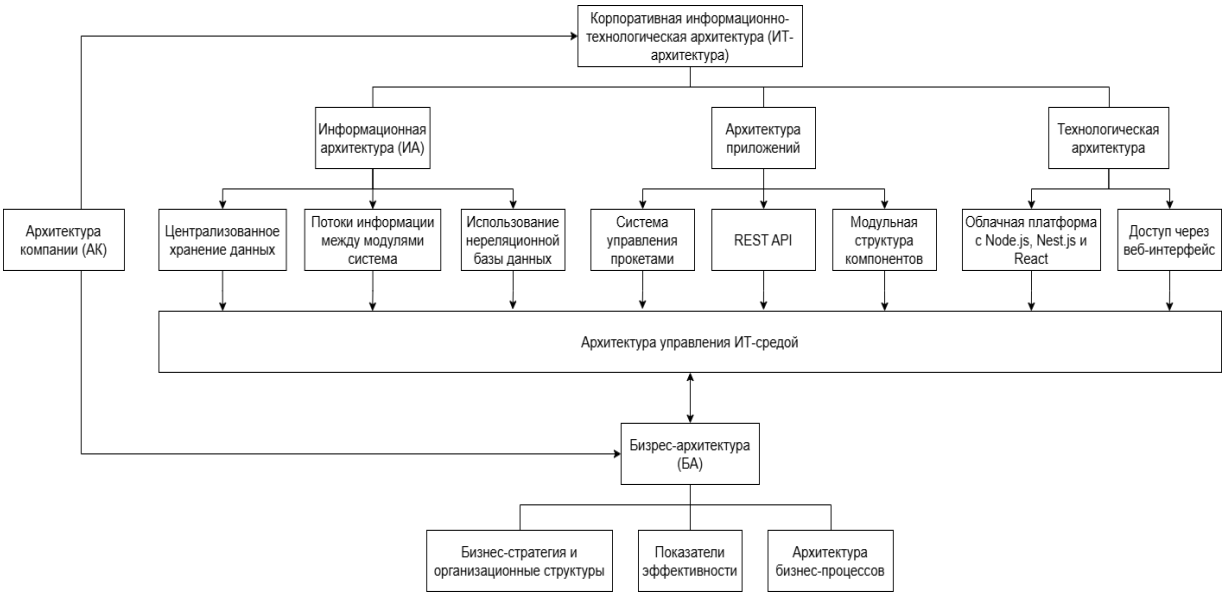

Рисунок 4 – Архитектурная модель компании с учётом внедрения системы управления проектами

На представленной схеме отображена структура архитектуры компании, объединяющая три ключевых компонента: архитектуру компании (АК), корпоративную информационно-технологическую архитектуру (ИТ-архитектуру) и бизнес-архитектуру (БА). Эти уровни взаимодействуют между собой, обеспечивая непрерывную поддержку бизнес-процессов организации за счёт современных IT-решений.

ИТ-архитектура представлена тремя уровнями:

- информационная архитектура (ИА) включает централизованное хранение данных, информационные потоки между модулями, а также использование нереляционной базы данных, что обеспечивает гибкость и высокую скорость обработки информации;

- архитектура приложений охватывает ключевые компоненты нового программного решения - систему управления проектами, REST API для интеграции с внешними и внутренними системами, а также модульную структуру, упрощающую масштабирование и сопровождение системы;
- технологическая архитектура включает стек технологий, используемый при реализации системы: облачная платформа на базе Node.js, Nest.js и React. Взаимодействие пользователей осуществляется через веб-интерфейс, доступный из любого устройства.

Бизнес-архитектура включает стратегические цели и задачи компании, организационные структуры, архитектуру бизнес-процессов и ключевые показатели эффективности, которые формируют требования к IT-системам. Новая система управления проектами нацелена на упрощение и повышение прозрачности управления.

Центральную роль в модели играет архитектура управления ИТ-средой, обеспечивающая согласованность взаимодействия всех уровней, включая процессы, данные, технологии и пользователей.

Таким образом, представленная архитектура отражает комплексный подход к внедрению новой информационной системы, учитывающий как технические, так и бизнес-аспекты функционирования предприятия.

1.2.2 Архитектурные особенности разрабатываемого ПО

Необходимо разработать систему на современных подходах к проектированию. Её структура предполагает независимые модули, которые взаимодействуют друг с другом через программные интерфейсы. Модульная архитектура позволит оперативно адаптировать систему к изменяющимся условиям, добавлять новые функции или корректировать существующие без необходимости полной переработки ПО.

Централизованное хранение данных обеспечивается клиент-серверной архитектурой, которая создаёт единую точку доступа к информации. Такая

структура способствует улучшению управления данными, так как все изменения мгновенно сохраняются в центральной базе, исключая дублирование или потерю информации.

Для обеспечения гибкости и стандартизации взаимодействия между компонентами системы используется REST API, которое стандартизирует процесс обмена данными и обеспечивает совместимость с большинством современных технологий [20].

Архитектура предусматривает высокую устойчивость к сбоям и возможность масштабирования. Система будет поддерживать рост нагрузки, связанный с увеличением числа пользователей и проектов, без снижения производительности.

1.2.3 Функциональные особенности разрабатываемого ПО

Основным функционалом системы является управление задачами и проектами. Для каждого проекта предусмотрена возможность создания иерархической структуры задач, где каждая из них может быть разделена на подзадачи с указанием сроков, приоритетов и ответственных лиц. Такая детализация обеспечивает логичную и удобную организацию рабочего процесса. Возможность редактирования задач позволяет оперативно вносить изменения, что важно для динамичного проектного управления.

Для защиты данных реализована ролевая модель доступа, позволяющая назначать права в зависимости от должностных обязанностей пользователей. Это обеспечивает высокий уровень безопасности и предотвращает несанкционированный доступ к информации.

1.3 Анализ существующих инструментов для управления проектами

Для определения оптимального решения для компании «ВорлдИнтерТех Рус» проведён анализ популярных систем управления проектами: Trello[19], Jira[15] и Asana[12]. Эти платформы активно используются для планирования, управления задачами и координации работы

команд. Каждая из них обладает уникальными архитектурными особенностями, преимуществами и недостатками.

Trello - инструмент на основе канбан-досок, разработанный в 2011 году и позже приобретённый Atlassian. Архитектура построена на клиент-серверной модели, с доступом через веб и мобильные приложения. Интерфейс интуитивно понятен, поддерживается назначение задач, дедлайны, комментарии и интеграции с внешними сервисами. Подходит для небольших команд, но ограниченные аналитические возможности и отсутствие иерархии задач делают Trello малоэффективным для крупных и сложных проектов.

Jira, созданная Atlassian в 2002 году, изначально предназначалась для отслеживания ошибок, но позже трансформировалась в мощную систему управления проектами с поддержкой Scrum и Kanban. Поддерживает облачное и локальное развертывание. Предоставляет гибкую систему задач и подзадач, а также встроенные аналитические инструменты. Подходит для крупных команд и комплексных проектов, но сложный интерфейс и перегруженность функциями могут затруднить освоение.

Asana, разработанная в 2008 году, представляет собой облачную платформу с клиент-серверной архитектурой. Обеспечивает визуальное управление задачами, подзадачами, календарями и досками. Отличается современным интерфейсом и автоматизацией процессов. Подходит для команд среднего размера, но ограниченные средства аналитики и слабая поддержка Agile-подходов снижают её применимость в управлении крупными проектами.

Для объективного сравнения Trello, Jira и Asana был выполнен анализ их функционала и соответствия требованиям отдела разработки. Основная цель заключалась в выявлении их преимуществ и ограничений в контексте задач компании «ВорлдИнтерТех Рус». Полученные результаты в таблице 1, позволят обосновать необходимость создания собственной системы управления проектами, которая будет учитывать специфические потребности компании и обеспечивать максимальную эффективность работы.

Таблица 1 – Сравнительный анализ готовых решений по разработанным требованиям

Требование	Инструменты управления проектами		
	trello	jira	asana
Функциональные требования	Подходит для простых проектов. Ограниченная поддержка сложных иерархий задач.	Поддерживает сложные проекты, иерархии задач и подзадач. Хорошо интегрируется с разработческими инструментами.	Подходит для проектов средней сложности. Ограниченная гибкость в кастомизации рабочих процессов
Удобство использования	Интуитивно понятный интерфейс, легко освоить.	Интерфейс сложный для новичков, требуется обучение.	Удобный интерфейс, легко освоить даже новым пользователям.
Надёжность	Надёжна для небольших команд, но неустойчива при больших нагрузках.	Высокая надёжность и отказоустойчивость для крупных проектов.	Достаточно надёжна, но возможны проблемы при высоких нагрузках.
Производительность	Быстрая работа для небольших команд. Замедляется при больших объёмах данных.	Отличная производительность для сложных и крупных проектов.	Хорошая производительность для средних команд.
Сопровождаемость	Легко поддерживать и обновлять, но ограничена в расширении функций.	Гибкая настройка и поддержка плагинов, но сложна в обслуживании.	Регулярные обновления, но ограниченная возможность кастомизации.
Безопасность	Базовая защита данных, минимальное разграничение прав доступа.	Высокий уровень безопасности, гибкая ролевая модель доступа.	Хорошие возможности для управления доступом, но ниже, чем у Jira.
Ролевая модель доступа	Простая модель с базовыми ролями и минимальными настройками доступа.	Гибкая и детализированная модель, позволяющая тонко настраивать права доступа для разных уровней и задач.	Поддержка стандартных ролей с ограниченными настройками прав доступа.

Продолжение таблицы 1

Требование	Инструменты управления проектами		
	trello	jira	asana
Интеграция	Поддерживает интеграцию с популярными инструментами, такими как Slack и Google Drive.	Широкие возможности интеграции с инструментами разработки и другими сервисами	Легко интегрируется с основными корпоративными сервисами, такими как Slack, Microsoft Teams и Google Drive.
Масштабируемость	Подходит для небольших команд и проектов. При росте количества задач и пользователей может начать тормозить.	С лёгкостью выдерживает высокие нагрузки и масштабируется для крупных команд и сложных проектов.	Отлично справляется с потребностями средних команд. Но при значительном увеличении нагрузки могут возникнуть сложности.

На основе проведенного анализа можно сделать выводы о том, что ни одна из рассмотренных систем не обеспечивает полного соответствия требованиям компании в части гибкости и адаптивности.

Специфика бизнес-процессов компании «ВорлдИнтерТех Рус» требует интеграции с внутренними системами, гибкой настройки под корпоративные процессы и надёжного разграничения прав доступа - что невозможно реализовать в полном объёме с помощью готовых решений. Кроме того, использование сторонних систем сопровождается высокими затратами на подписку, особенно при масштабировании.

Разработка собственной системы позволяет снизить долгосрочные расходы, устранить зависимость от лицензионных ограничений и адаптировать функциональность под текущие задачи без дополнительных затрат.

В отличие от готовых решений, собственное ПО может развиваться вместе с компанией - дополняться новыми модулями, адаптироваться под

изменяющиеся требования и масштабироваться при росте бизнеса. Это обеспечивает гибкость, технологическую независимость и устойчивость к внешним ограничениям.

Таким образом, создание собственной системы управления проектами, позволит повысить эффективность управления, сократить издержки и обеспечить устойчивую цифровую поддержку ключевых процессов компании.

1.4 Требования к функционалу и интерфейсу системы

Для определения требований к системе управления проектами использована методология FURPS+, обеспечивающая структурированный подход к учёту функциональных и нефункциональных характеристик разрабатываемого программного обеспечения [18].

Функциональные требования:

а) управление пользователями:
   1) регистрация: администратор должен иметь возможность создавать учетные записи,
   2) аутентификация: пользователи должны иметь возможность входить в систему,
   3) управление ролями: администраторы должны иметь возможность назначать роли и права доступа для различных пользователей;

б) управление проектами:
   1) система должна предоставить возможность создать, редактировать и удалять проект,
   2) назначение сроков проекта,
   3) поддержка структуризации проекта на этапы и задачи,
   4) возможность добавить проект в список избранных,
   5) возможность посмотреть список последних открытых проектов;

в) управление задачами:

1) система должна предоставить возможность создать, редактировать и удалять задачу,
2) возможность добавлять ответственных сотрудников на задачу,
3) назначение сроков задачи,
4) назначение приоритетности задачи,
5) возможность создать подзадачи к задаче,
6) возможность оставлять комментарии к задаче;

г) управление участниками проекта:

1) возможность добавлять и удалять участников проекта,
2) возможность посмотреть список участников проекта;

д) управление аккаунтом пользователя:

1) просмотр личных данных пользователя,
2) возможность изменить аватарку пользователя,
3) возможность выйти из аккаунта пользователя;

е) учёт ресурсов сотрудников:

1) возможность посмотреть полный список сотрудников, отсортированный по специальности,
2) возможность получить контактные данные сотрудника,
3) возможность получить информация о том, в каких проектах задействован сотрудник и сколько имеет в них задач;

ж) отчетность:

1) генерация отчета по проектам и задачам;

з) интеграция с другими системами:

1) поддержка API для взаимодействия с внешними приложениями, такими как почта, календарь и другими.

Нефункциональные требования определяют качественные характеристики и ограничения системы, влияя на удобство её использования, надежность, производительность, безопасность и удобство сопровождения.

а) удобство использования:

1) интуитивный интерфейс: простота использования интерфейса для сокращения времени обучения,
2) навигация и поиск: удобная система поиска и навигации для быстрого доступа к функциям;

б) надежность:

1) высокая доступность: обеспечение доступности системы не менее 95%, с максимально допустимым простоем не более 1-2 часов в месяц,
2) автоматическое сохранение изменений: реализация автоматического сохранения всех изменений для предотвращения потери данных;

в) безопасность:

1) ролевая модель доступа: разграничение прав доступа на основе роли пользователя в проекте,
2) шифрование данных: защита данных через шифрование при хранении и передаче;

г) производительность:

1) скорость обработки: оперативное выполнение запросов пользователей за 1 секунду или менее при стандартной нагрузке,
2) масштабируемость: поддержка работы до 100 активных пользователей одновременно без снижения производительности;

д) сопровождаемость:

1) модульная структура кода: код разделён на модули для упрощения внесения изменений,
2) обновляемость: обновления могут внедряться без остановки работы системы,
3) масштабируемая архитектура: архитектура позволяет легко адаптироваться к увеличению нагрузки.

Такой подход к формулированию требований обеспечивает баланс между функциональностью, удобством и надёжностью системы, что напрямую влияет на её эффективность и соответствие задачам компании.

Интерфейс должен быть интуитивно понятным и не требовать длительного освоения. При его проектировании учитывались принципы доступности, логичности и визуальной простоты. В основе дизайна разделение на функциональные блоки, понятные элементы управления и удобная навигация, что способствует комфортной и продуктивной работе пользователя.

Визуализация требований в таблице 2 помогает наглядно отразить ключевые аспекты, необходимые для создания удобного и эффективного программного продукта.

Таблица 2 – Функциональные и нефункциональные требования к системе

Функциональные требования	Не функциональные
Регистрация пользователей	Интуитивный интерфейс
Аутентификация пользователей	Навигация и поиск
Управление проектами	Высокая доступность
Управление ролями	Автоматическое сохранение изменений
Управление задачами	Ролевая модель доступа
Управление участниками проекта	Шифрование данных
Управление аккаунтом пользователя	Скорость обработки
Учет ресурсов сотрудников	Масштабируемость
Создание отчетов	Модульная структура кода
Интеграция с другими системами	Обновляемость

«Визуальное моделирование в UML можно представить как некоторый процесс постепенного спуска от наиболее обшей и абстрактной концептуальной модели исходной системы к логической, а затем и к физической модели соответствующей программной системы. Для достижения

этих целей вначале строится модель в форме так называемой диаграммы вариантов использования (use case diagram), которая описывает функциональное назначение системы или, другими словами, то, что система будет делать в процессе своего функционирования» [3].

На диаграмме выделены три основные роли: администратор, руководитель и сотрудник. Каждая роль имеет свои сценарии работы с системой, которые обеспечивают выполнение соответствующих задач. Диаграмма представлена на рисунке 5.

Рисунок 5 – Диаграмма прецедентов

Администратор обладает максимальными правами в системе и осуществляет полный контроль над проектами и задачами. Он создаёт и удаляет проекты, регистрирует пользователей, назначает исполнителей, редактирует параметры задач и формирует отчёты. Администратор управляет всеми комментариями и имеет доступ к информации о доступности сотрудников.

Руководитель работает только с созданными им или назначенными ему проектами. Он контролирует выполнение задач, управляет их содержанием, формирует отчёты и координирует действия команды. Его полномочия ограничены рамками конкретных проектов.

Сотрудник взаимодействует исключительно с задачами, назначенными ему в рамках проектов, в которых он принимает участие. После авторизации он получает доступ к своим задачам в проектах, отслеживает их статус, выполняет работу в установленные сроки и оставляет комментарии.

Использование связей типа «include» и «extend» позволяет гибко отразить зависимости и расширяемость функционала, обеспечивая чёткое распределение ролей и обязанностей в системе.

Вывод по главе 1.

В первой главе обоснована необходимость разработки собственной системы управления проектами для компании «ВорлдИнтерТех Рус». Проведён анализ существующих решений, выявлены их ограничения и несоответствие требованиям бизнеса. Разработана архитектурная модель будущей системы, основанная на современных веб-технологиях, обеспечивающих масштабируемость, отказоустойчивость и интеграцию с корпоративной инфраструктурой. Сформулированы функциональные и нефункциональные требования с учётом модели FURPS+, а также определены роли пользователей. Полученные результаты стали основой для дальнейшего проектирования и реализации программного обеспечения, соответствующего целям компании.

Глава 2 Проектирование программного обеспечения

2.1 Методология проектирования программного обеспечения

Проектирование ПО - ключевой этап разработки, на котором определяется структура системы, состав компонентов, их взаимодействие и логика обработки данных. Для описания архитектуры системы управления проектами использовано визуальное моделирование на языке UML.

«UML-диаграмма – это специализированный язык графического описания, предназначенный для объектного моделирования в сфере разработки различного программного обеспечения. Данный язык имеет широкий 61 профиль и представляет собой открытый стандарт, в котором используются различные графические обозначения, чтобы создать абстрактную модель системы. UML создавался для того, чтобы обеспечить определение, визуализацию, документирование, а также проектирование всевозможных программных систем» [1].

Визуальное моделирование позволяет формально представить как статическую структуру, так и динамическое поведение системы, повышая точность проектных решений и улучшая командную коммуникацию, снижая риск ошибок при реализации.

В рамках данной работы были использованы следующие диаграммы:

- диаграмма классов;
- концептуальная модель базы данных;
- логическая модель базы данных;
- физическая модель базы данных;
- диаграмма последовательности;
- диаграмма состояний;
- диаграмма компонентов;
- диаграмма развертывания;
- диаграмма структуры интерфейса.

Использование диаграмм, перечисленных выше, обеспечивает наглядное и всестороннее представление системы, отражающее её архитектуру, логику и поведение при работе.

2.2 Структурное моделирование

Для эффективной разработки программного обеспечения важно не только определить, что должно делать приложение, но и как оно будет организовано изнутри. С этой целью в рамках проектирования архитектуры была построена диаграмма классов.

Диаграмма классов, представленная на рисунке 6, отражает основные сущности предметной области, определяет их атрибуты, методы, а также взаимосвязи между ними.

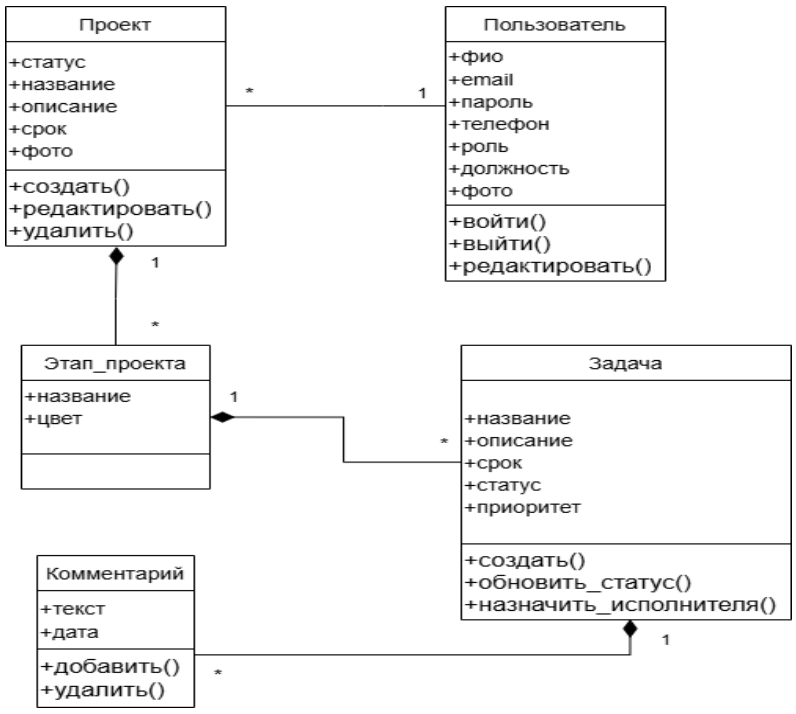

Рисунок 6 – Диаграмма классов

Класс «Проект» описывает ключевые параметры проекта: название, статус, срок, описание и изображение. Он связан с классами «Этап_проекта»

и «Пользователь», что позволяет учитывать структуру проекта и закреплять ответственных. Методы класса включают создание, редактирование и удаление проекта.

Класс «Пользователь» содержит ФИО, email, пароль, контактные данные, роль и аватар. Реализованы методы авторизации, выхода и редактирования профиля.

Класс «Этап_проекта» отражает структуру проекта, включает название и цвет этапа, связан с задачами и проектом, формируя иерархию исполнения.

Класс «Задача» хранит сведения о названии, описании, сроке, приоритете и статусе задачи. Методы позволяют создавать задачи, изменять статус и назначать исполнителя.

Класс «Комментарий» фиксирует текстовые замечания к задачам с указанием даты. Поддерживаются добавление и удаление комментариев.

Связи между классами описывают ассоциации между сущностями: один пользователь может выполнять несколько задач, задачи относятся к этапам, а этапы к конкретному проекту.

«Концептуальная база данных – информационная модель предметной области банка данных. Это описание с помощью набора абстракций структуры всех данных, необходимых для решения всего комплекса задач в рамках предметной области конкретного банка данных» [9]. Концептуальная модель представлена на рисунки 7.

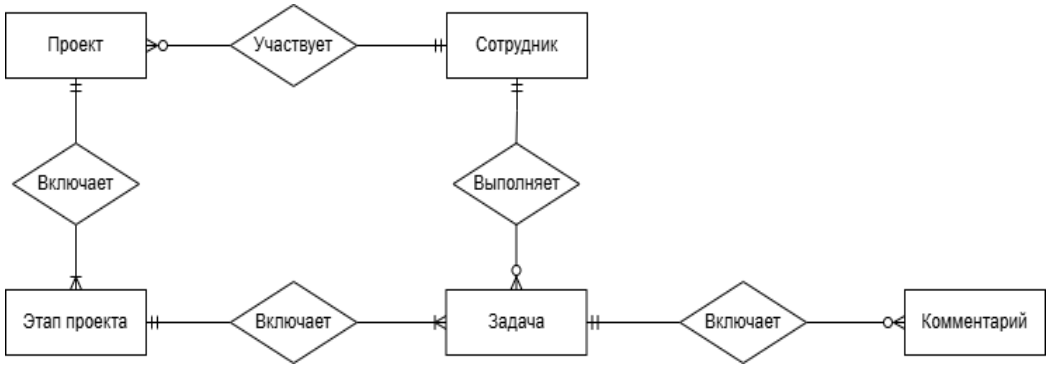

Рисунок 7 – Концептуальная модель базы данных

На диаграмме представлены следующие сущности и их взаимосвязи:

- проект – центральная сущность, которая может состоять из нескольких этапов. Связь "Включает" связывает проект с его этапами;
- этап проекта – сущность, детализирующая процесс выполнения проекта. Каждый этап может содержать несколько задач, что представлено связью "Включает";
- задача – сущность, описывающая конкретное действие или работу в рамках этапа проекта. Задача может быть назначена одному или нескольким сотрудникам, что отображается через связь "Выполняет";
- сотрудник – сущность, представляющая пользователей системы. Сотрудники связаны с проектами связью "Участвует"; могут выполнять задачи, что показано через связь "Выполняет";
- комментарий – сущность, связанная с задачей. Она позволяет фиксировать дополнительную информацию по задачам, что способствует улучшению коммуникации и документирования работы.

Диаграмма демонстрирует структуру данных и логику взаимодействия между элементами системы, что создает фундамент для построения эффективной базы данных и реализации функциональности системы.

«При проектировании базы данных на первом этапе создается ее начальный прототип – логическая модель данных. Логическая модель позволяет описать понятия в рамках данной предметной области, взаимосвязь между ними, а также отразить ограничения, связанные с их конкретным применением. Концептуальный выбор логической модели данных предопределяет уровень эффективности той или иной программной реализации базы данных» [2]. В данной работе создана модель, основанная на документно-ориентированном подходе, обеспечивающем гибкость и масштабируемость.

Вместо традиционных таблиц используются коллекции, содержащие документы в формате JSON. Такой формат упрощает хранение вложенных и связанных данных, ускоряет доступ к информации и легко адаптируется к изменениям в бизнес-логике. Это особенно эффективно для систем управления проектами, где важны структурированность, прозрачность и гибкость хранения.

На рисунке 8 представлена логическая модель базы данных, включающая коллекции сотрудников, проектов, этапов, задач и комментариев. Эти сущности отражают ключевые объекты системы и формируют основу взаимодействия между пользователями и бизнес-процессами. Структура коллекций оптимизирована для уменьшения избыточности и упрощения управления данными.

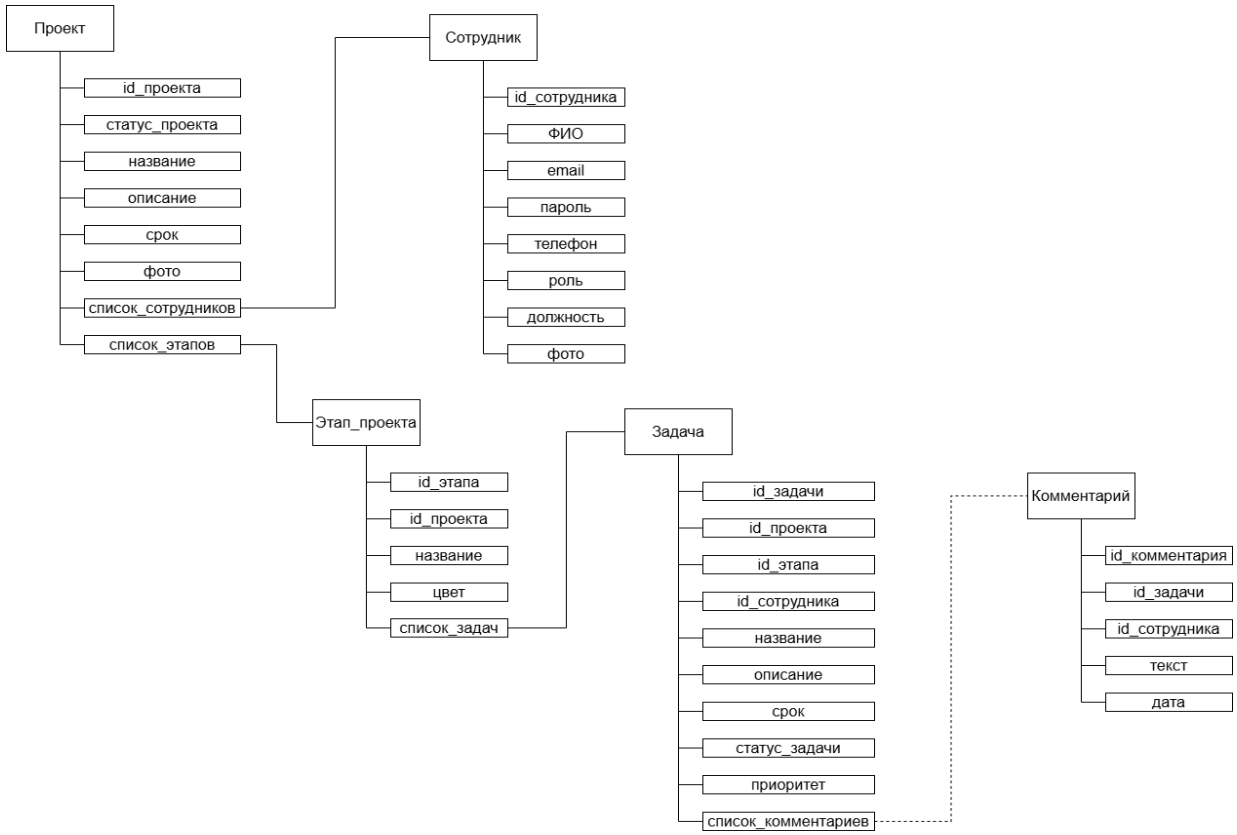

Рисунок 8 – Логическая модель базы данных

Логическая модель базы данных отражает ключевые сущности системы управления проектами и их взаимосвязи. Она демонстрирует иерархию данных: от проектов и их этапов до задач и связанных с ними комментариев. Структура учитывает участие сотрудников в проектах и задачах, а также фиксацию обратной связи в процессе работы.

Модель построена с учётом документно-ориентированного подхода, что обеспечивает логичную организацию данных, минимизирует избыточность и упрощает навигацию по взаимосвязанным элементам. Диаграмма служит основой для построения базы данных, оптимизированной под требования гибкости, масштабируемости и прозрачности.

Для хранения данных была выбрана MongoDB, гибкая NoSQL-платформа. Анализ MongoDB и её альтернатив приведён в таблице 3.

Таблица 3 – Сравнительный анализ платформ для работы с базой данных

Критерий	Платформа		
	MongoDB	PostgreSQL	Firebase Firestore
Тип базы данных	NoSQL	Реляционная	NoSQL
Гибкость структуры	Высокая	Низкая	Средняя
Масштабируемость	Горизонтальная	Вертикальная	Горизонтальная
Удобство интеграции	Высокое	Среднее	Высокое

MongoDB была выбрана за её гибкость в работе с динамической структурой данных, лёгкость интеграции с Node.js и горизонтальную масштабируемость. PostgreSQL, хотя и мощная реляционная база данных, менее удобна для динамичных данных. Firestore прост в использовании, но его функциональность ограничена для реализации сложных запросов [14].

«Физическое проектирование. Процесс подготовки описания реализации базы данных на вторичных запоминающих устройствах; на этом этапе рассматриваются основные отношения, организация файлов и индексов, предназначенных для обеспечения эффективного доступа к данным, а также все связанные с этим ограничения целостности и средства защиты» [6]. При её

создании используется логическая структура, адаптированная с учётом особенностей выбранной СУБД, форматов данных и технических ограничений. Физическая модель представлена на рисунке 9, она включает в себя определение типов данных, индексов и ключевых связей, необходимых для оптимальной работы приложения.

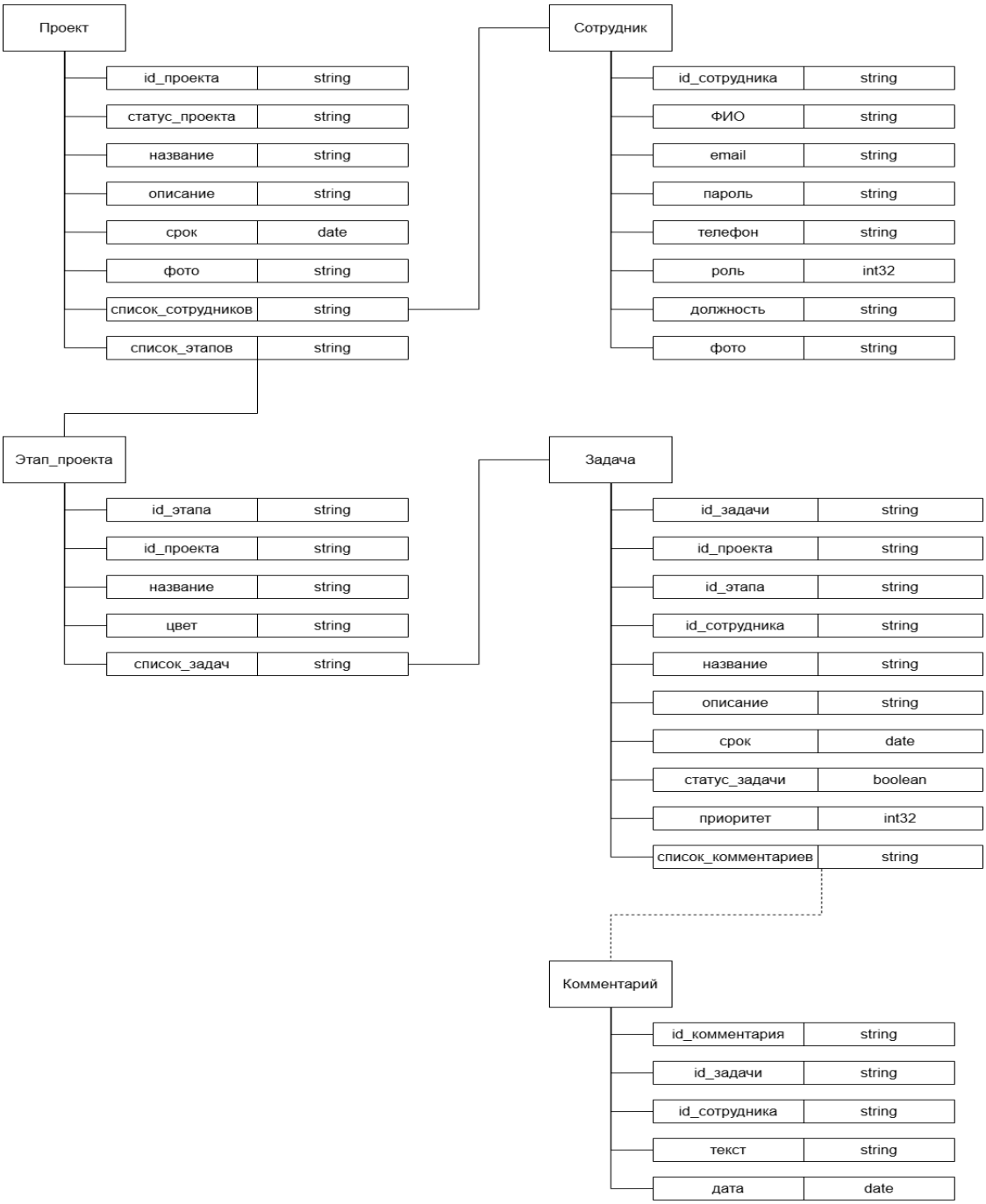

Рисунок 9 – Физическая схема базы данных

Физическая модель базы данных представляет собой реализацию структуры данных, включая типы данных и схемы коллекций.

Коллекция «Проект» включает данные о проектах:

- id_проекта - уникальный идентификатор проекта, используется для связи с другими коллекциями, имеет тип данных string;
- список сотрудников – сотрудники ответственные за проект, связывает проект с конкретным пользователем, тип данных string;
- название - название проекта, отображается в интерфейсе для идентификации, тип данных string;
- описание - текстовое описание проекта, содержит ключевую информацию, тип данных string;
- срок - срок выполнения проекта, задается датой завершения, тип данных date;
- статус_проекта - статус текущего состояния проекта, тип данных string;
- фото - изображение, связанное с проектом, используется для визуализации, тип данных string;
- список_этапов – список этапов на которые делится проект, тип данных string.

Эта коллекция хранит основную информацию о проектах и связывает другие коллекции.

Коллекция «Сотрудник» содержит информацию о пользователях системы:

- id_сотрудника - уникальный идентификатор сотрудника, используется для идентификации, тип данных string;
- ФИО - полное имя сотрудника, отображается в интерфейсе, тип данных string;
- email - адрес электронной почты для авторизации и уведомлений, тип данных string;

- пароль - зашифрованный пароль для защиты данных, тип данных string;
- телефон - контактный номер сотрудника, тип данных string;
- роль - уровень доступа или роль сотрудника в системе, тип данных int32;
- должность - позиция сотрудника в компании, тип данных string;
- фото - изображение профиля сотрудника, тип данных string.

Коллекция обеспечивает данные для разграничения доступа и взаимодействия сотрудников.

Коллекция «Этап проекта» описывает этапы выполнения проектов:

- id_этапа - уникальный идентификатор этапа, тип данных string;
- id_проекта - идентификатор связанного проекта, связывает этап с конкретным проектом, тип данных string;
- название - название этапа, используется для отображения и идентификации, тип данных string;
- цвет - цветовое обозначение этапа для визуализации в интерфейсе, тип данных string;
- список_задач – задачи, которые хранит данный этап, тип данных string

Коллекция связана с задачами и проектами.

Коллекция «Задача» хранит информацию о задачах:

- id_задачи - уникальный идентификатор задачи, тип данных string;
- id_проекта - идентификатор связанного проекта, связывает задачу с проектом, тип данных string;
- id_этапа - идентификатор связанного этапа, структурирует задачи по этапам, тип данных string;
- id_сотрудника - идентификатор ответственного сотрудника, тип данных string;
- название - название задачи, отображается в интерфейсе, тип данных string;

- описание - текстовое описание задачи, тип данных string;
- срок - дата завершения задачи, тип данных date;
- статус_задачи - текущий статус выполнения задачи, тип данных boolean;
- приоритет - уровень приоритета задачи, тип данных int32.
- Список_комментариев – комментарии, которые пользователи оставили к задаче, тип данных string

Коллекция обеспечивает четкую организацию и структурирование задач в системе.

Коллекция «Комментарий» хранит комментарии пользователей:

- id_комментария - уникальный идентификатор комментария, тип данных string;
- id_задачи - идентификатор задачи, к которой относится комментарий, тип данных string;
- id_сотрудника - идентификатор автора комментария, тип данных string;
- текст - содержание комментария, тип данных string;
- дата - дата и время создания комментария, тип данных date.

Коллекция обеспечивает функционал для взаимодействия и обсуждения задач.

Каждая коллекция была разработана с учетом её функционального назначения, что обеспечило упорядоченную организацию данных и их взаимосвязей.

Использование коллекций, таких как «Проект», «Сотрудник», «Этап проекта», «Задача» и «Комментарий», позволило создать гибкую и масштабируемую систему, поддерживающую основные процессы управления проектами, задачами и взаимодействия сотрудников. Заданные типы данных и поля обеспечивают хранение всей необходимой информации и полностью соответствуют требованиям системы.

Разработанная физическая модель базы данных является основой для реализации системы, обеспечивая её высокую производительность, безопасность и удобство эксплуатации.

Для описания взаимодействия компонентов системы при создании нового проекта была разработана диаграмма последовательности, представленная на рисунке 10. Она позволяет наглядно отразить порядок вызова операций, участие различных компонентов в процессе, а также показать поток данных между клиентской частью, сервером и базой данных.

Рисунок 10 – Диаграмма последовательности «Создания проекта»

На диаграмме отражён сценарий создания проекта пользователем в веб-приложении:

- пользователь заполняет форму создания проекта в клиентском интерфейсе;
- клиентское приложение формирует и отправляет POST-запрос на сервер по маршруту /project/create;

- серверное приложение принимает запрос, выполняет валидацию входных данных и передаёт их в базу данных;
- база данных сохраняет проект и возвращает результат операции;
- сервер передаёт клиенту ответ об успешном создании;
- пользователь автоматически перенаправляется к доске созданного проекта.

Такое представление упрощает понимание распределения ответственности между компонентами и помогает при реализации бизнес-логики на этапе программирования.

Для понимания жизненного цикла задач в системе была разработана диаграмма состояний, представленная на рисунке 11. «UML State Machine Diagram (Диаграмма состояний) - это диаграмма из разряда поведенческих, которая показывает то, какие состояния может иметь объект моделирования (то, состояния чего мы рисуем) и как они меняются в течение жизни этого самого объекта» [11].

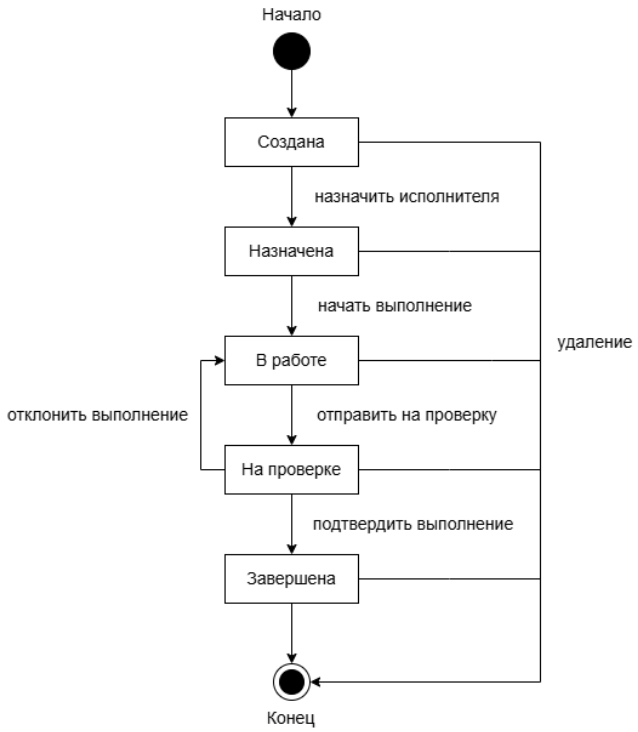

Рисунок 11 – Диаграмма состояний «Задачи»

На диаграмме отображены следующие состояния задачи:

- создана - начальное состояние после регистрации задачи в системе;
- назначена - исполнитель назначен, задача ожидает начала выполнения;
- в работе - задача выполняется сотрудником;
- на проверке - исполнитель завершил выполнение и отправил задачу на проверку;
- завершена - задача успешно проверена и закрыта.

Также предусмотрены альтернативные переходы:

- из состояния «На проверке» задача может быть отклонена, после чего возвращается обратно в состояние «В работе» для доработки;
- из любого состояния задача может быть удалена, что завершает её жизненный цикл преждевременно.

Переход между состояниями осуществляется на основе действий пользователя или системы.

2.3 Выбор технологий и инструментов для разработки

2.3.1 Языки программирования

В процессе разработки системы управления проектами был проведён выбор языков программирования, соответствующих техническим требованиям и целям проекта. Выбор сопровождался сравнением языков по ключевым критериям, результаты приведены в таблице 4.

Таблица 4 – Сравнительный анализ языков программирования для клиентской части

Критерий	Языки программирования		
	JavaScript	Dart	Python
Цель использования	Разработка интерфейса	Используется с Flutter	Общего назначения

Продолжение таблицы 4

Критерий	Языки программирования		
	JavaScript	Dart	Python
Производительность	Высокая	Средняя	Средняя
Совместимость с браузерами	Поддерживается всеми браузерами	Ограничена	Нужны дополнительные библиотеки
Удобство синтаксиса	Простота и универсальность	Легкий для мобильной разработки	Интуитивно понятный
Популярность	Широкое распространение	Узкоспециализированный	Популярный для обработки данных

JavaScript выбран для клиентской части благодаря универсальности и полной поддержке всеми современными браузерами. Он обеспечивает быструю разработку интерфейсов. Dart, хотя и эффективен с Flutter, не предназначен для нативной работы в браузере. Python уступает JavaScript по производительности и гибкости в контексте веб-разработки.

В таблице 5 представлен результат сравнения языков серверной части.

Таблица 5 – Сравнительный анализ языков программирования для серверной части

Критерий	Языки программирования		
	TypeScript	Ruby	Go
Цель использования	Серверная часть	Серверная часть	Высокопроизводительные серверы
Производительность	Высокая	Средняя	Высокая
Строгая типизация	Есть	Нет	Есть
Удобство синтаксиса	Прост для JS-разработчиков	Легкий и интуитивный	Сложный
Популярность	Широкая поддержка в экосистеме JS	Высокая	Растет в кругах высоконагруженных проектов

TypeScript стал оптимальным выбором для серверной части за счёт строгой типизации и интеграции с JavaScript. Он позволяет избежать множества ошибок и упрощает поддерживаемость кода. Ruby, хотя и удобен,

не отвечает требованиям производительности. Go, несмотря на свою скорость, требует больше времени на изучение и внедрение в проект.

2.3.2 Фреймворки и инструменты

Выбор фреймворков и инструментов для разработки системы управления проектами определяет эффективность разработки, удобство сопровождения и производительность конечного продукта.

Анализ преимуществ и недостатков фреймворков для клиентской части представлен в таблице 6.

Таблица 6 – Сравнительный анализ фреймворков для клиентской части

Критерий	Фреймворки		
	React	Angular	Vue.js
Простота использования	Высокая	Низкая	Высокая
Производительность	Высокая	Средняя	Высокая
Экосистема	Широкая	Широкая	Средняя
Масштабируемость	Высокая	Высокая	Средняя

React стал оптимальным выбором за счёт его компонентного подхода, высокой производительности из-за виртуального DOM и широкой экосистеме, которая поддерживает интеграцию с различными инструментами [17]. Angular был отклонён из-за сложности, не оправдывающей себя для нашего проекта, а Vue.js уступил React в популярности и масштабируемости.

Для выбора подходящего фреймворка для серверной части системы был проведён сравнительный анализ, представленный в таблице 7.

Таблица 7 – Сравнительный анализ фреймворков для серверной части

Критерий	Фреймворки		
	Nest.js	Express.js	Koa.js
Архитектура	Модульная	Отсутствует	Минималистичная

Продолжение таблицы 7

Критерий	Фреймворки		
	Nest.js	Express.js	Koa.js
Поддержка TypeScript	Встроенная	Частичная	Частичная
Удобство разработки	Высокое	Среднее	Низкое
Ширина возможностей	Высокая	Среднее	Средняя

Nest.js был выбран так как имеет модульную архитектуре, встроенную поддержке TypeScript и наличие широкого набора инструментов. Express.js не предоставляет строгой архитектуры, усложняя поддержку крупных проектов, а Koa требует значительных усилий для настройки.

Для стилизации интерфейса использовался SCSS. Сравнение SCSS и его альтернатив (LESS и Tailwind CSS) представлено в таблице 8.

Таблица 8 – Сравнительный анализ инструментов для стилизации

Критерий	Инструмент		
	SCSS	LESS	Tailwind CSS
Распространённость	Высокая	Отсутствует	Высокая
Вложенность	Поддерживается	Поддерживается	Не поддерживается
Подход к стилизации	Традиционный	Традиционный	Утилитарный
Гибкость	Высокая	Высокая	Низкая

SCSS был выбран за его широкую поддержку, популярность и традиционный подход к стилизации, который идеально интегрируется с экосистемой React. LESS уступает в распространённости, а Tailwind CSS требует значительных изменений в подходе к написанию стилей.

Использование React, Nest.js и SCSS в проекте обеспечило баланс между производительностью, удобством сопровождения и гибкостью разработки.

Эти инструменты оказались наиболее подходящими для реализации системы управления проектами, учитывая её требования и специфику.

2.3.3 Платформы для разработки

При проектировании и реализации СУП важным этапом стал выбор платформ для разработки. Учитывались критерии совместимости с используемыми языками и инструментами, удобство для команды разработчиков, производительность и возможность масштабирования.

Для серверной части была выбрана платформа Node.js. Сравнение Node.js и её альтернатив приведено в таблице 9.

Таблица 9 – Сравнительный анализ платформ для серверной части

Критерий	Платформа		
	Node.js	Django	Ruby on Rails
Производительность	Высокая	Средняя	Средняя
Поддержка асинхронности	Полная	Ограниченная	Ограниченная
Экосистема	Богатая	Средняя	Средняя
Гибкость	Высокая	Средняя	Низкая

Node.js обеспечила высокую производительность из-за асинхронной архитектуры и широкую поддержку за счёт богатой экосистемы библиотек [16]. Django предлагает встроенные решения для многих задач, но требует больше ресурсов для настройки сложных API. Ruby on Rails удобен для быстрого прототипирования, но не предоставляет той же гибкости и скорости.

Выбор Node.js в связке с MongoDB позволил достичь необходимой производительности, гибкости и масштабируемости. Эта платформа эффективно обрабатывает данные под нагрузкой и соответствует техническим требованиям проекта.

2.3.4 Паттерны разработки

В разработке серверной части системы управления проектами был применён паттерн Dependency Injection, обеспечивающий гибкость, модульность и удобство поддержки кода. Вместо создания зависимостей

внутри компонентов, они передаются извне, что упрощает архитектуру и повышает тестируемость.

В проекте Dependency Injection реализован средствами фреймворка Nest.js, который предоставляет встроенные механизмы для управления модулями и сервисами. Такой подход позволил сократить дублирование кода, упростить замену компонентов и повысить прозрачность структуры приложения.

2.4 Архитектура и взаимодействие компонентов

«Диаграмма компонентов описывает особенности физического представления системы. Определяет архитектуру разрабатываемой системы. Устанавливает зависимости между программными компонентами: исходный, бинарный и исполняемый код. Во многих средах разработки модуль или компонент соответствует файлу» [8]. Диаграмма компонентов системы управления проектами представлена на рисунке 12.

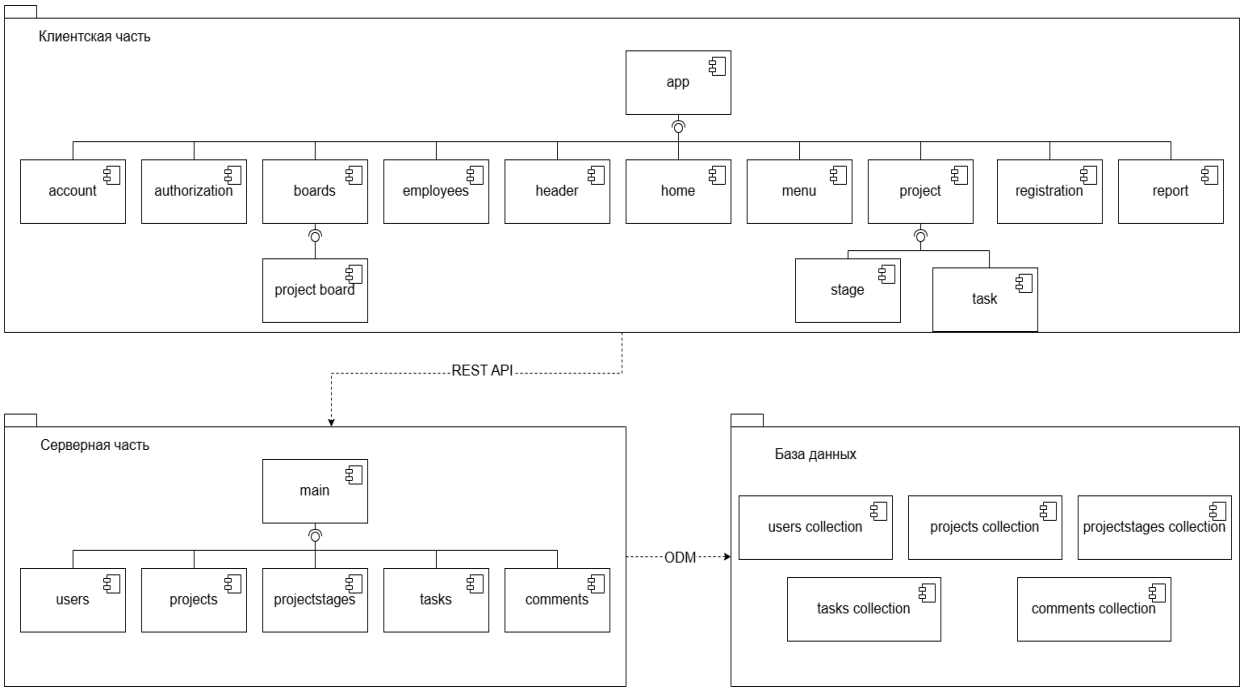

Рисунок 12 – Диаграмма компонентов системы управления проектами

Диаграмма компонентов системы управления проектами демонстрирует её трёхуровневую архитектуру: клиентская часть, сервер и база данных. Такая структура обеспечивает модульность, гибкость разработки и удобство сопровождения.

Клиентская часть реализована как приложение на React.js. Основной компонент App управляет маршрутизацией и организацией пользовательских интерфейсов. Функциональные компоненты сгруппированы по логическим разделам. Project, stage и task обеспечивают работу с проектами и задачами, включая создание, редактирование и удаление. Компоненты PopupCreateProject, PopupCreateTask и другие всплывающие окна реализуют взаимодействие с сервером через REST API, обеспечивая выполнение CRUD-операций.

Серверная часть построена на Nest.js и структурирована в виде модулей. Компонент Main, связывает модули и управляет маршрутизацией запросов. Каждый модуль включает контроллеры, сервисы и DTO, что упрощает обработку запросов, реализацию бизнес-логики и валидацию данных.

Данные хранятся в нереляционной базе данных. Структура построена на коллекциях, отражающих основные сущности системы. Через Mongoose обеспечивается преобразование между объектами TypeScript и JSON-документами, что сохраняет целостность и типизацию данных.

Взаимодействие между уровнями осуществляется через REST API. Клиент отправляет запросы, сервер обрабатывает их и обращается к базе данных, после чего возвращает результаты в формате JSON. Это позволяет поддерживать актуальность интерфейса и обеспечивать синхронную работу компонентов.

«Диаграмма развёртывания является физической диаграммой в языке UML. Она отображает физические взаимосвязи между программными и аппаратным компонентами проектируемой системы. Корпоративные приложения часто требуют для своей работы некоторой ИТ-инфраструктуры, хранят информацию в базах данных, расположенных где-то на серверах

компании, вызывают веб-сервисы, используют общие ресурсы и т. д. В таких случаях полезно иметь графическое представление инфраструктуры, на которую будет развернуто приложение» [5].

Развертывание системы управления проектами требует тщательного планирования для обеспечения её доступности, производительности и безопасности. Основные компоненты системы, их взаимодействие и особенности размещения демонстрируется на диаграмме, представленной на рисунке 13.

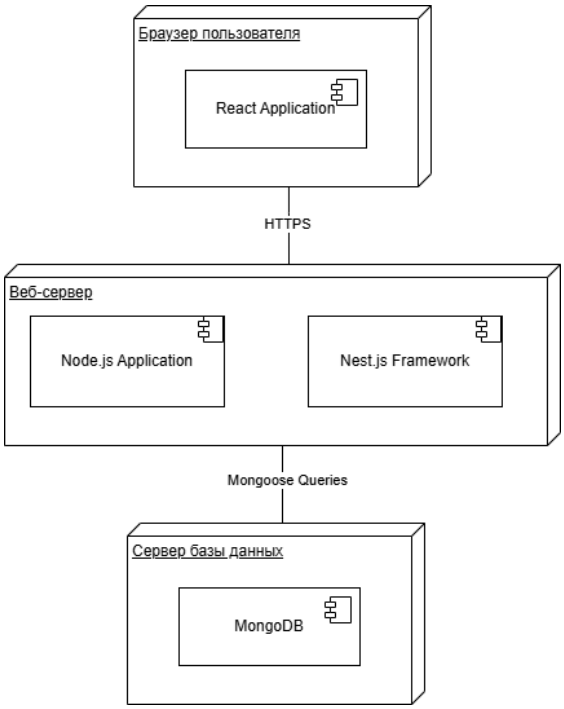

Рисунок 13 - Диаграмма развертывания системы управления проектами

Диаграмма развертывания отражает трёхуровневую архитектуру системы, включающую клиентское приложение, серверную часть и базу данных.

Клиентская часть представляет собой React-приложение, выполняемое в браузере и обменивающееся данными с сервером по HTTPS. Интерфейс управляет задачами, проектами и пользователями, отправляя JSON-запросы и обновляя данные в реальном времени.

Сервер на Nest.js реализует бизнес-логику, обрабатывает клиентские запросы и взаимодействует с базой данных через Mongoose. MongoDB используется как хранилище JSON-документов, а Mongoose обеспечивает преобразование данных и сохранение их структуры.

Компоненты системы объединены через REST API, что обеспечивает гибкость, надёжность и удобство масштабирования.

2.5 Интерфейс пользователя

Проектирование пользовательского интерфейса (UI) представляет собой важный этап в создании информационной системы, поскольку напрямую влияет на удобство взаимодействия пользователя с системой и общее восприятие программного продукта [13]. Интерфейс должен быть интуитивно понятным, функционально обоснованным и соответствующим архитектуре приложения.

На этапе проектирования интерфейса был сформирован базовый каркас навигации между основными разделами системы, что позволило структурировать пользовательские сценарии и задать логику переходов. На рисунке 14 представлена визуализация структуры интерфейса, отражающая иерархию экранов и их взаимосвязи.

Рисунок 14 - Диаграмма структуры интерфейса

Диаграмма отражает основные элементы интерфейса: главная страница выступает точкой входа, раздел «Доски» открывает доступ к проектам, в «Аккаунте» пользователь управляет личными данными, а вкладка «Сотрудники» отображает список участников. Функция создания проекта доступна напрямую из интерфейса. Внутри каждой доски реализованы задачи, список участников и настройки проекта - всё необходимое для управления рабочим процессом.

Такой подход обеспечивает логичную и последовательную навигацию по системе, что упрощает взаимодействие с интерфейсом, снижает количество действий, необходимых для выполнения задач, и способствует повышению производительности пользователей. Структурированность и визуальная ясность интерфейса позволяют минимизировать время обучения и снизить количество ошибок, связанных с пользовательским взаимодействием.

Для оценки визуальной реализации была также разработана демонстрационная версия главной страницы интерфейса. Макет, представленный на рисунке 15, отражает ключевые элементы взаимодействия и визуального восприятия.

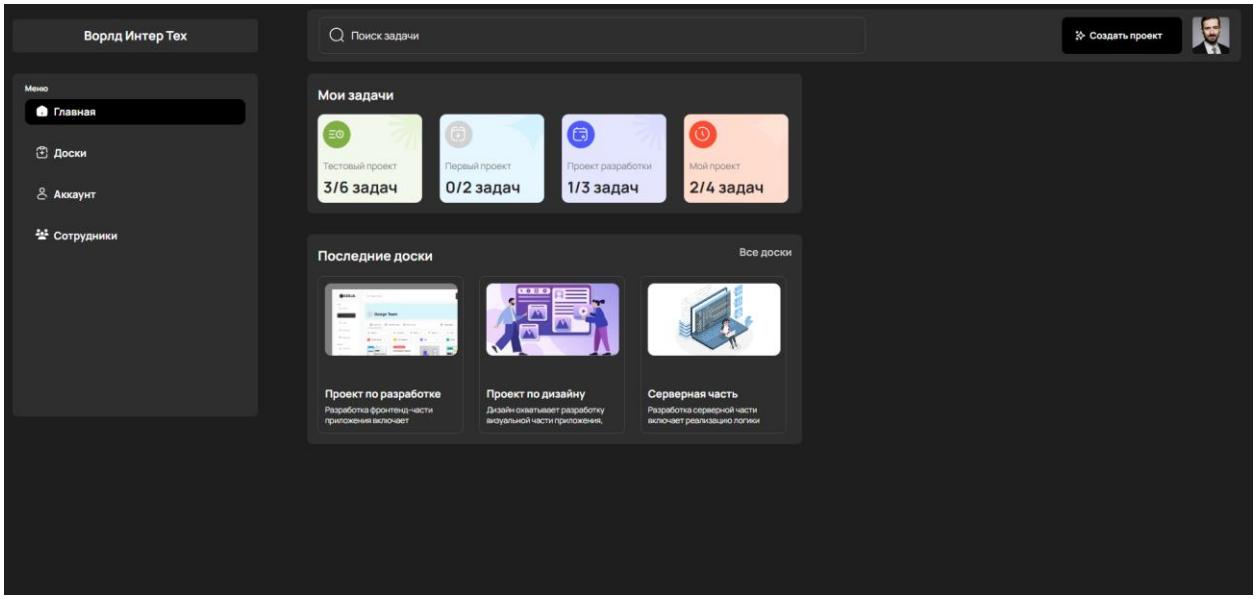

Рисунок 15 – Главная страница интерфейса

На изображении представлены основные функциональные блоки: меню навигации с доступом к основным разделам, панель поиска задач, а также блоки «Мои задачи» и «Последние доски». Последний компонент визуализирует активность пользователя в системе, обеспечивая быстрый доступ к недавно использовавшимся проектам.

Дизайн интерфейса ориентирован на современную компонентную структуру и реализован с учетом принципов удобства и минимализма. Использование темной цветовой схемы снижает нагрузку на зрение, а карточная организация информации упрощает восприятие контента.

Таким образом, интерфейс разработанной системы ориентирован на улучшение пользовательского опыта за счет четкой логики, визуальной структурности и высокого уровня доступности основных функций.

Вывод по главе 2

Во второй главе выполнено комплексное проектирование системы управления проектами, включающее архитектурные, логические и интерфейсные аспекты. Были разработаны UML-диаграммы, отражающие структуру, поведение и взаимодействие компонентов системы. Проектирована документно-ориентированная база данных, обеспечивающая масштабируемость и гибкость. Обоснован выбор технологического стека (React, Nest.js, Node.js), фреймворков и паттернов проектирования. Также реализована компонентная структура интерфейса, направленная на удобство пользователей.

Глава 3 Тестирование и разработка программного обеспечения

3.1 Пример реализации системы управления проектами

Ключевым файлом клиентской части проекта является main.jsx, в который подключаются все компоненты системы. Навигация реализована с использованием библиотеки react-router-dom, обеспечивающей маршрутизацию между страницами веб-приложения.

Для каждого модуля создана отдельная директория, содержащая два файла: компонент в формате jsx и файл стилей в формате scss. Такая структура обеспечивает логическое разделение, упрощает поддержку кода, ускоряет навигацию по проекту и повышает его масштабируемость.

Первоначально были разработаны модули Header.jsx и Menu.jsx, так как они присутствуют на всех страницах интерфейса. Компонент Header принимает пропсы для открытия модального окна поиска задач и для создания нового проекта. Также он отображает фотографию пользователя, полученную с сервера. Компонент Menu реализует навигационную панель. Работа модулей представлена на рисунке 16.

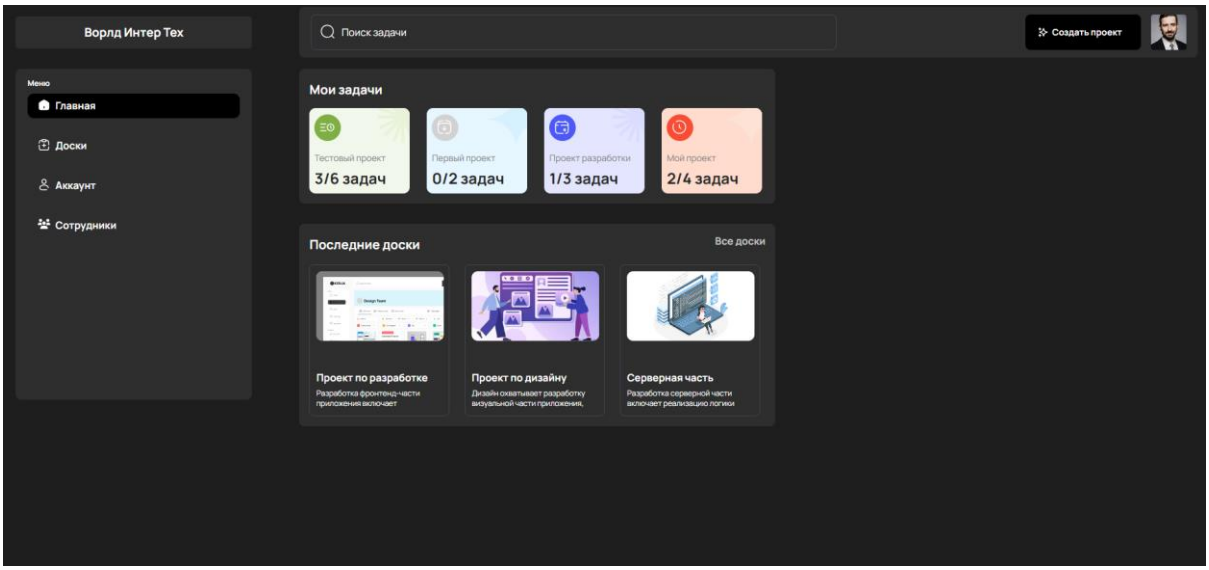

Рисунок 16 – Страница «Главная»

После авторизации пользователь попадает на главную страницу, реализованную компонентом Home, который получает с сервера данные о задачах и последних проектах, а затем отображает их в интерфейсе.

Компонент PopupCreateProject представляет модальное окно создания проектов. В нем пользователь вводит данные, выбирает фотографию для проекта, а также имеет возможность добавить участников в проект и указать его даты. На рисунке 17 представлена работу модуля

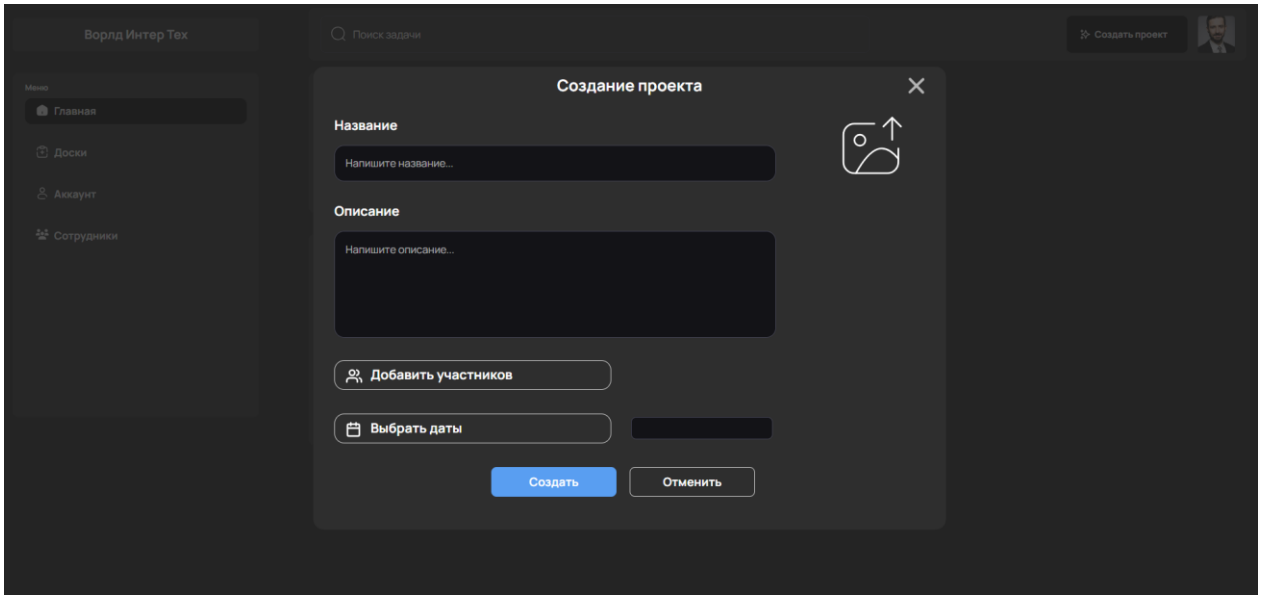

Рисунок 17 – Модальное окно «Создание проекта»

Функционал добавления участников в проект был реализован в виде отдельного компонента popupAddMemberProject. Указание даты разработки проекта осуществлено с использованием библиотеки react-datepicker, что позволило упростить выбор и валидацию дат на клиентской стороне.

Следующим этапом разработки стал модуль «Аккаунт», предназначенный для взаимодействия пользователя с личными данными. В модуле реализована функция изменения фотографии профиля, также реализован выхода из системы, обеспечивающая корректное завершение пользовательской сессии и защиту данных.

Модуль аккаунта представлен на рисунке 18.

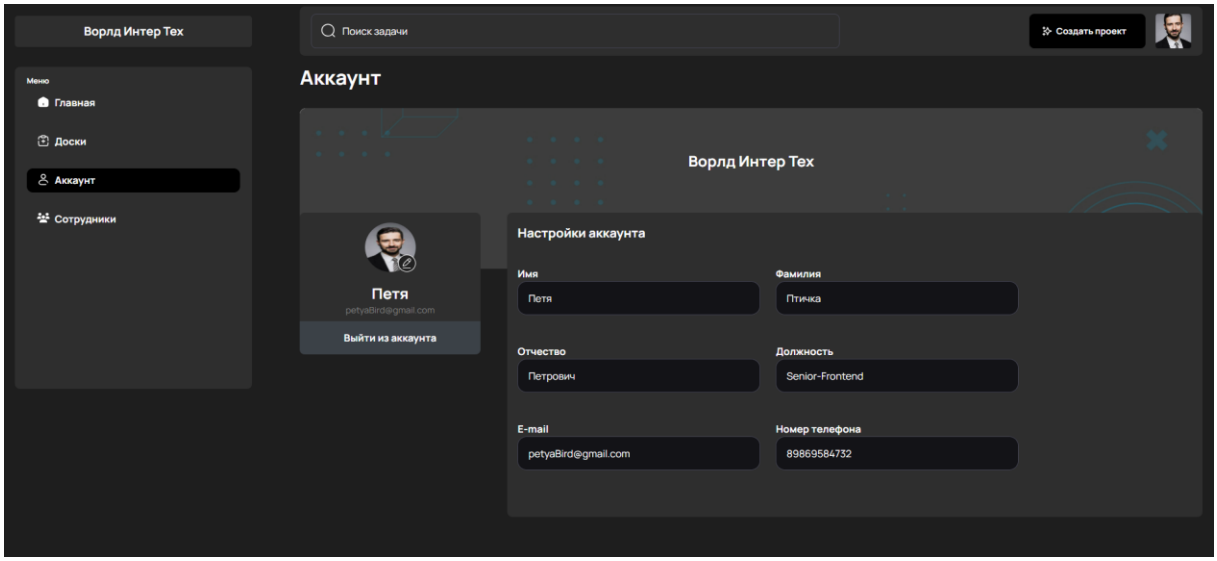

Рисунок 18 – Модуль аккаунта пользователя

После был реализован модуль «Доски». Данный модуль имеет в себе еще один компонент, который отвечает отдельно за каждый подгруженный проект. На рисунке 19 представлена работа модуля «Доски».

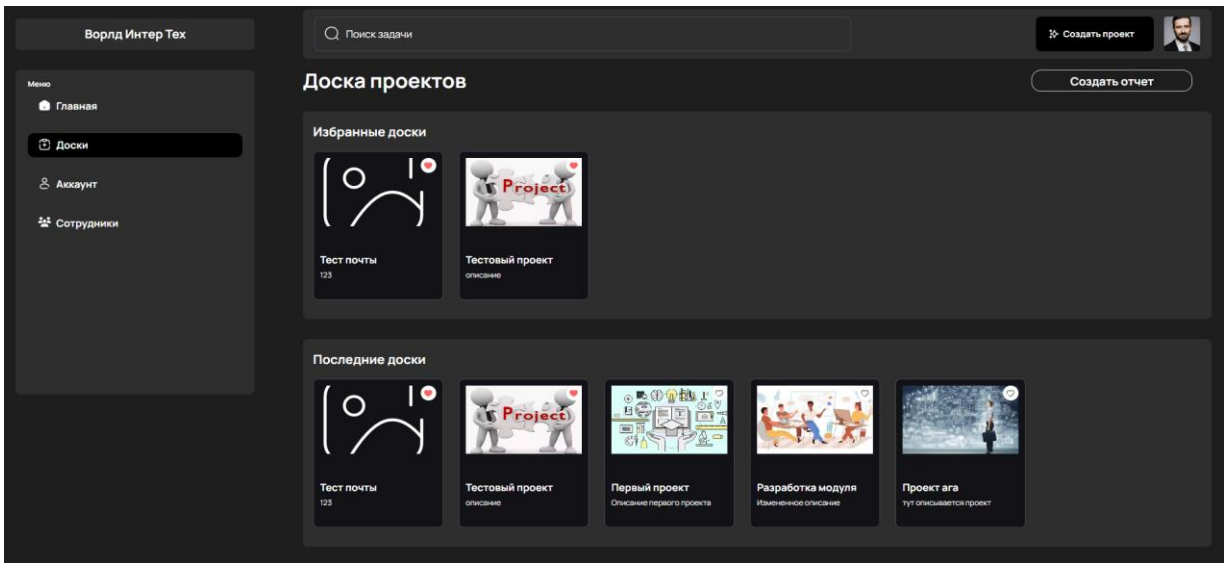

Рисунок 19 – Модуль «Доски»

Компонент Project включал в себя еще два дополнительных компонента:

- Stage - для корректного отображения этапов проекта;
- Task - для корректного отображения задач

На рисунке 20 представлен реализованный модуль страницы проекта.

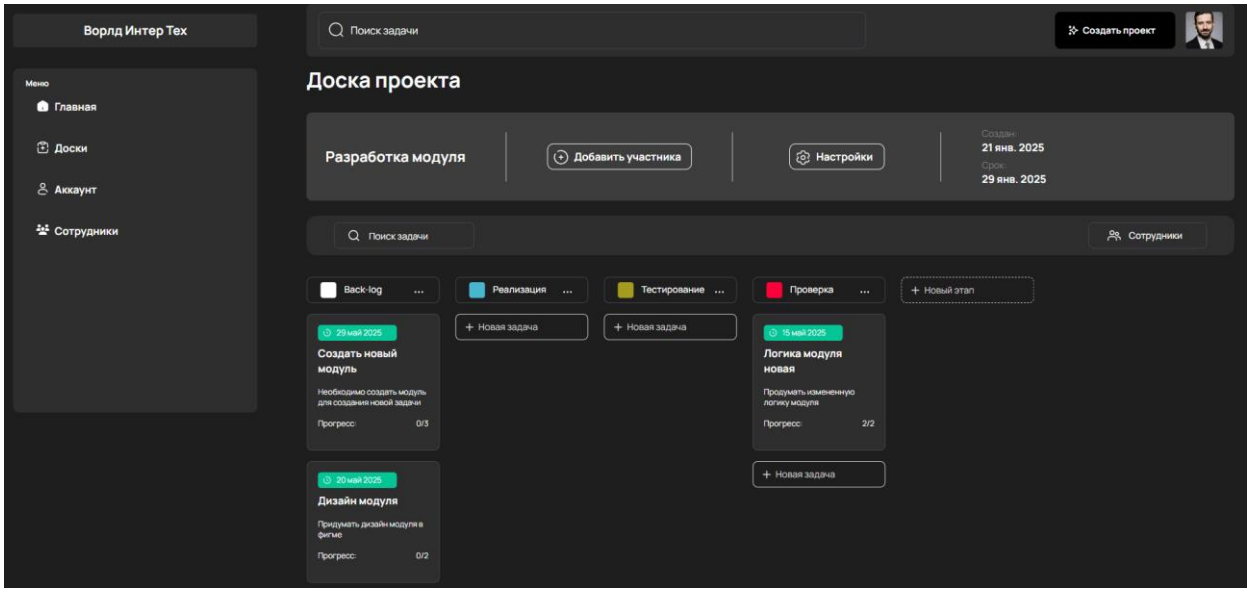

Рисунок 20 – Модуль «Доска проекта»

На данном рисунке представлена схема, демонстрирующая, как проект структурируется на более мелкие элементы - этапы и задачи. Такая иерархическая организация обеспечивает чёткое разделение процессов, что, в свою очередь, способствует лучшему планированию, распределению ответственности и контролю выполнения работ. Разбиение проекта на логические этапы позволяет отслеживать прогресс выполнения на каждом уровне, а наличие задач в рамках каждого этапа делает процесс управления более прозрачным и управляемым.

Следующим этапом разработки стал модуль «Сотрудники», предназначенный для отображения информации о кадровом составе компании. Сотрудники представлены в отсортированном виде по должностям, что упрощает восприятие структуры организации. Модуль отражает участие

каждого сотрудника в проектах и задачах, демонстрируя количество задействованных проектов и назначенных задач. Реализация модуля представлена на рисунке 21.

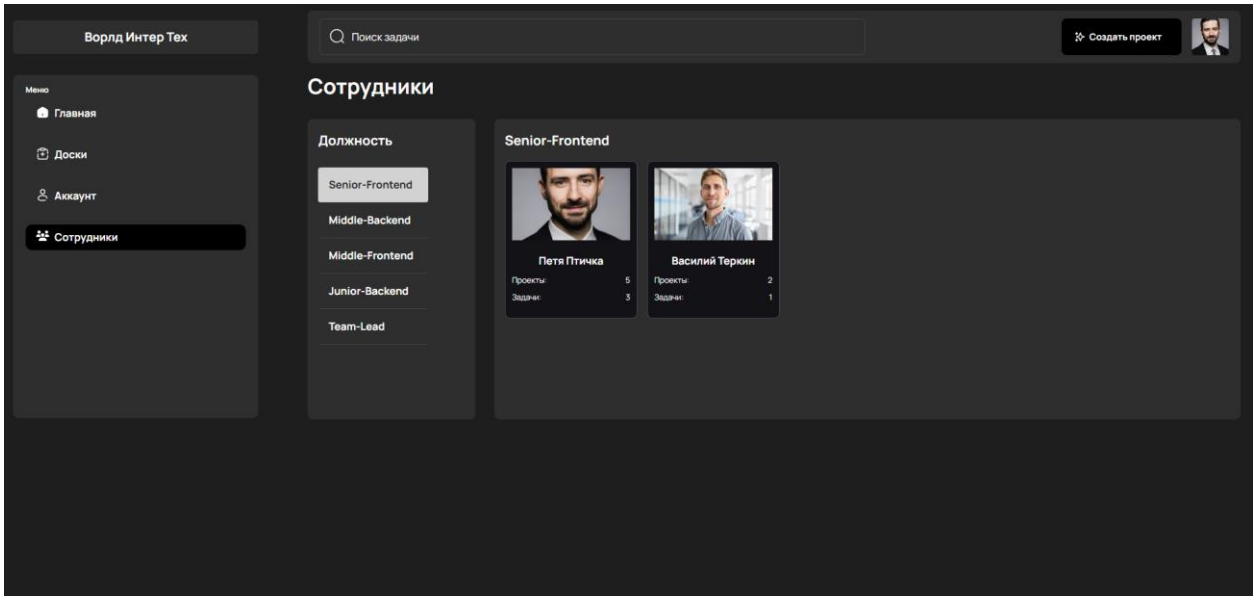

Рисунок 21 – Модуль «Сотрудники»

Также был реализован модуль «Отчет», представленный на рисунке 22.

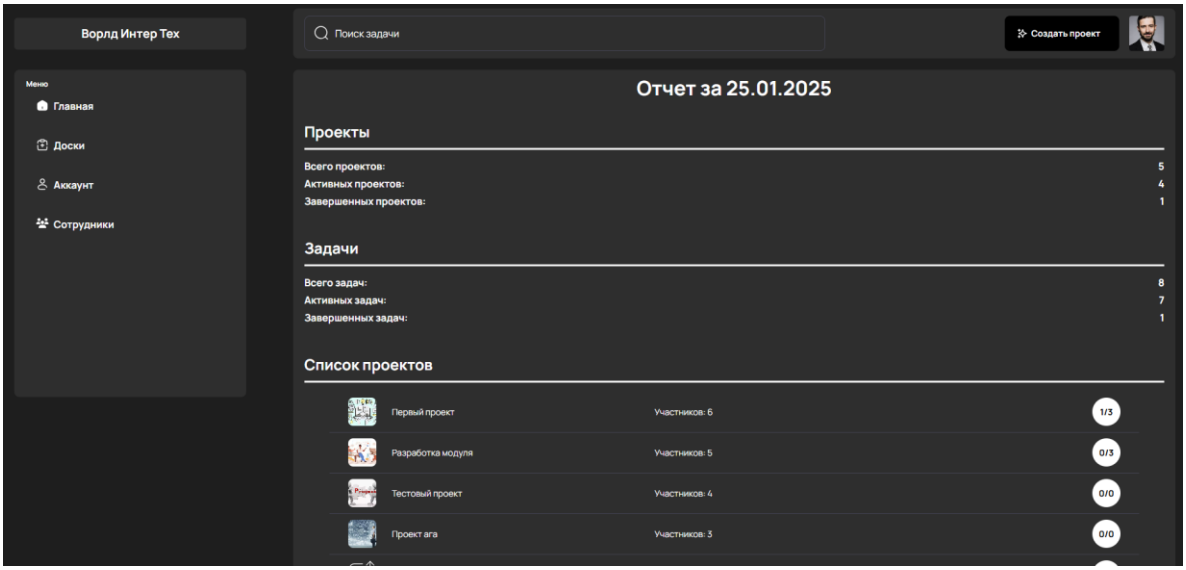

Рисунок 22 – Модуль «Отчет»

По нажатию пользователем на кнопку «Создать отчет», во время загрузки страницы с отчетом, идет запрос на сервер для получения необходимых данных, которые далее используются для отображения отчета. Данные позволяют получить информацию о проектах, задачах, а также список всех проектов с его количеством участником и задач.

3.2 Организация процесса тестирования

Для тестирования системы управления проектами были определены цели, а также разработана необходимая документацию, которая обеспечила структурированный и воспроизводимый подход к проверке функциональности системы.

Цели тестирования включают в себя:

- проверку соответствия реализованного функционала заявленным требованиям;
- обеспечение стабильной и корректной работы всех модулей приложения при различных сценариях использования и взаимодействия;
- выявление и устранение возможных ошибок и дефектов в пользовательском интерфейсе, бизнес-логике и взаимодействии компонентов;
- оценку производительности и стабильности системы при повышенной нагрузке;
- проверку корректности интеграции с внешними сервисами, включая систему уведомлений по электронной почте.

Для обеспечения качества, стабильности и функциональной целостности разрабатываемой системы управления проектами, на этапе реализации была проведена всесторонняя проверка её ключевых компонентов. Процесс тестирования строился на основе основных типов программных

тестов: функционального, модульного, интеграционного и нагрузочного. Такой подход позволил не только проверить отдельные модули, но и убедиться в корректности их взаимодействия между собой и устойчивости системы под высокой нагрузкой.

Сформированный тест-план охватывает весь разработанный функционал: от пользовательской аутентификации до генерации отчётности и интеграции с внешними сервисами. Для каждой функции были определены цель тестирования и применяемый метод, исходя из её роли в архитектуре приложения.

В таблице 10 представлены ключевые функции, подлежащие тестированию, с указанием соответствующей цели и выбранного метода проверки

Таблица 10 – Тест-план

Функциональность	Цель тестирования	Метод тестирования
Регистрация	Проверка возможности создания нового пользователя администратором	Функциональное тестирование
Аутентификация	Проверка входа в систему	Функциональное тестирование
Управление ролями	Проверка назначения ролей пользователей	Функциональное тестирование
Создание проекта	Проверка создания нового проекта и корректности его отображения	Модульное тестирование, функциональное тестирование
Редактирование проекта	Проверка изменения параметров проекта	Модульное тестирование, функциональное тестирование
Удаление проекта	Проверка удаления проекта и его недоступности	Модульное тестирование, функциональное тестирование
Назначение сроков проекта	Проверка установки и отображения сроков	Функциональное тестирование
Структуризация проекта	Проверка иерархии этапов и задач	Интеграционное тестирование

Продолжение таблицы 10

Функциональность	Цель тестирования	Метод тестирования
Добавление участников в проект	Проверка добавления новых участников в проект	Функциональное тестирование
Удаление участником из проекта	Проверка удаления участников из проекта	Функциональное тестирование
Список участников проекта	Проверка отображения участников в проекте	Интеграционное тестирование
Избранные проекты	Проверка функции избранных проектов и отображение их	Функциональное тестирование
Последние проекты	Проверка корректности отображения последних проектов	Функциональное тестирование
Создание задачи	Проверка создания новой задачи в проекте	Модульное тестирование, функциональное тестирование
Редактирование задачи	Проверка редактирования параметров задачи	Модульное тестирование, функциональное тестирование
Удаление задачи	Проверка корректного удаления задачи	Модульное тестирование, функциональное тестирование
Назначение сотрудников на задачу	Проверка назначения исполнителей на задачи	Функциональное тестирование
Назначение сроков задачи	Проверка установки сроков выполнения задачи	Функциональное тестирование
Назначение приоритетности задачи	Проверка возможности установки приоритета	Функциональное тестирование
Создание подзадачи к задаче	Проверка иерархической вложенности задач	Функциональное тестирование

Продолжение таблицы 10

Функциональность	Цель тестирования	Метод тестирования
Добавление комментария к задаче	Проверка добавления комментариев и их отображения	Модульное тестирование, функциональное тестирование
Просмотр личных данных пользователя	Проверка отображения информации аккаунта	Функциональное тестирование
Изменение аватарки	Проверка загрузки и отображения нового аватара	Функциональное тестирование
Выход из системы	Проверка выхода из системы	Функциональное тестирования
Просмотр списка сотрудников	Проверка отображения и сортировки списка сотрудников	Функциональное тестирование
Информация о сотруднике	Проверка отображения детальной информации о сотруднике	Функциональное тестирования
Генерация отчета по проектам	Проверка генерации и корректности отчета по проектам	Интеграционное тестирование
Интеграция с почтой	Проверка отправки уведомлений на почту	Интеграционное тестирование
Доступность системы	Проверка непрерывной работы сервиса	Нагрузочное тестирование
Автосохранение изменений в системе	Проверка автосохранения при отключении или перезагрузке	Функциональное тестирование
Шифрование данных	Проверка хранения данных в зашифрованном виде	Функциональное тестирование

Продолжение таблицы 10

Функциональность	Цель тестирования	Метод тестирования
Скорость обработки	Проверка времени отклика системы	Нагрузочное тестирование
Масштабируемость	Проверка устойчивости к росту количества пользователей	Нагрузочное тестирование

Разделение функций по типам тестирования позволяет систематизировать проверку и обеспечить наиболее полное покрытие сценариев использования. Модульное тестирование применялось для отладки внутренних компонентов: создания, редактирования и удаления сущностей. Интеграционные тесты подтверждали корректность связи между модулями - например, отображение участников, генерация отчётов и интеграция с почтовым сервисом. Функциональное тестирование обеспечивало соответствие ключевых пользовательских функций заявленным требованиям. Отдельное внимание было уделено нагрузочным испытаниям, целью которых стала проверка масштабируемости системы и её устойчивости при одновременной работе большого количества пользователей.

3.3 Модульное тестирование

«Модульное тестирование (unit testing). Проверяет отдельные модули или компонентное наполнение ПО на корректность их функционирования. Тесты проводятся над отдельными функциями, процедурами или классами» [10].

В рамках клиентской части разрабатываемой информационной системы тестирование проводилось с использованием инструментов Vitest и React

Testing Library, что обеспечило высокую гибкость при моделировании пользовательских действий и контроле за состоянием интерфейса.

В ходе модульного тестирования особое внимание уделялось тем компонентам, с которыми пользователь взаимодействует наиболее часто.

Создание проекта реализуется функцией handleForm, вызываемой при заполнении и отправке формы создания. В рамках теста проверялась корректность формирования POST-запроса, передача всех обязательных данных формы (название, описание, даты, участники, изображение), а также перенаправление пользователя на страницу нового проекта после успешного ответа от сервера. Отдельно проверялось, что запрос отправляется на правильный маршрут и содержит корректную структуру данных. Код теста представлен в приложении А.

Редактирование проекта осуществляется функцией handleProjectEdit. При выполнении теста проводилась проверка отправки POST-запроса на маршрут вида project/:id/edit с передачей актуальных данных формы. В случае успешного ответа проверялось выполнение действий по обновлению состояния проекта и возврату пользователя на страницу проекта. В тесте также контролировался вызов функции navigate, указывающей на корректную маршрутизацию после редактирования. Код теста представлен в приложении Б.

Удаление проекта реализовано через функцию handleProjectDelete, вызов которой сопровождается отправкой запроса методом DELETE и перенаправлением пользователя на список всех проектов. В ходе теста проверялась правильность маршрута запроса, наличие авторизационных параметров, а также факт вызова функции navigate("/boards") после подтверждения удаления. Тестирование подтвердило корректную обработку действий пользователя по завершению работы над проектом. Код теста представлен в приложении В.

Создание задачи покрывается тестом функции handleSubmit. Здесь проверялась отправка POST-запроса на маршрут task/create с полным набором

данных задачи, включая идентификатор проекта, название, описание, приоритет и диапазон дат. После успешного ответа проводилась проверка вызова функции invalidateQueries, необходимой для актуализации списка задач, а также вызов функции закрытия формы создания задачи. Код теста представлен в приложении Г.

Редактирование задачи осуществляется при помощи функции handleChange. Тестирование охватывало два случая: редактирование текстовых данных (например, названия) и изменение приоритета. В первом случае проверялась отправка запроса с новым значением поля на соответствующий маршрут task/:id/update/title. Во втором случае дополнительно проверялся вызов функции refetch, инициирующей обновление данных в пользовательском интерфейсе при изменении приоритета. Результаты тестов подтвердили корректную обработку каждого изменения со стороны интерфейса. Код теста представлен в приложении Д.

Удаление задачи выполняется функцией handleTaskDelete, и тест проводился аналогично сценарию удаления проекта. После вызова функции осуществлялась проверка отправки запроса методом DELETE на маршрут task/:id, а также инициировалась перезагрузка данных через taskQuery.refetch. Отслеживался сам факт удаления и соответствующее обновление состояния в интерфейсе. Код теста представлен в приложении Д.

Добавление комментария реализуется функцией handleCommentSubmit, отправляющей текст комментария и текущую дату на сервер. В ходе модульного теста проверялась передача правильных данных на маршрут task/:id/comment, очистка текстового поля после отправки и вызов функции commentsQuery.refetch. Проведённый тест подтвердил, что после отправки пользователь видит актуальный список комментариев, включающий только что добавленную запись. Код теста представлен в приложении Д.

В завершение процесса модульного тестирования был выполнен запуск всех подготовленных тестов с использованием команды npm run test, инициирующей работу тестового раннера Vitest.

На рисунке 23 представлены результаты выполнения модульных тестов на клиентской стороне проекта.

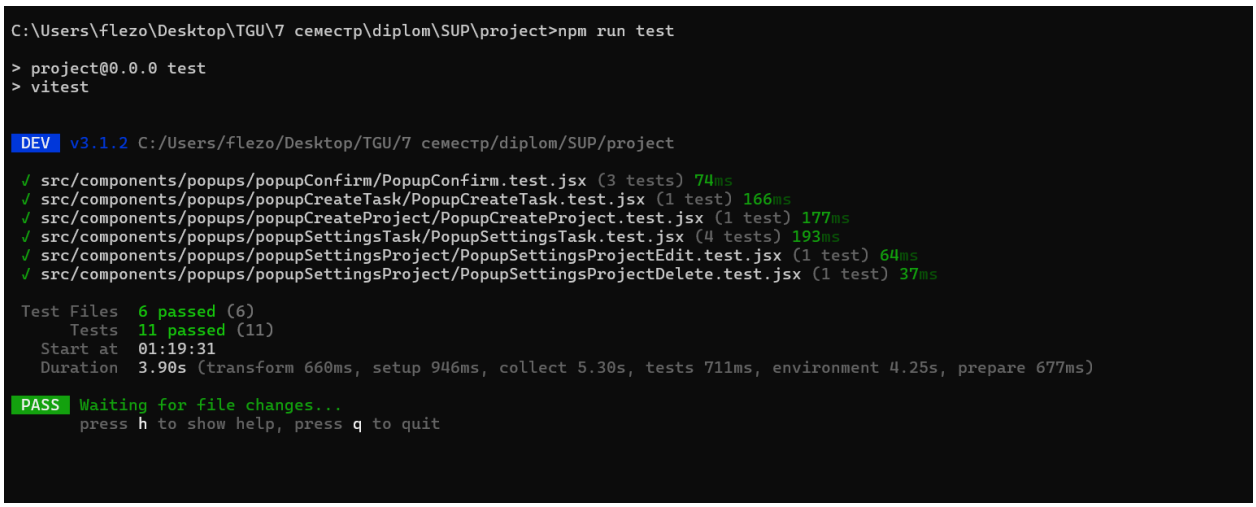

Рисунок 23 – Результаты модульного тестирования

Результаты тестирования показали следующее:

- все 6 тестовых файлов успешно загружены и выполнены без ошибок;
- общее количество протестированных функций составляет 11, что соответствует числу реализованных модульных сценариев, описанных ранее;
- каждый из тестов выполнился с минимальным временем отклика, не превышающим 200 мс, что подтверждает лёгкость тестируемых модулей и отсутствие избыточных зависимостей;
- строка PASS в нижней части терминала свидетельствует об успешном завершении всех проверок.

Таким образом, приведённые данные подтверждают стабильную работу модулей пользовательского интерфейса, надёжную интеграцию с серверной частью через HTTP-запросы, а также корректную реализацию ключевых пользовательских сценариев - от создания проекта до добавления комментариев к задачам.

3.4 Функциональное тестирование

«Функциональное тестирование (functional testing). Данный вид подразумевает проверку того, что ПО правильно решает пользовательские задачи, проверка функциональных задач. Функциональный вид тестирования сосредотачивается на функциях и возможностях приложения, а также его соответствии требованиям и ожиданиям пользователей. Основная цель функционального тестирования — убедиться, что все функции и операции приложения работают корректно и взаимодействуют друг с другом так, как ожидается» [10].

Данный вид тестирования был выбран как ключевой этап проверки системы, поскольку ориентирован на поведение приложения с позиции конечного пользователя, позволяет выявить отклонения от спецификации и проверяет взаимодействие компонентов через интерфейс и API. Такой подход обеспечивает уверенность в стабильности, предсказуемости и корректности работы реализованных функций.

Проверке подверглись все основные модули системы: регистрация и авторизация пользователей, управление проектами и задачами, ролевая модель доступа, отображение информации о сотрудниках, генерация отчётности, интеграция с электронной почтой и поиск по задачам.

Каждая роль была протестирована на предмет корректности интерфейса, доступа к функциям и обработки данных. Тесты выполнялись вручную по заранее определённым пользовательским сценариям, с фиксацией ожидаемых и фактических результатов.

Для системной оценки были разработаны тест-кейсы, охватывающие ключевые сценарии. В каждом тесте указывались цель, предусловия, шаги и ожидаемый результат и фактический результат. Структура тестов обеспечила охват всего функционала для всех категорий пользователей, подтвердив корректную реализацию ролей, устойчивость поведения интерфейса и соответствие бизнес-логике.

Результаты функционального тестирования представлены в таблице 11.

Таблица 11 – Тест-кейсы

Название	Предусловие	Шаги	Ожидаемый результат	Статус	Фактический результат
Регистрация нового пользователя администратором	Пользователь авторизирован с правами администратора	1. Открыть страницу регистрации 2. Заполнить все поля формы 3. Нажать кнопку «Зарегестрировать»	Успешная регистрация, появляется сообщение о успешной регистрации	Успех	Успешная регистрация, появляется сообщение о успешной регистрации (рисунок Е1)
Аутентификация пользователя с корректными данными	Пользователь зарегистрирован в системе	1. Открыть страницу авторизации 2. Ввести email и пароль 3. Нажать кнопку «Войти»	Успешная авторизация, переход на главную страницу	Успех	Успешная авторизация, переход на главную страницу (рисунок Е2)
Аутентификация пользователя с некорректными данными	-	1. Открыть страницу авторизации 2. Ввести некорректные email и пароль 3. Нажать кнопку «Войти»	Ошибка авторизации, появляется сообщение об ошибке	Успех	Ошибка авторизации, появляется сообщение об ошибке «Неверная почта или пароль» (рисунок Е3)
Аутентификация пользователя с пустыми полями	-	1. Открыть страницу авторизации 2. Нажать кнопку «Войти»	Ошибка авторизации, появляется сообщение об ошибке	Успех	Ошибка авторизации, появляется сообщение об ошибке «Заполните все обязательные поля» (рисунок Е4)

Продолжение таблицы 11

Название	Предусловие	Шаги	Ожидаемый результат	Статус	Фактический результат
Создание проекта с корректными данным	Пользователь авторизирован с правами администратора или руководителя	1. Открыть модальное окно создания проекта 2. Заполнить все поля формы 3. Выбрать фото для проекта 4. Добавить участников в проект 5. Выбрать даты сроков проекта 6. Нажать кнопку «создать»	Проект успешно создается, пользователя перенаправляет на страницу созданного проекта	Успех	Проект успешно создается, пользователя перенаправляет на страницу созданного проекта (рисунок Е.5).
Создание проекта с пустыми полями формы	Пользователь авторизирован с правами администратора или руководителя	1. Открыть модальное окно создания проекта 2. Заполнить не все поля формы 3. Нажать кнопку «создать»	Проект не создается. Появляется сообщение об ошибке	Успех	Проект не создается. Появляется сообщение об ошибке: "Заполните все обязательные поля" (рисунок Е.6).

Продолжение таблицы 11

Название	Предусловие	Шаги	Ожидаемый результат	Статус	Фактический результат
Отображение проектов во вкладке «Доски»	Пользователь авторизирован в системе	1. Открыть страницу «Доски»	Пользователь видит список всех проектов, в которых принимает участие	Успех	Пользователь видит список всех проектов, в которых принимает участие (рисунок Е7)
Отображение последних проектов во вкладке «Доски»	Пользователь авторизирован в системе, открывал один или более проект	1. Открыть страницу «Доски»	Пользователь видит список последних проектов в отдельной категории	Успех	Пользователь видит список последних проектов в отдельной категории (рисунок Е8)
Добавление проекта в избранное	Пользователь авторизирован в системе и участвует в одном или более проектах	1. Открыть страницу «Доски» 2. Нажать на значок сердца у любого проекта	Проект отобразился в списке «Избранные доски», сердце стало красного цвета	Успех	Проект отобразился в списке «Избранные доски», сердце стало красного цвета (рисунок Е9)

Продолжение таблицы 11

Название	Предусловие	Шаги	Ожидаемый результат	Статус	Фактический результат
Редактирование проекта	Пользователь авторизирован в системе, имеет права администратора или руководителя, участвует в одном или более проектах, открыл страницу проекта	1. Нажать на копку «Настройки» 2. Изменить название, описание, дату сроков 3. Нажать «Сохранить»	Все изменения сохранились и отобразились на странице проекта	Успех	Все изменения сохранились и отобразились на странице проекта (рисунок Е10)
Удаление проекта	Пользователь авторизирован в системе, имеет права администратора или руководителя, участвует в одном или более проектах. Открыл страницу проекта	1. Нажать на кнопку «Настройки» 2. Нажать на кнопку «Удалить проект». 3. Нажать на кнопку «Подтвердить»	Проект удаляется из БД, пользователь перенаправляется на страницу «Доски».	Успех	Проект удаляется из БД, пользователь перенаправляется на страницу «Доски».

Продолжение таблицы 11

Название	Предусловие	Шаги	Ожидаемый результат	Статус	Фактический результат
Права доступа по ролям	Пользователь авторизирован в системе, имеет права сотрудника4	1. Перейти в доски 2. Открыть любой проект 3. Нажать на «Добавить участника» 4. Нажать на «Настройки» 5. Нажать на «Участники проекта»	Появляется модальное окно с ошибкой доступа. Также у сотрудника отсутствуют вкладка «Сотрудники», кнопка создания проекта и кнопка удаления участника из проекта	Успех	Появляется модальное окно с ошибкой доступа. Также у сотрудника отсутствуют вкладка «Сотрудники», кнопка создания проекта и кнопка удаления участника из проекта (рисунки Е11-Е13)
Создание задачи	Пользователь авторизирован в системе, находится на странице проекта	1. Нажать кнопку «Новая задача» в созданном этапе 2. Заполнить поля названия и описания, выбрать дату, приоритет 3. Нажать кнопку «Создать»	В этапе отображается созданная задача	Успех	В этапе отображается созданная задача (рисунок Е14)

Продолжение таблицы 11

Название	Предусловие	Шаги	Ожидаемый результат	Статус	Фактический результат
Редактирование задачи, автоматическое сохранение в системе	Пользователь авторизирован в системе, участвует в одном или более проектах, находится на странице проекта, в проекте созданы задачи	1. Нажать на любую задачу в проекта 2. Изменить название, описание и сроки задачи 3. Закрыть модульное окно задачи	Измененная задача отображается на странице корректно проекта. Сохранение происходит автоматически, без нажатия на «Сохранить»	Успех	Измененная задача отображается на странице корректно проекта. Сохранение происходит автоматически, без нажатия на «Сохранить» (рисунки Е15 – Е16)
Назначение сотрудников на задачу	Пользователь авторизирован в системе, участвует в одном или более проектах, находится на странице проекта, в проекте созданы задачи	1. Нажать на любую задачу 2. Нажать «Добавить участников» 3. Нажать на одного или более участников	Участники добавлены в задачу	Успех	Участники добавлены в задачу (рисунок Е17)

Продолжение таблицы 11

Название	Предусловие	Шаги	Ожидаемый результат	Статус	Фактический результат
Назначение приоритета задачи	Пользователь авторизирован в системе, участвует в одном или более проектах, находится на странице проекта, в проекте созданы задачи	1. Нажать на любую задачу 2. Выбрать приоритет задачи от 1 до 5	Указан приоритет задачи, выбранный уровень приоритета подсвечивается	Успех	Указан приоритет задачи, выбранный уровень приоритета подсвечивается (рисунок Е18)
Создание подзадач к задаче и корректность работы подзадач	Пользователь авторизирован в системе, участвует в одном или более проектах, находится на странице проекта, в проекте созданы задачи, открыта модальное окно любой задачи	1. Нажать кнопку «Создать чек-лист» 2. Нажать кнопку «Добавить элемент» 3. Написать название 4. Нажать кнопку «Подтвердить» 5. Создать еще две подзадачу 6. Нажать на чек-бокс одной задачи, чтобы отметить её выполненной 7. Навестить на вторую подзадачу и нажать на крестик для удаления	Подзадачи работают корректны, их можно создать, отметить выполнение и удалить	Успех	Подзадачи работают корректны, их можно создать, отметить выполнение и удалить (рисунок Е19)

Продолжение таблицы 11

Название	Предусловие	Шаги	Ожидаемый результат	Статус	Фактический результат
Создание подзадач к задаче и корректность работы подзадач	Пользователь авторизирован в системе, участвует в одном или более проектах, находится на странице проекта, в проекте созданы задачи, открыта модальное окно любой задачи	1. Нажать кнопку «Создать чек-лист» 2. Нажать кнопку «Добавить элемент» 3. Написать название 4. Нажать кнопку «Подтвердить» 5. Создать еще две подзадачу 6. Нажать на чек-бокс одной задачи, чтобы отметить её выполненной 7. Навестить на вторую подзадачу и нажать на крестик для удаления	Подзадачи работают корректны, их можно создать, отметить выполнение и удалить	Успех	Подзадачи работают корректны, их можно создать, отметить выполнение и удалить (рисунок Е19)
Создание комментария к задаче	Пользователь авторизирован в системе, участвует в одном или более проектах, находится на странице проекта, в проекте созданы задачи, открыта модальное окно любой задачи	1. Нажать на текстовое окно комментария для его активации 2. Написать текст комментария 3. Нажать «Отправить»	Комментарий корректно отображается в задаче	Успех	Комментарий корректно отображается в задаче (рисунок Е20)

Продолжение таблицы 11

Название	Предусловие	Шаги	Ожидаемый результат	Статус	Фактический результат
Удаление задачи	Пользователь авторизирован в системе, находится на странице проекта, в котором созданы задачи, открыто окно задачи с комментарием	1. 1. Навестись курсором на свой комментарий 2. Нажать на красный значок корзины для удаления	Комментарий успешно удален	Успех	Комментарий успешно удален (рисунок Е21)
Добавление участников в проект и просмотр списка проекта	Пользователь авторизирован в системе, имеет права руководителя или администратора, участвует в проектах, находится на странице проекта	1. Нажать на кнопку «Добавить участника» 2. В окне выбора участников выбрать одного или более сотрудника 3. Нажать кнопку добавить	Сотрудники добавлены на проект и отображаются в модальном окне проекта «Сотрудники»	Успех	Сотрудники добавлены на проект и отображаются в модальном окне проекта «Сотрудники» (рисунки Е22 – Е23)

Продолжение таблицы 11

Название	Предусловие	Шаги	Ожидаемый результат	Статус	Фактический результат
Просмотр личных данных пользователя	Пользователь авторизирован в системе	1. Открыть вкладку «Аккаунт»	Во вкладке «аккаунт» корректно отображаются все данные пользователя	Успех	Во вкладке «аккаунт» корректно отображаются все данные пользователя (рисунок 24)
Изменение аватарки пользователя	Пользователь авторизирован в системе, находится на вкладке «Аккаунт»	1. Нажать на значок карандаша рядом с аватаркой 2. Выбрать файл для загрузки 3. Нажать «Открыть»	Фотография успешно загрузилась, корректно отображается во всех вкладках	Успех	Фотография успешно загрузилась, корректно отображается во всех вкладках (рисунок 25)
Выход из системы	Пользователь авторизирован в системе, находится на вкладке «Аккаунт»	1. Нажать на кнопку «Выйти из аккаунта»	Система выходит из аккаунта пользователя и перенаправляет на страницу авторизации	Успех	Система выходит из аккаунта пользователя и перенаправляет на страницу авторизации

Продолжение таблицы 11

Название	Предусловие	Шаги	Ожидаемый результат	Статус	Фактический результат
Просмотр списка сотрудников	Пользователь авторизирован в система, имеет права доступа администратора или руководителя	1. Перейти во вкладку «Сотрудники»	Во вкладке «сотрудники» корректно отображаются все сотрудники, отсортированные по своей должности, так же указаны во сколько проектах и задачах задействованы сотрудники	Успех	Во вкладке «сотрудники» корректно отображаются все сотрудники, отсортированные по своей должности, так же указаны во сколько проектах и задачах задействованы сотрудники (рисунок Е26)
Просмотр подробной информации о сотруднике	Пользователь авторизирован в система, имеет права доступа администратора или руководителя, находится на вкладке «Сотрудники»	1. Нажать на любого сотрудника	Открывается модальное окно с подробной информацией о сотруднике, показано его фото, ФИО, контактные данные, должность, а так же в каких проектах задействован и сколько задач имеет	Успех	Открывается модальное окно с подробной информацией о сотруднике, показано его фото, ФИО, контактные данные, должность, а так же в каких проектах задействован и сколько задач имеет (рисунок Е27)

Продолжение таблицы 11

Название	Предусловие	Шаги	Ожидаемый результат	Статус	Фактический результат
Шифрование данных	В системе есть зарегистрированные пользователи	1. Зарегистрировать нового пользователя с известным паролем 2. Перейти в базу данных 3. Найти запись о пользователе 4. Проверить поле password	Пароль хранится в виде хэш-строки. В базе не видно настоящего пароля	Успех	Пароль хранится в виде хэш-строки. В базе не видно настоящего пароля (рисунок Е28)

В рамках функционального тестирования была проверена корректность работы всех ключевых модулей и пользовательских сценариев системы управления проектами. Тестирование проводилось вручную с учётом трёх ролей: администратора, руководителя и сотрудника, что позволило убедиться в надёжной реализации механизма разграничения прав доступа и логике интерфейса.

Всего было выполнено более 20 тест-кейсов, охватывающих критически важные функции: авторизацию, управление проектами и задачами, назначение исполнителей, работу с подзадачами и комментариями, автоматическое сохранение данных, генерацию отчётов, редактирование профиля и управление доступом. Дополнительно протестированы элементы интерфейса и взаимодействие между участниками проекта.

Все тесты были успешно пройдены, что подтверждено итоговой таблицей с указанием фактических результатов и визуальными примерами. Зафиксированы корректная обработка действий пользователя, устойчивое отображение данных, отсутствие критических ошибок, стабильная работа с граничными значениями и надёжность механизмов фильтрации и навигации.

Результаты функционального тестирования подтвердили соответствие системы заявленным требованиям, её стабильность и готовность к эксплуатации. Разработанное решение может быть рекомендовано к внедрению и масштабированию в рамках корпоративной ИТ-инфраструктуры.

3.5 Интеграционное тестирование

«Интеграционное тестирование. Проверяется взаимодействие и взаимосвязь между различными модулями или компонентами программного обеспечения. Цель — убедиться, что они работают вместе и взаимодействуют без ошибок» [10].

Особое внимание уделено сценариям, связанным с иерархией проекта, отображением участников, формированием отчётности и отправкой уведомлений.

Результаты тестирования подтвердили корректность связей между компонентами, целостность данных и устойчивость взаимодействия между уровнями архитектуры. Ключевые сценарии и их итоги приведены в таблице 12.

Таблица 12 – Интеграционные тест-кейсы

Название	Предусловие	Шаги	Ожидаемый результат	Статус	Фактический результат
Структуризация проекта на этапы и задачи	Пользователь авторизован, имеет роль «руководитель» или «администратор»	1. Создать проект 2. Добавить несколько этапов 3. Создать задачи в этапах	Этапы отображаются в правильной иерархии; задачи корректно связаны с этапами и отображаются внутри них	Успех	Этапы отображаются в правильной иерархии; задачи корректно связаны с этапами и отображаются внутри них (рисунок Е14)
Отображение списка участников проекта	Пользователь авторизирован в системе, участвует в одном или более проектах, находится на странице проекта	1. Нажать на «Сотрудники»	Открылось модальное окно с участниками проекта, все участники отображаются корректно, с указанной должностью	Успех	Открылось модальное окно с участниками проекта, все участники отображаются корректно, с указанной должностью (рисунок Е13)

Продолжение таблицы 12

Название	Предусловие	Шаги	Ожидаемый результат	Статус	Фактический результат
Генерация отчёта по проекту	Пользователь авторизирован и имеет права доступа администратора или руководителя	1. Перейти на вкладку «Доски» 2. Нажать «Создать отчёт»	Отчёт содержит актуальные данные по всем проектам и задачам, а также показал список всех проектов	Успех	Отчёт содержит актуальные данные по всем проектам и задачам, а также показал список всех проектов (рисунок Е29)
Интеграция с почтой	Пользователь авторизирован и имеет права доступа администратора или руководителя, участвует в одном или более проектах, находится на странице проекта	1. Добавить сотрудника в проект 2. Добавить этого же сотрудника на задачу 3. Проверить почту добавленного сотрудника	Пользователь получает email-уведомление о добавление его в новый проект и новую задачу	Успех	Пользователь получает email-уведомление о добавление его в новый проект и новую задачу (рисунок Е30)

Особое внимание при тестировании было уделено ключевым узлам системы: структурированию проектов, отображению участников, формированию отчётности и отправке уведомлений. Результаты показали, что соответствующие компоненты корректно обмениваются данными, отображают актуальную информацию и работают в соответствии с назначенными пользовательскими ролями.

Успешное выполнение всех тестов подтвердило стабильную интеграцию клиентской части, серверной логики и базы данных, а также надёжное взаимодействие с внешними сервисами. Это свидетельствует о готовности системы к эксплуатации в реальных условиях и её способности обеспечивать целостность и устойчивость бизнес-процессов.

3.6 Нагрузочное тестирование

Нагрузочное тестирование представляет собой вид нефункционального тестирования, направленный на оценку производительности системы при интенсивной нагрузке. Оно позволяет определить, насколько стабильно и эффективно функционирует приложение при одновременном обращении большого количества пользователей, а также оценить устойчивость к пиковым нагрузкам и скорость обработки запросов.

Цель данного этапа - проверить производительность и отказоустойчивость системы управления проектами в условиях, приближенных к реальной эксплуатации. Основное внимание уделяется времени отклика, стабильности работы сервера, наличию сбоев и способности системы одновременно обслуживать множество пользователей.

На начальном этапе была проведена проверка доступности системы с использованием инструмента мониторинга. В течение суток каждые 60 секунд фиксировались параметры доступности веб-приложения. Результаты показали 100% аптайм как за последние 24 часа, так и за 30 дней, среднее время отклика составило 238 мс, текущее - 160 мс. Подключение осуществлялось по

протоколу HTTPS с действующим сертификатом, случаев недоступности не зафиксировано. Результаты представлены на рисунке 24.

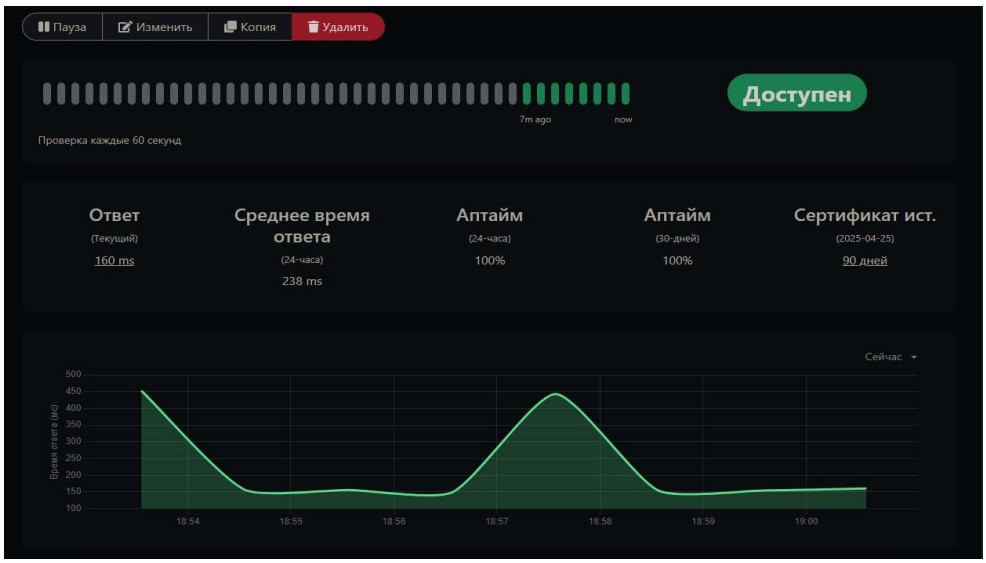

Рисунок 24 – Доступность системы

Во втором тесте система подвергалась искусственной нагрузке в виде 100 параллельных пользователей, выполняющих запросы к API на протяжении 60 секунд. Использовался инструмент командной строки wrk, нацеленный на маршрут получения проектов. Результаты представлены на рисунке 25.

```
Max time (s):        60
Concurrent clients:  100
Running on cores:    4
Agent:               none

Completed requests:  8516
Total errors:        0
Total time:          60.024 s
Mean latency:        698.8 ms
Effective rps:       142

Percentage of requests served within a certain time
  50%      601 ms
  90%      769 ms
  95%      861 ms
  99%      911 ms
 100%      951 ms (longest request)
```

Рисунок 25 – Нагрузка системы 100 пользователями

Проведённое нагрузочное тестирование продемонстрировало, что система управления проектами обладает высокой устойчивостью, надёжностью и готова к эксплуатации в условиях одновременного обращения большого количества пользователей.

Результаты проверки доступности показали стабильную работу веб-приложения при круглосуточном мониторинге: аптайм составил 100% как за сутки, так и за 30-дневный период, без единого сбоя. Среднее время отклика сервера оставалось в пределах 238 мс, что соответствует требованиям к современным веб-системам и обеспечивает комфортную работу для конечного пользователя.

Нагрузочный тест с 100 параллельными клиентами подтвердил способность приложения обрабатывать более 140 запросов в секунду без потери производительности и с нулевым количеством ошибок. Более 99% всех запросов были выполнены менее чем за 1 секунду, а максимальная зафиксированная задержка составила 951 мс, что допустимо при пиковых нагрузках.

Таким образом, система демонстрирует отличные показатели по отказоустойчивости, скорости обработки данных и масштабируемости. Это позволяет утверждать, что приложение может быть надёжно использовано в корпоративной среде с потенциальным увеличением пользовательской нагрузки.

Вывод по главе 3

В третьей главе была проведена всесторонняя проверка системы управления проектами, включающая модульное, функциональное, интеграционное и нагрузочное тестирование. Проверены все ключевые пользовательские сценарии и взаимодействие компонентов системы. Все тест-кейсы выполнены успешно. Система показала высокую стабильность, надёжность и масштабируемость при одновременной нагрузке до 100 пользователей. Результаты тестирования подтверждают, что приложение готово к эксплуатации и соответствует заявленным требованиям.

Заключение

В рамках данной выпускной квалификационной работы была выполнена разработка информационной системы управления проектами для компании «ВорлдИнтерТех Рус», адаптированной под специфические бизнес-процессы и требования организации. Проведённый анализ существующих решений показал их недостаточную гибкость, ограниченные возможности масштабирования и интеграции, что обосновало необходимость создания собственного программного продукта.

На этапе проектирования была разработана архитектура программного обеспечения, включающая модульную клиент-серверную структуру, нереляционную базу данных и REST API для интеграции с внешними сервисами. Использование современного технологического стека беспечило масштабируемость, отказоустойчивость и удобство поддержки системы. Визуальные модели в формате UML позволили формализовать логику и структуру будущего приложения.

Практическая часть работы охватывала реализацию пользовательского интерфейса, серверной логики и алгоритмов обработки данных, а также проведение комплексного тестирования. Система прошла модульные, функциональные, интеграционные и нагрузочные испытания, по итогам которых была подтверждена её стабильность, безопасность и производительность при одновременной работе большого числа пользователей.

Разработанное программное обеспечение успешно решает задачу автоматизации управления проектами, обеспечивает эффективное распределение задач, контроль сроков, взаимодействие сотрудников и генерацию отчётности. Система готова к внедрению и дальнейшему развитию в рамках корпоративной ИТ-инфраструктуры, способствуя повышению эффективности процессов, снижению операционных рисков и увеличению прозрачности управления в компании.

Список используемой литературы и используемых источников

1. А. Г. Белик, В. Н. Цыганенко. Проектирование и архитектура программный систем [Электронный ресурс]: учеб. пособие / А. Г. Белик, В. Н. Цыганенко ; Минобрнауки России, ОмГТУ. – Омск : Изд-во ОмГТУ, 2016. – 96 с. : ил. ISBN 978-5-8149-2258-8

2. Войтюк Т.Е., Осетрова И.С., Основы проектирования реляционных баз данных средствами инструментальной среды– СПб: Университет ИТМО, 2020. – 70 с.

3. Диаграмма вариантов использования [Электронный ресурс]: https://clck.ru/3MGZ2j (дата обращения: 08.01.2025).

4. Диаграмма Исикавы [Электронный ресурс]: https://clck.ru/3MGZ3H (дата обращения: 04.01.2025).

5. Диаграмма развёртывания [Электронный ресурс]: https://clck.ru/3MPWjg (дата обращения: 23.01.2025).

6. Кох Ю.А. Физическое проектирование базы данных [Электронный ресурс]: https://scienceforum.ru/2017/article/2017030333 (дата обращения: 14.02.2025)

7. Маклаков С. В. Моделирование бизнес-процессов с AllFusion Process Modeler (BPwin 4.1) / С. В. Маклаков. — М. : Диалог-МИФИ, 2003. — 238 с.

8. Рыбалка С.А. Диаграмма компонентов (Component Diagram) [Электронный ресурс]: https://clck.ru/3MMoPq (дата обращения: 20.02.2025).

9. Сергеева Т.И. Базы данных: модели данных, проектирование, язык SQL: учеб. пособие / Т.И. Сергеева, М.Ю. Сергеев. Воронеж: ФГБОУ ВПО «Воронежский государственный технический университет», 2012. 233 с.

10. Тестирование и 7 основных этапов его проведения [Электронный ресурс]: https://neiros.ru/blog/code/testirovanie-i-7-osnovnykh-etapov-ego-provedeniya/ (дата обращения: 25.02.2025).

11. Шестеров Г. UML: State Machine Diagram [Электронный ресурс]: https://shesterov.by/tpost/n26iurdac1-uml-state-machine-diagram (дата обращения: 16.02.2025)

12. Asana [Электронный ресурс] — URL: https://asana.com/ru (дата обращения: 26.12.2024).

13. Ater T. Building Progressive Web Apps: Bringing the Power of Native to the Browser. O'Reilly Media 2017. 288 p

14. Banker, Kyle MongoDB in Action M. Pearson Education, 2016 – 375c.

15. Jira [Электронный ресурс] — URL: https://www.atlassian.com/software/jira/ (дата обращения: 26.12.2024).

16. Node.js [Электронный ресурс] — URL: https://nodejs.org/docs/latest/api/ (дата обращения: 10.01.2025)

17. React [Электронный ресурс] — URL: https://legacy.reactjs.org/ (дата обращения: 08.01.2025)

18. Software Requirements [Электронный ресурс]. URL: http://beepvolume.com/oop/2020/software-requirements/ (дата обращения: 29.12.2024).

19. Trello [Электронный ресурс] — URL: https://trello.com/ (дата обращения: 26.12.2024).

20. What is REST? [Электронный ресурс]. URL: https://www.codecademy.com/article/what-is-rest (дата обращения: 06.01.2025)

Приложение А

Листинг файла PopupCreateProject.test.jsx

```
import React from 'react';
import { render, fireEvent, waitFor } from '@testing-library/react';
import { describe, it, expect, vi, beforeEach } from 'vitest';
import { QueryClient, QueryClientProvider } from '@tanstack/react-query';
import PopupCreateProject from './PopupCreateProject';
import axios from 'axios';

vi.mock('axios');

describe('PopupCreateProject', () => {
  beforeEach(() => {
    delete window.location;
    window.location = { href: '' };
  });

  const queryClient = new QueryClient();

  const renderWithClient = (ui) =>
    render(<QueryClientProvider
client={queryClient}>{ui}</QueryClientProvider>);

  it('должен отправить данные формы и выполнить редирект', async () =>
{
    axios.post.mockResolvedValue({
      data: {
        projectId: 'project123'
```

```
    }
  });

  const { getByLabelText, getByText } = renderWithClient(
    <PopupCreateProject
      onOpenCreateProject={() => {}}
      onOpenAddMember={() => {}}
      openAddMember={false}
    />
  );

  fireEvent.change(getByLabelText(/название/i), {
    target: { value: 'Проект ВКР' }
  });

  fireEvent.change(getByLabelText(/описание/i), {
    target: { value: 'Описание проекта' }
  });

  const createBtn = getByText(/создать/i);
  fireEvent.click(createBtn);

  await waitFor(() => {
    expect(axios.post).toHaveBeenCalled();
    expect(window.location.href).toBe('/project/project123');
  });
 });
});
```

Приложение Б

Листинг файла PopupSettingsProjectEdit.test.jsx

```
import React from 'react';
import { render, fireEvent, waitFor } from '@testing-library/react';
import { describe, it, expect, vi, beforeEach } from 'vitest';
import PopupSettingsProject from './PopupSettingsProject';
import axios from 'axios';

vi.mock('axios');
vi.mock('react-router-dom', () => ({
  useNavigate: () => vi.fn(),
  useRevalidator: () => ({ revalidate: vi.fn() }),
  useRouteLoaderData: () => ({
    id: 'test123',
    title: 'Тестовый проект',
    description: 'Описание проекта',
    dates: [null, null],
    projectAvatar: '/images/avatar.png',
    complete: false
  })
}));

describe('handleProjectEdit', () => {
  beforeEach(() => {
    vi.clearAllMocks();
  });

  it('отправляет обновлённые данные проекта и выполняет переход',
async () => {
```

```
    axios.post.mockResolvedValue({ data: { success: true } });

    const { getByText } = render(
      <PopupSettingsProject
        onOpenSettingsProject={() => {}}
        onOpenConfirm={() => {}}
        openConfirm={false}
      />
    );

    fireEvent.click(getByText(/сохранить/i));

    await waitFor(() => {
      expect(axios.post).toHaveBeenCalledWith(
        'http://localhost:3000/project/test123/edit',
        expect.any(Object),
        { withCredentials: true }
      );
    });
  });
});
```

Приложение В

Листинг файла PopupSettingsProjectDelete.test.jsx

```
import React from 'react';
import { render, fireEvent, waitFor } from '@testing-library/react';
import { describe, it, expect, vi } from 'vitest';
import PopupConfirm from '../popupConfirm/PopupConfirm';
import axios from 'axios';

vi.mock('axios');

describe('PopupConfirm → handleProjectDelete', () => {
  it('выполняет axios.delete и переход на /boards', async () => {
    const navigateMock = vi.fn();

    vi.mocked(axios.delete).mockResolvedValue({ data: { success: true } });

    const mockDelete = () => {
      axios.delete('http://localhost:3000/project/test123', {
        withCredentials: true
      });
      navigateMock('/boards');
    };

    const { getByText } = render(
      <PopupConfirm
        handleProjectDelete={mockDelete}
        handleProjectComplete={() => {}}
        confirmTitle="удалить"
        onOpenConfirm={() => {}}
```

```
      />
    );

    fireEvent.click(getByText(/подтвердить/i));

    await waitFor(() => {
      expect(axios.delete).toHaveBeenCalledWith(
        'http://localhost:3000/project/test123',
        { withCredentials: true }
      );
    });

    expect(navigateMock).toHaveBeenCalledWith('/boards');
  });
});
```

Приложение Г

Листинг файла PopupCreateTask.test.jsx

```
import React from 'react';
import { render, fireEvent, waitFor } from '@testing-library/react';
import { describe, it, expect, vi } from 'vitest';
import { QueryClient, QueryClientProvider } from '@tanstack/react-query';
import PopupCreateTask from './PopupCreateTask';
import axios from 'axios';

vi.mock('axios');

describe('handleSubmit (PopupCreateTask)', () => {
 const queryClient = new QueryClient();

 const renderWithClient = (ui) =>
  render(<QueryClientProvider
client={queryClient}>{ui}</QueryClientProvider>);
 it('отправляет данные формы и выполняет post-запрос', async () => {
  const onOpenCreateTaskMock = vi.fn();
  axios.post.mockResolvedValue({ data: { success: true } });
  const { getByText, getByLabelText } = renderWithClient(
   <PopupCreateTask
    onOpenCreateTask={onOpenCreateTaskMock}
    projectId="project456"
    lastClickedStage="stage789"
   />
  );

  // Вводим данные
```

```
    fireEvent.change(getByLabelText(/название/i), {
      target: { value: 'Новая задача' }
    });
    fireEvent.change(getByLabelText(/описание/i), {
      target: { value: 'Описание задачи' }
    });
    // Кликаем по "Создать"
    const createBtn = getByText(/создать/i);
    fireEvent.click(createBtn);
    await waitFor(() => {
      expect(axios.post).toHaveBeenCalledWith(
        'http://localhost:3000/task/create',
        expect.objectContaining({
          projectId: 'project456',
          stageId: 'stage789',
          title: 'Новая задача',
          description: 'Описание задачи',
          priority: expect.any(Number),
          dates: expect.any(Array)
        }),
        { withCredentials: true }
      );
      // Проверяем вызов закрытия попапа
      expect(onOpenCreateTaskMock).toHaveBeenCalled();
    });
  });
});
```

Приложение Д

Листинг файла PopupSettingsTask.test.jsx

```
import React from 'react';
import { render, fireEvent, waitFor } from '@testing-library/react';
import { describe, it, expect, vi } from 'vitest';
import { QueryClient, QueryClientProvider } from '@tanstack/react-query';
import PopupSettingsTask from './PopupSettingsTask';
import axios from 'axios';

// Моки
vi.mock('axios');
vi.mock('react-router-dom', () => ({
  useParams: () => ({ taskId: 'task123' }),
  useNavigate: () => vi.fn()
}));

// Утилита для обёртки с QueryClientProvider
const renderWithClient = (ui) => {
  const queryClient = new QueryClient();
  return render(<QueryClientProvider client={queryClient}>{ui}</QueryClientProvider>);
};

describe('PopupSettingsTask', () => {
  beforeEach(() => {
    vi.clearAllMocks();
  });

  it('удаляет задачу при клике по кнопке "Удалить задачу"', async () => {
```

```
        axios.get.mockResolvedValueOnce({
          data: { id: 'task123', title: 'Test Task', description: 'Desc', priority: 2, subTasks: [] }
        });
        axios.get.mockResolvedValueOnce({ data: [] }); // комментарии
        axios.delete.mockResolvedValueOnce({ data: { success: true } });

        const { getByText } = renderWithClient(
          <PopupSettingsTask
            onOpenSettingsTask={() => {}}
            onOpenAddMember={() => {}}
            openAddMember={false}
          />
        );

        const deleteBtn = await waitFor(() => getByText(/удалить задачу/i));
        fireEvent.click(deleteBtn);

        await waitFor(() => {
          expect(axios.delete).toHaveBeenCalledWith(
            'http://localhost:3000/task/task123',
            { withCredentials: true }
          );
        });
      });

      it('отправляет POST-запрос при редактировании названия задачи', async () => {
```

```
    axios.get.mockResolvedValueOnce({
      data: { id: 'task123', title: 'Старая задача', description: 'Описание', priority: 2, subTasks: [] }
    });
    axios.get.mockResolvedValueOnce({ data: [] });
    axios.post.mockResolvedValueOnce({ data: { success: true } });

    const { getByLabelText } = renderWithClient(
      <PopupSettingsTask
        onOpenSettingsTask={() => {}}
        onOpenAddMember={() => {}}
        openAddMember={false}
      />
    );

    const titleInput = await waitFor(() => getByLabelText(/название/i));
    fireEvent.change(titleInput, { target: { value: 'Новая задача' } });
    fireEvent.blur(titleInput);

    await waitFor(() => {
      expect(axios.post).toHaveBeenCalledWith(
        'http://localhost:3000/task/task123/update/title',
        { value: 'Новая задача' },
        { withCredentials: true }
      );
    });
  });
```

```
it('отправляет POST-запрос и вызывает refetch при изменении приоритета', async () => {
  axios.get.mockResolvedValueOnce({
    data: { id: 'task123', title: 'Task', description: '...', priority: 2, subTasks: [] }
  });
  axios.get.mockResolvedValueOnce({ data: [] });
  axios.post.mockResolvedValueOnce({ data: { success: true } });

  const { getByText } = renderWithClient(
    <PopupSettingsTask
      onOpenSettingsTask={() => {}}
      onOpenAddMember={() => {}}
      openAddMember={false}
    />
  );

  const priorityBtn = await waitFor(() => getByText('3'));
  fireEvent.click(priorityBtn);

  await waitFor(() => {
    expect(axios.post).toHaveBeenCalledWith(
      'http://localhost:3000/task/task123/update/priority',
      { value: 3 },
      { withCredentials: true }
    );
  });
});
```

```
        it('отправляет комментарий и очищает поле', async () => {
          axios.get
            .mockResolvedValueOnce({
              data: { id: 'task123', title: 'Task', description: '...', priority: 2, subTasks: [] }
            })
            .mockResolvedValueOnce({ data: [] }); // comments

          axios.post.mockResolvedValueOnce({ data: { success: true } });

          const { getByPlaceholderText, getByText } = renderWithClient(
            <PopupSettingsTask
              onOpenSettingsTask={() => {}}
              onOpenAddMember={() => {}}
              openAddMember={false}
            />
          );

          const commentInput = await waitFor(() => getByPlaceholderText(/напишите комментарий/i));
          fireEvent.change(commentInput, { target: { value: 'Это мой комментарий' } });

          const sendButton = getByText(/отправить/i);
          fireEvent.click(sendButton);

          await waitFor(() => {
            expect(axios.post).toHaveBeenCalledWith(
```

```
        'http://localhost:3000/task/task123/comment',
        expect.objectContaining({
          text: 'Это мой комментарий',
          date: expect.any(Date)
        }),
        { withCredentials: true }
      );

      // Проверка, что поле очистилось
      expect(commentInput.value).toBe('');
    });
  });

});
```

Приложение Е

Результаты тестирования

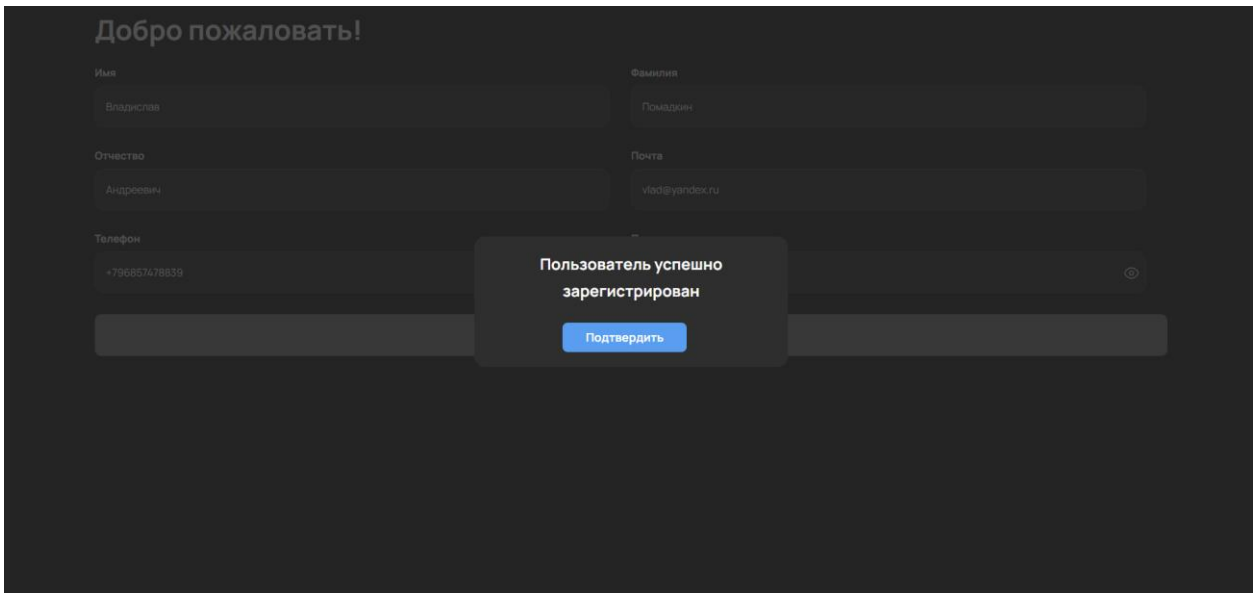

Рисунок Е1 – Успешная регистрация

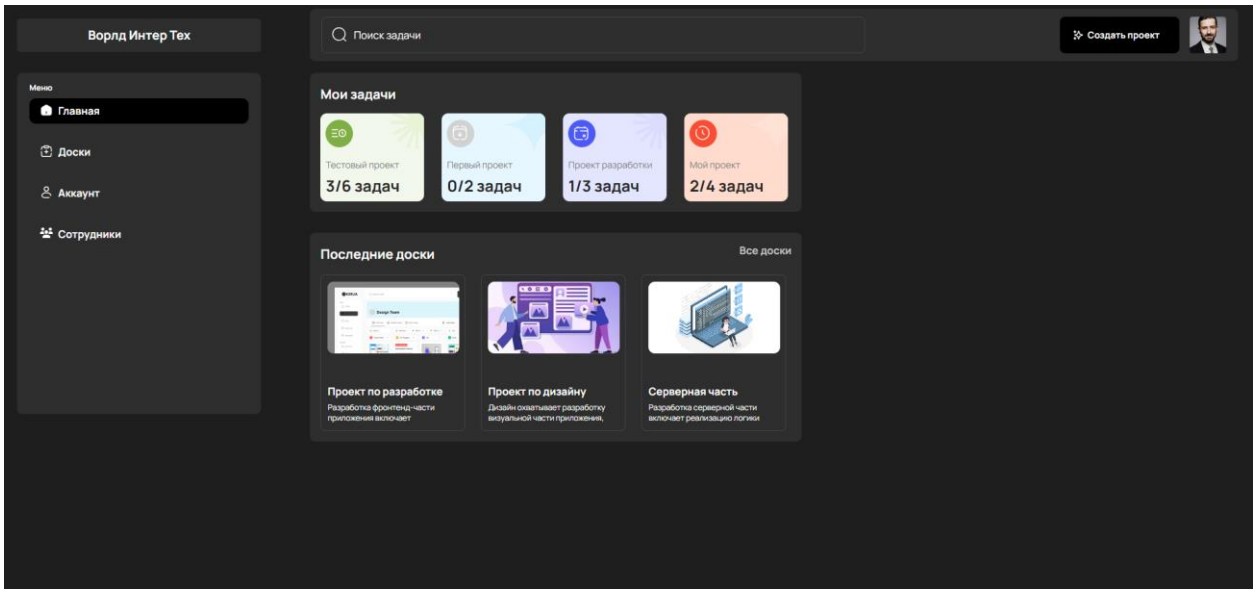

Рисунок Е2 – Успешная авторизация

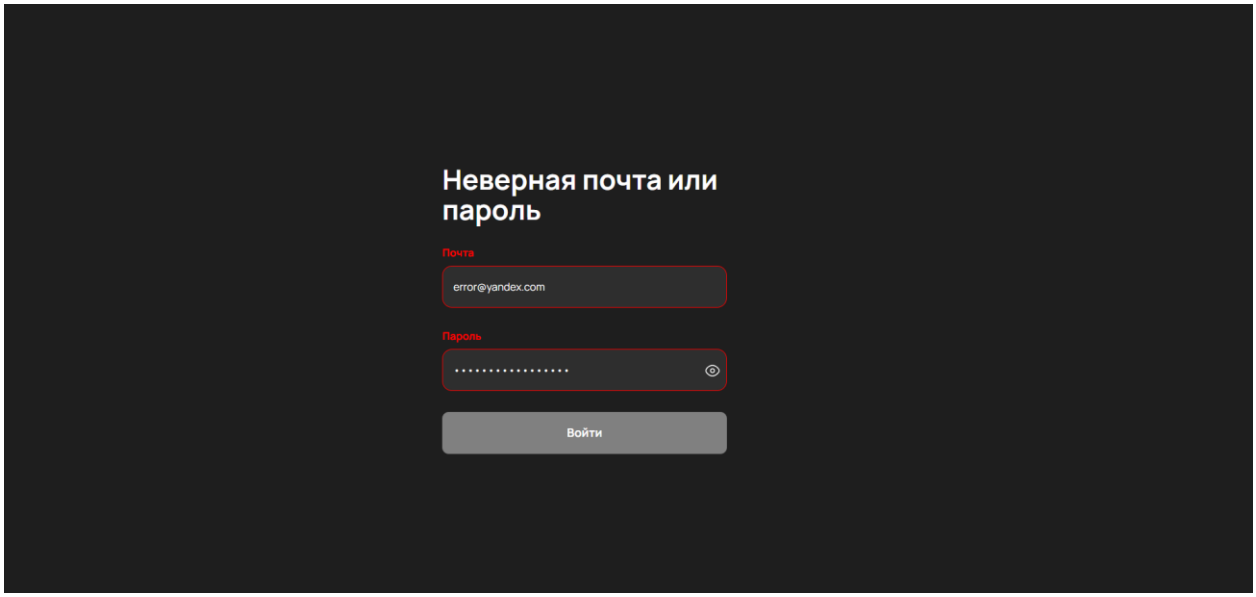

Рисунок Е3 – Неверная почта или пароль

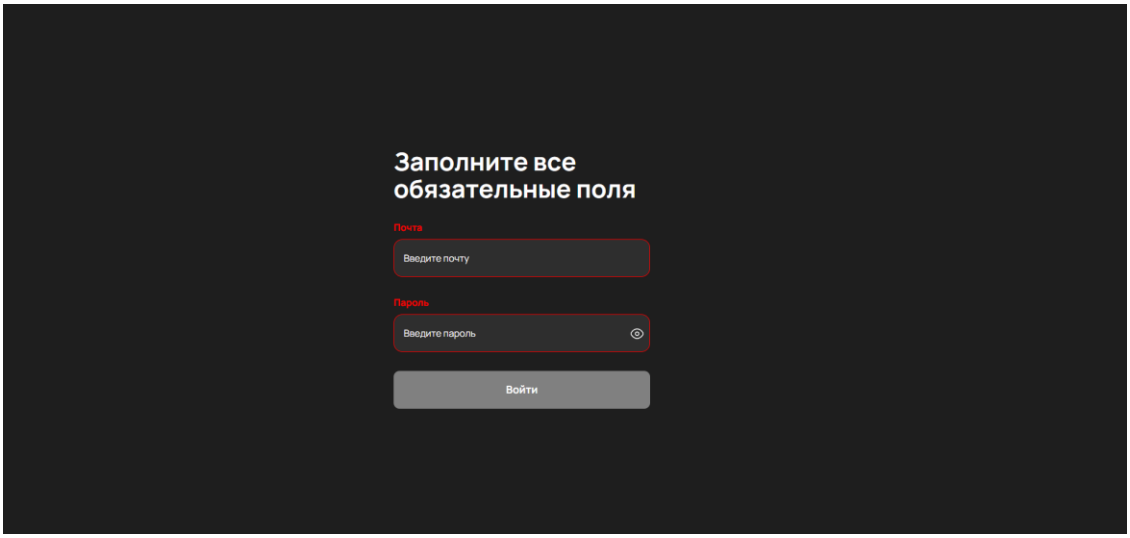

Рисунок Е4 – Не заполнены ключевые поля при авторизации

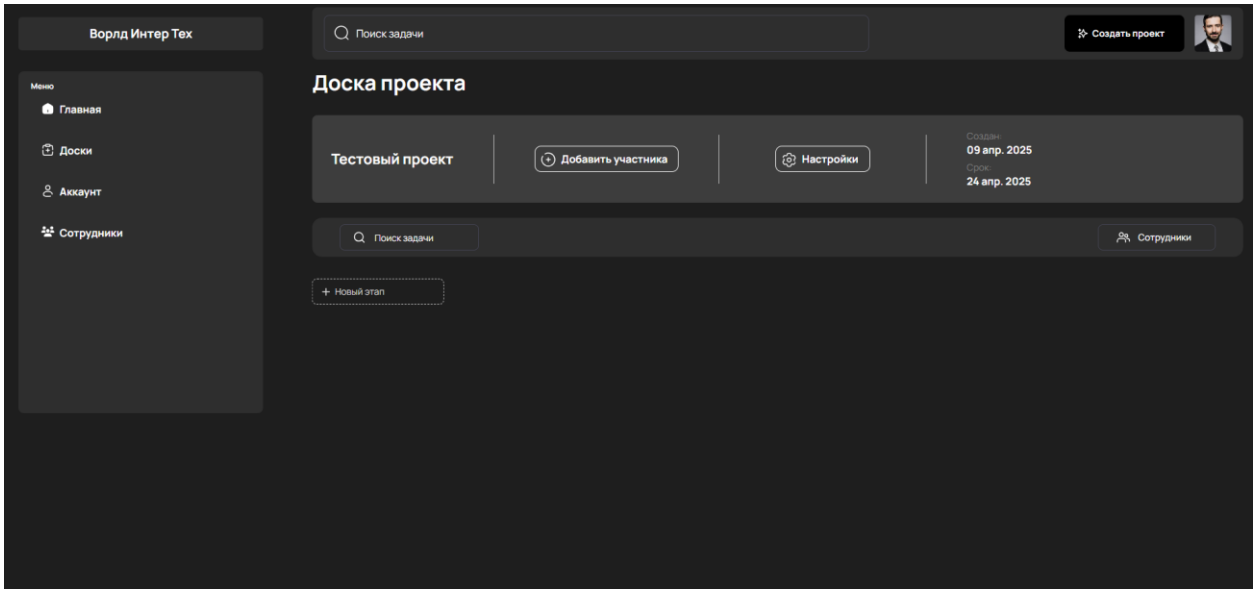

Рисунок Е5 – Успешное создание проекта

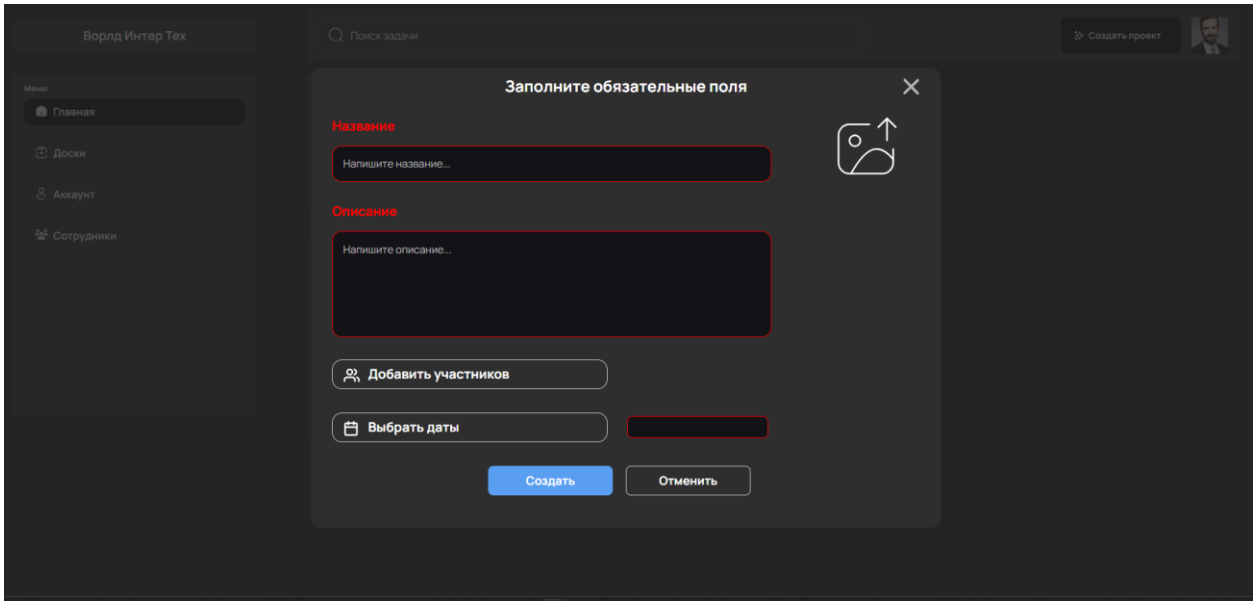

Рисунок Е6 – Не написаны ключевые поля при создании проекта

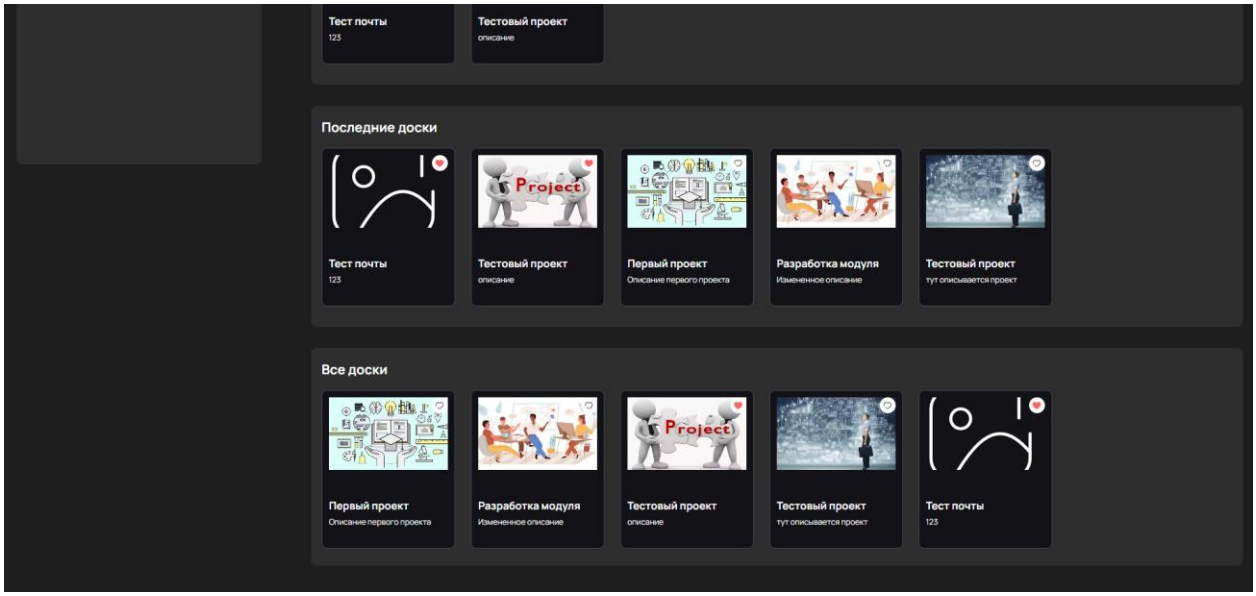

Рисунок Е7 – Все доски

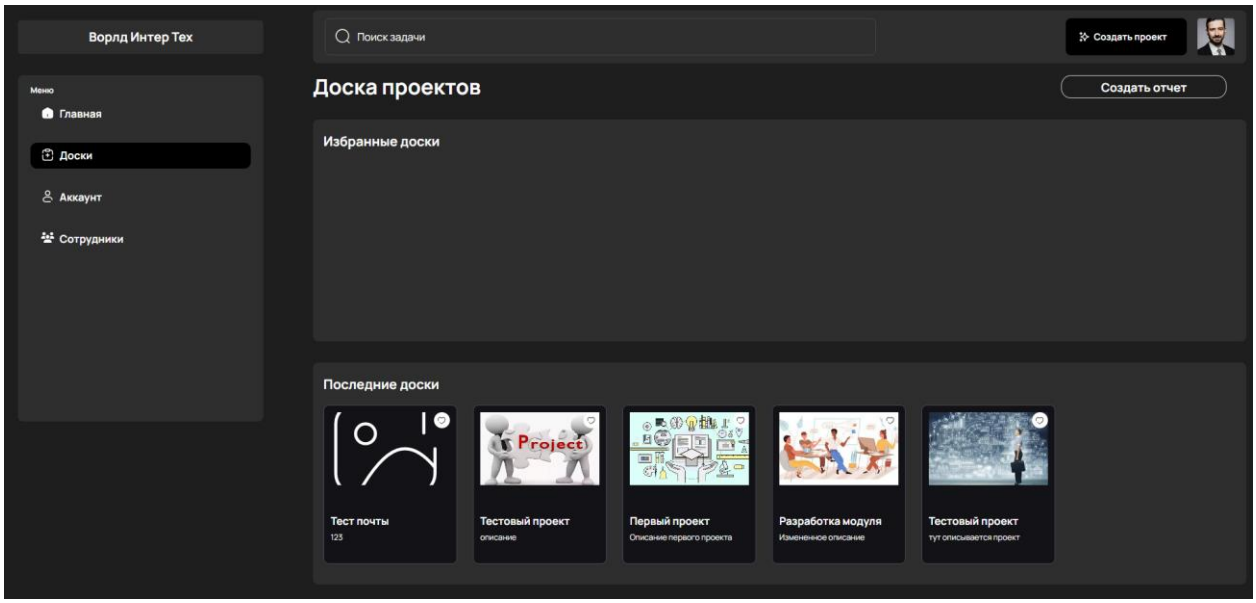

Рисунок Е8 – Последние доски

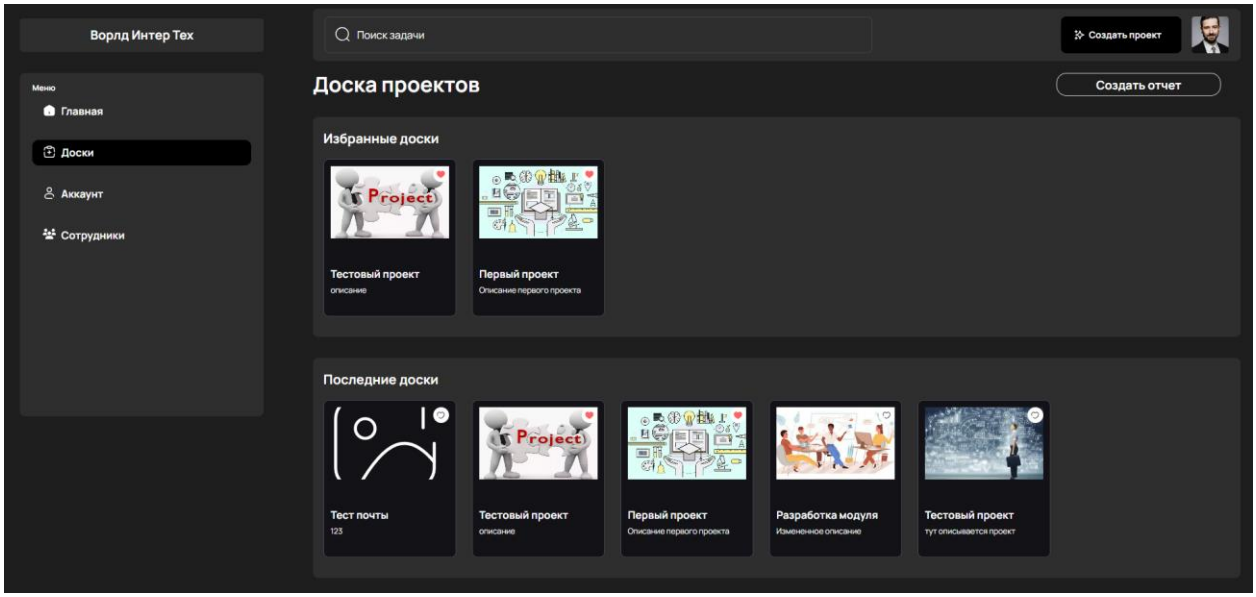

Рисунок Е9 – Избранные доски

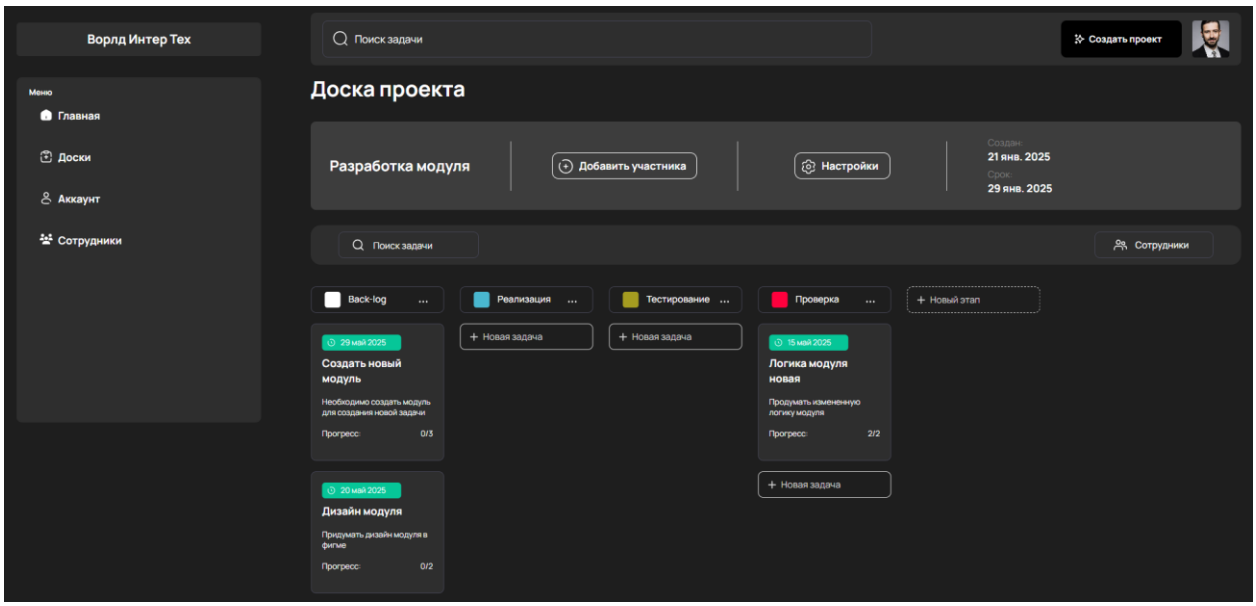

Рисунок Е10 – Изменение настроек проекта

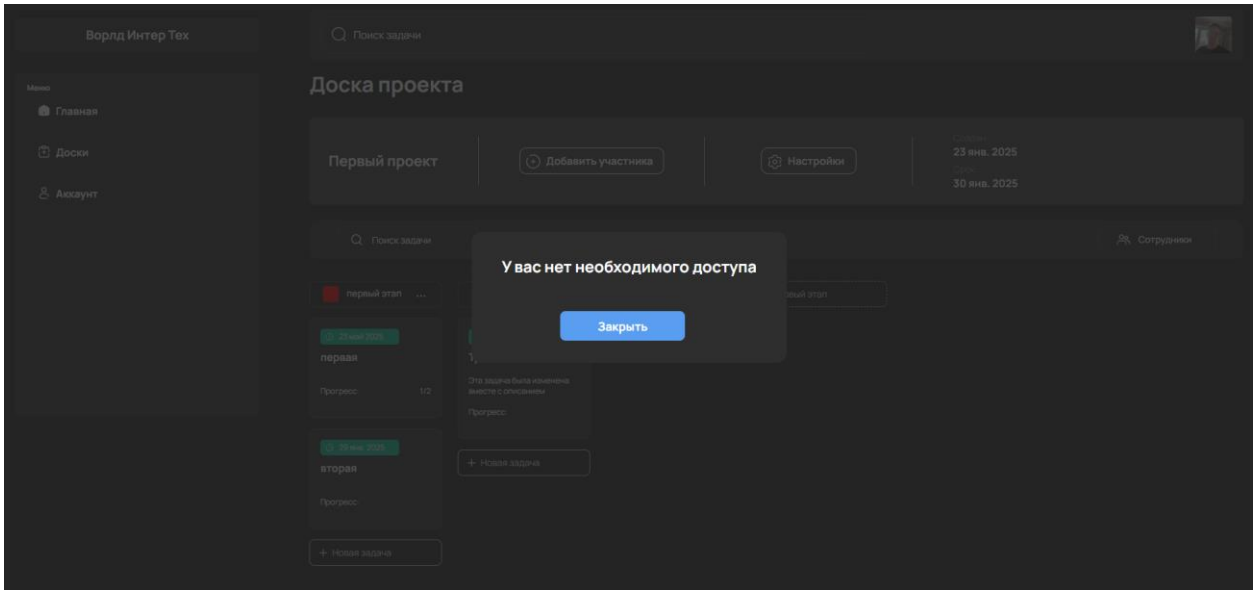

Рисунок Е11 – Ошибка прав доступа

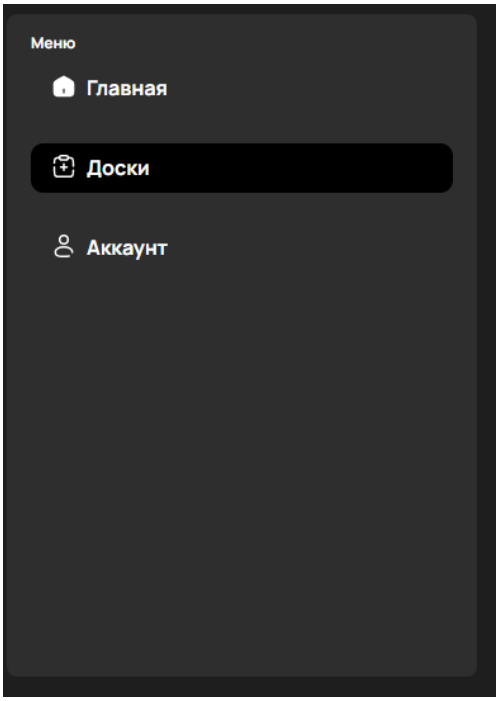

Рисунок Е12 – Меню пользователя с ролью «Сотрудник»

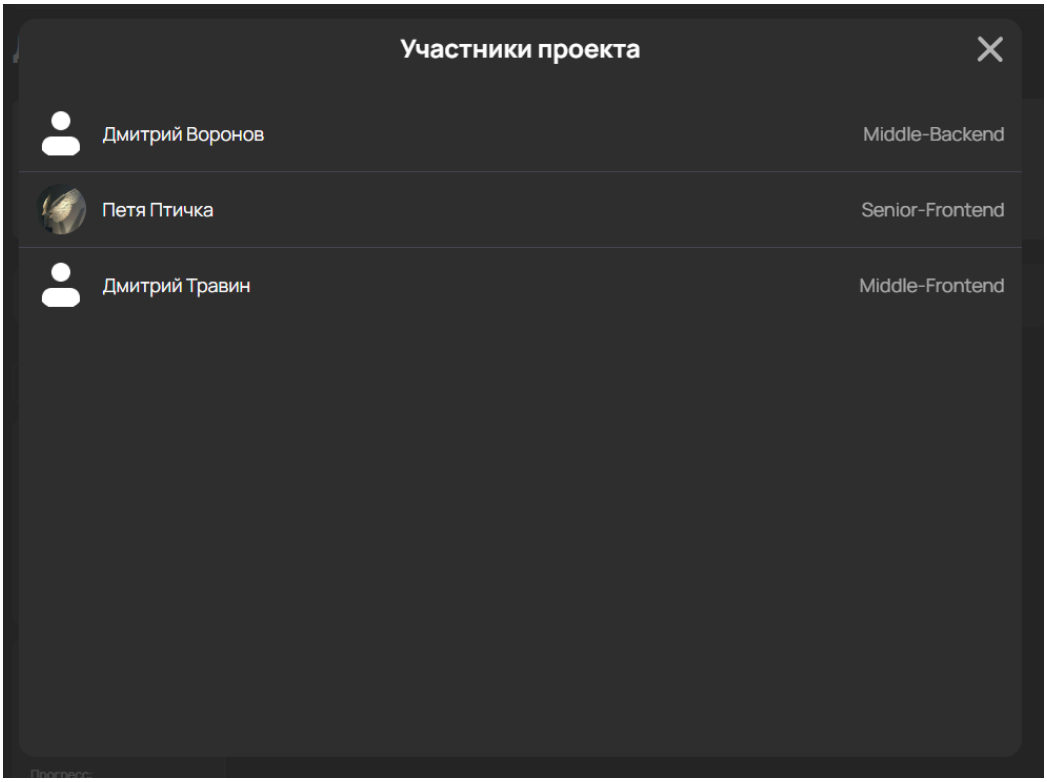

Рисунок Е13 – Отсутствие кнопки «Удалить» у пользователя с ролью «Сотрудник»

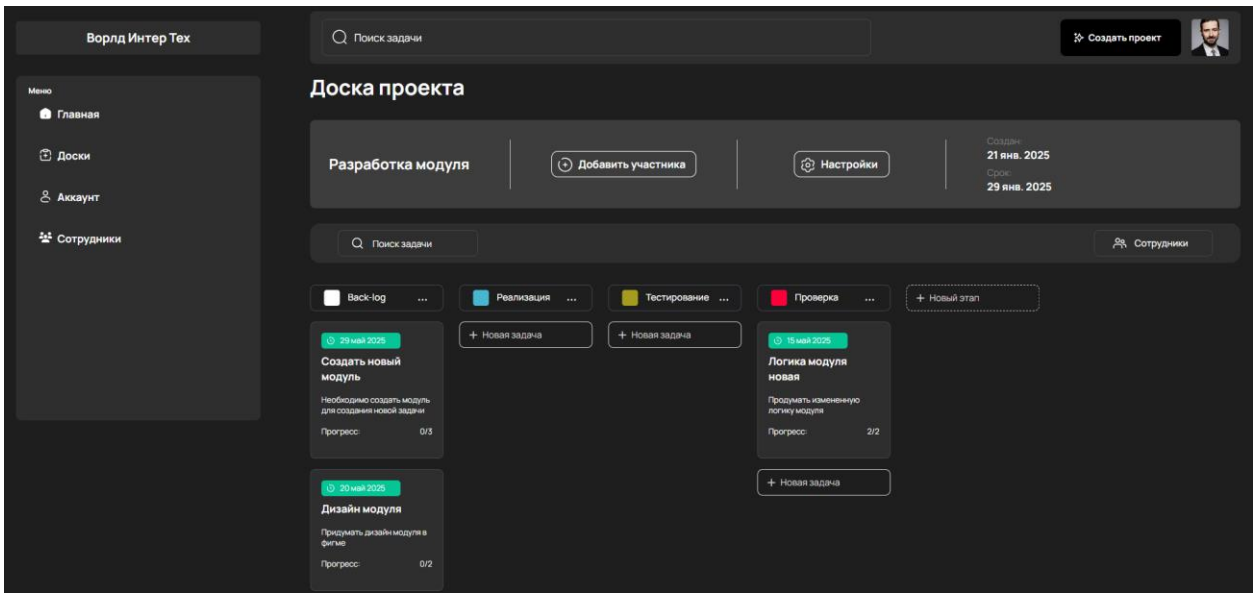

Рисунок Е14 – Структуризация проекта

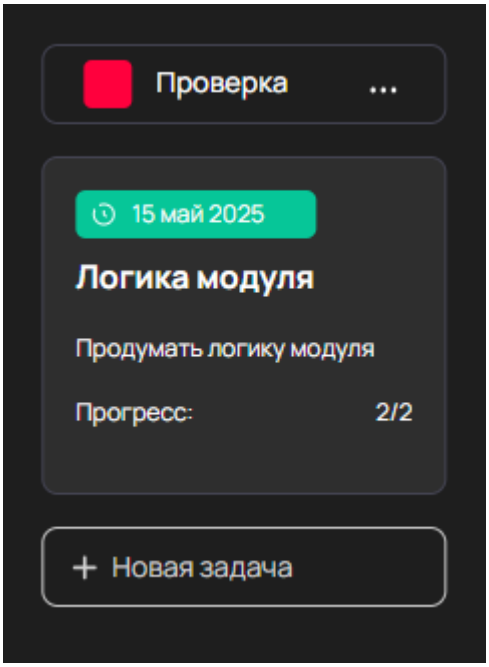

Рисунок Е15 – Задача до изменения

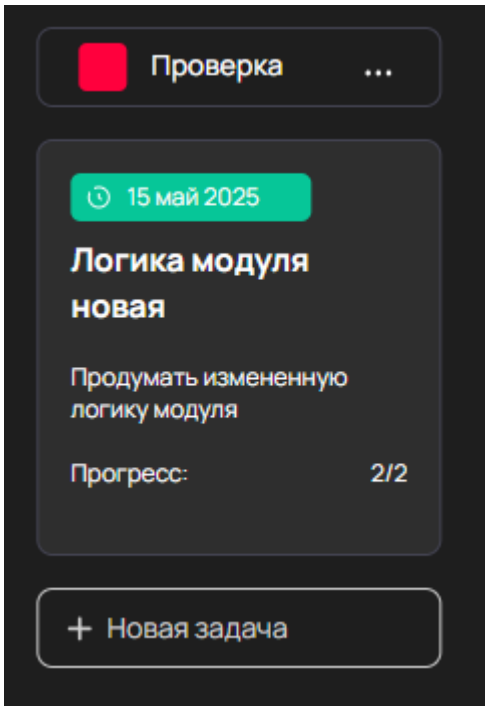

Рисунок Е16 – Задача после изменения

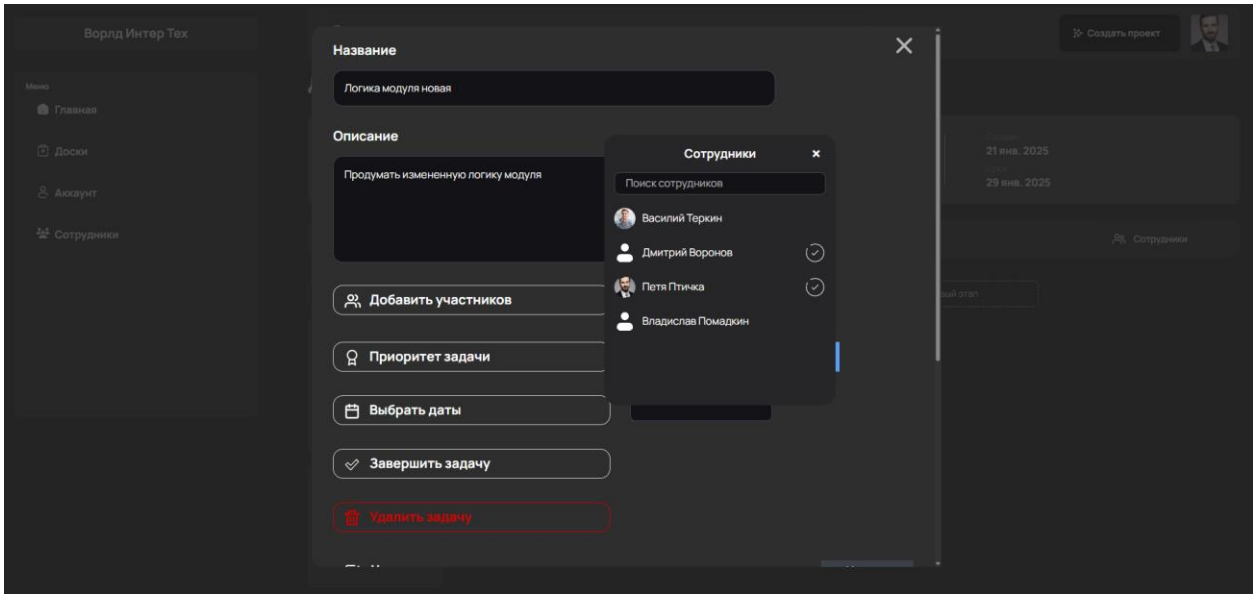

Рисунок Е17 – Добавление участников в задачу

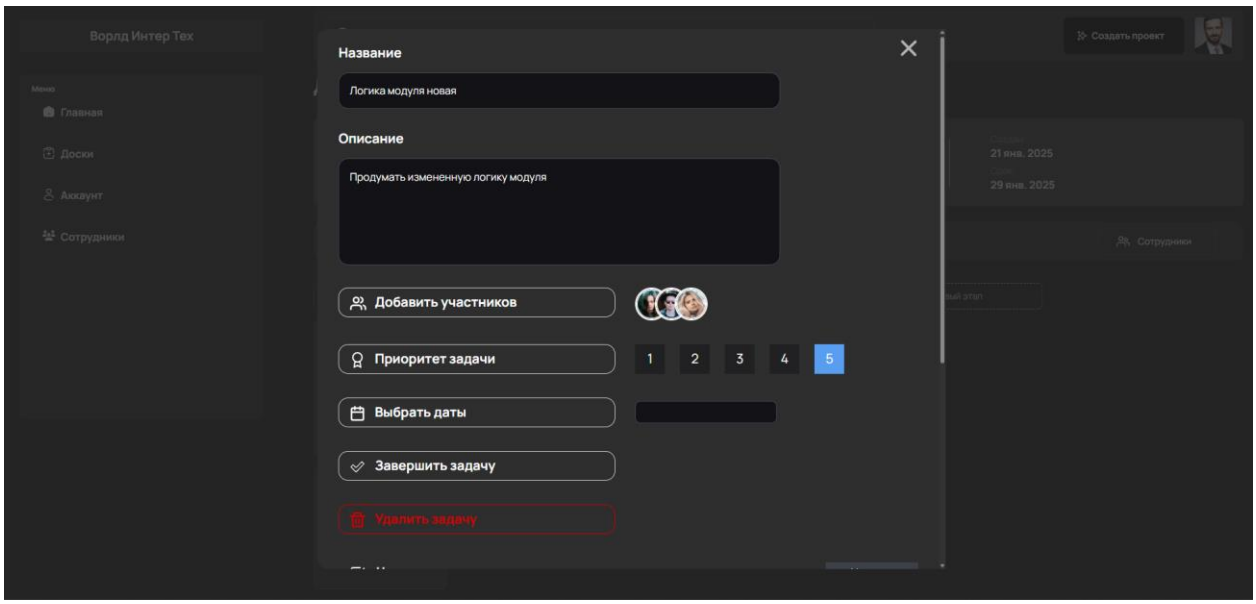

Рисунок Е18 – Выбор приоритета задачи

Продолжение приложения Е

Рисунок Е19 – Создание и проверка подзадач

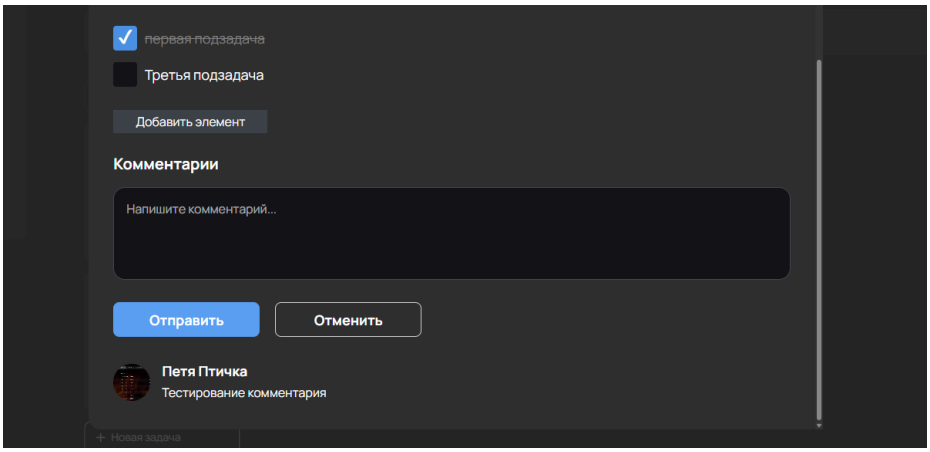

Рисунок Е20 – Создание нового комментария

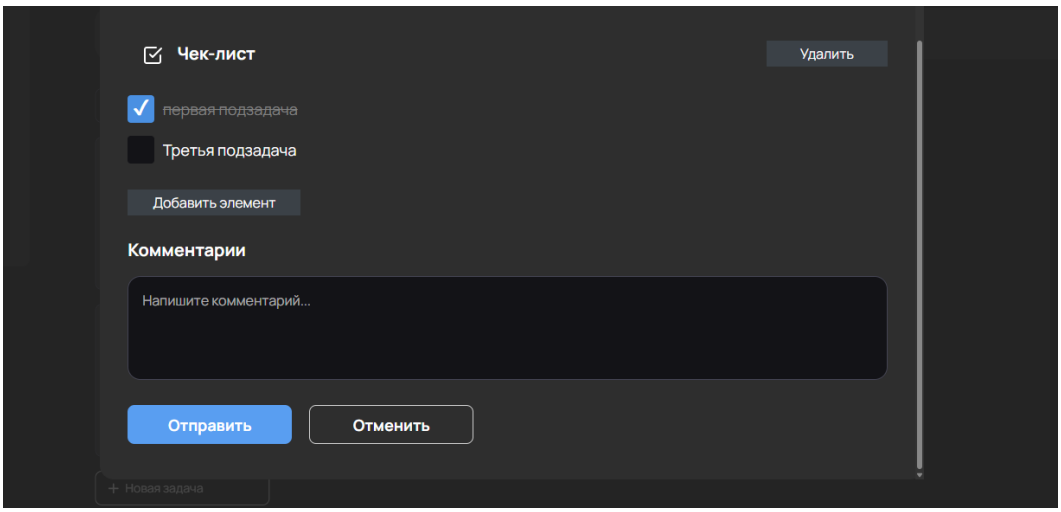

Рисунок Е21 – Удаление комментария

Продолжение приложения Е

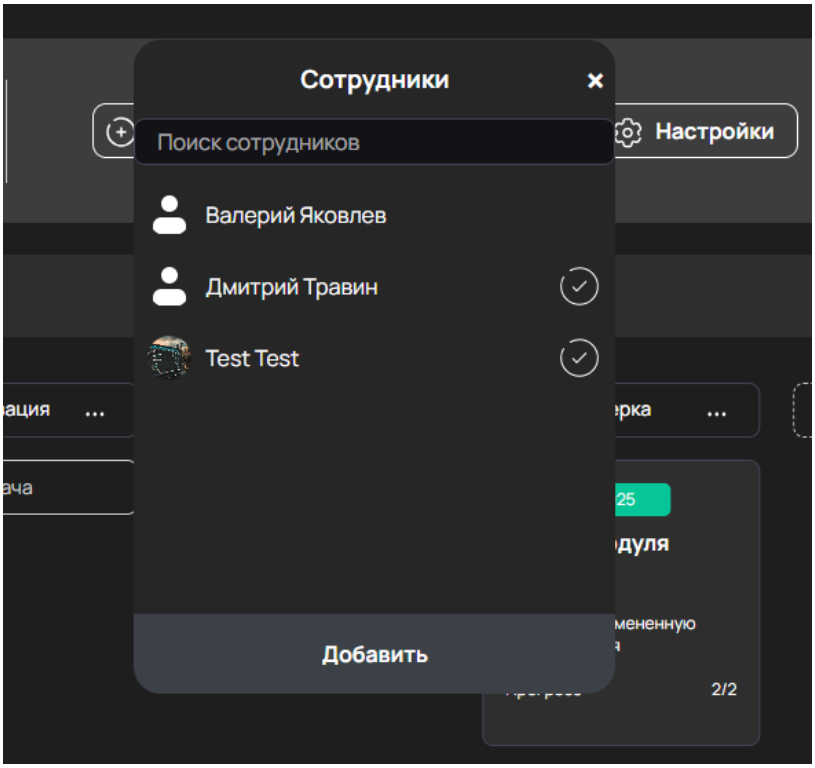

Рисунок Е22 – Выбор сотрудников для добавления в проект

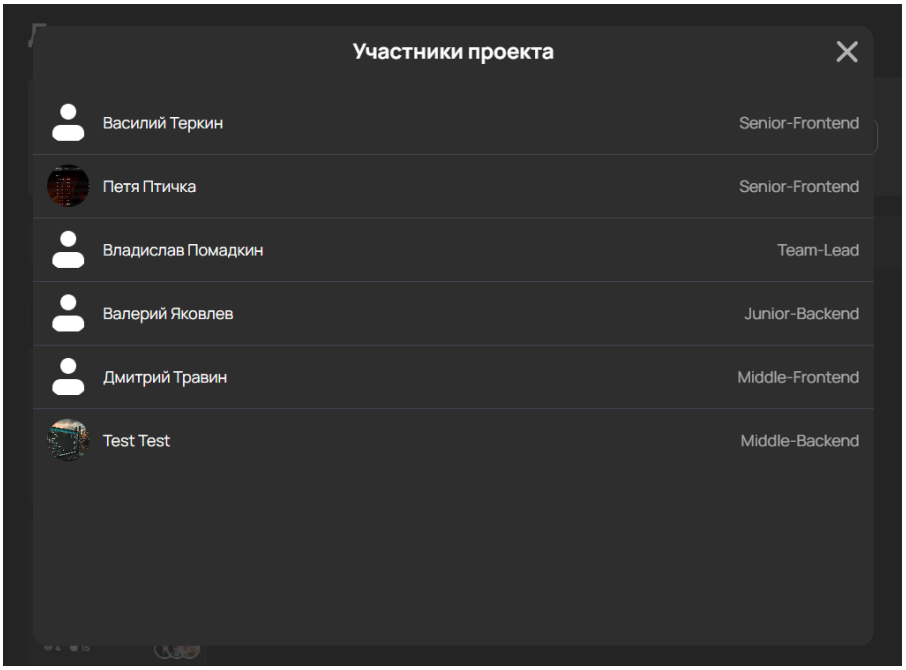

Рисунок Е23 – Отображение всех сотрудников проекта

Продолжение приложения Е

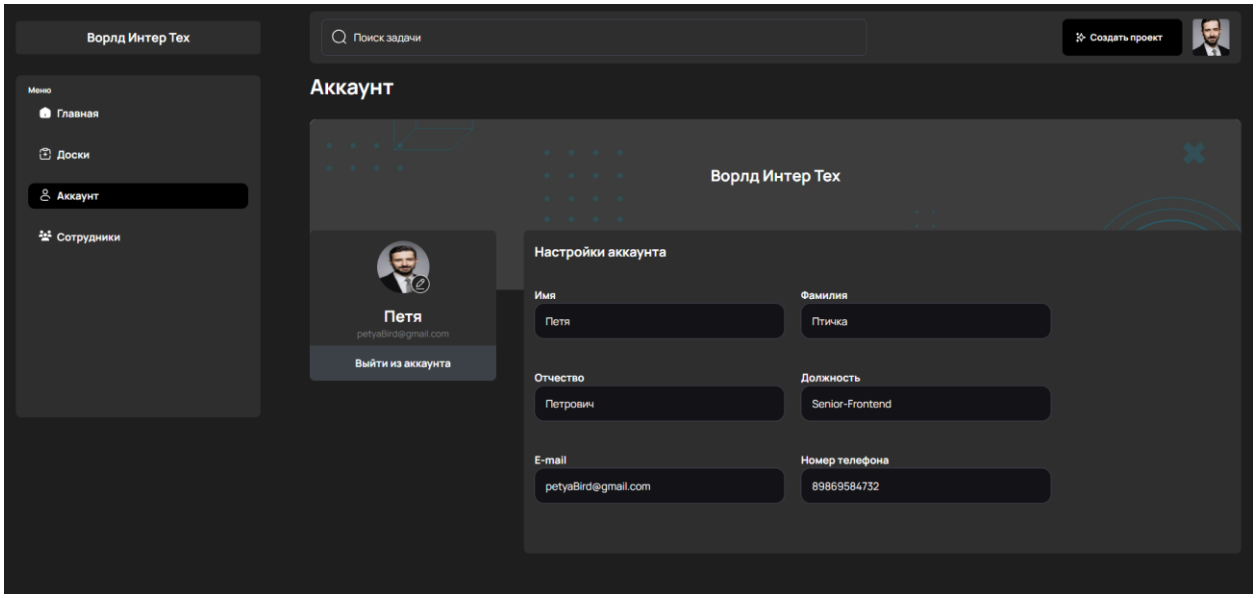

Рисунок Е24 – Отображение данных сотрудника

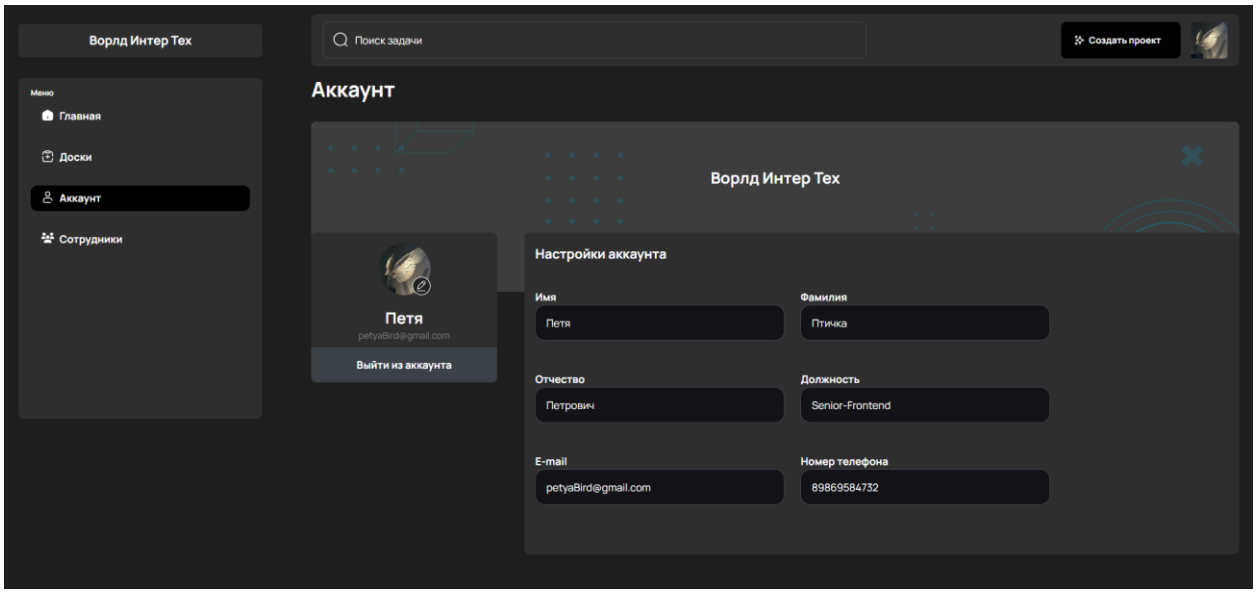

Рисунок Е25 – Изменение аватарки пользователя

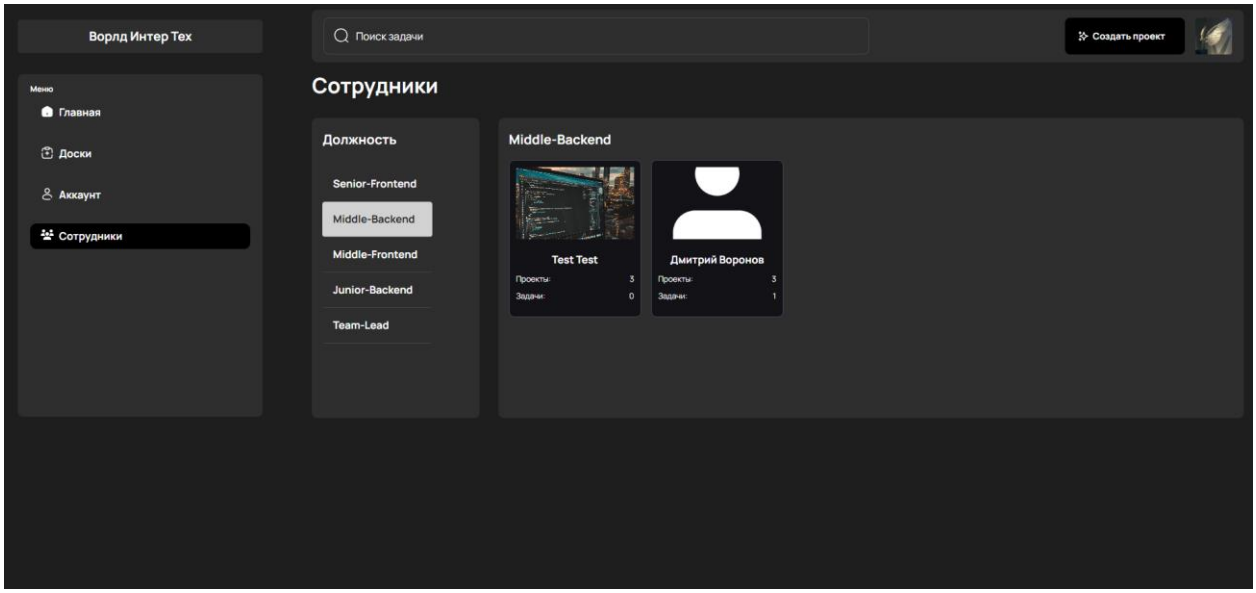

Рисунок Е26 – Отображение сотрудников во вкладке «Сотрудники»

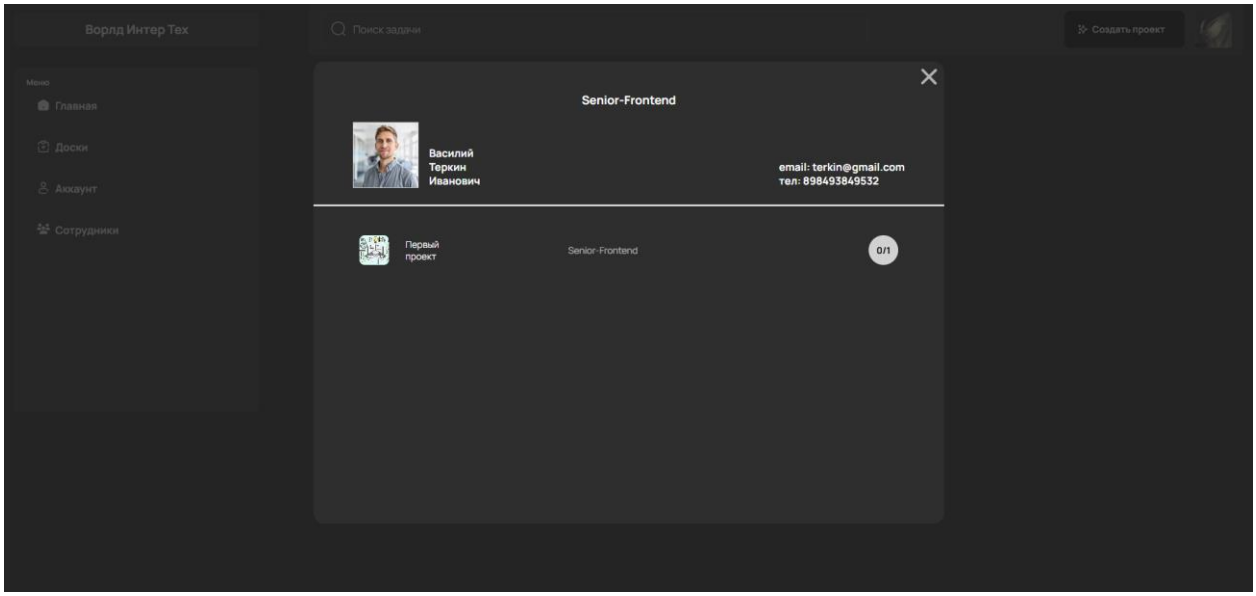

Рисунок Е27 – Подробная информация о сотруднике

Продолжение приложения Е


```
_id: ObjectId('678d6ba126b343592701b4be')
id: "01b78d8c-0f1b-4158-bfdf-290b89018b6e"
firstName: "Василий"
lastName: "Теркин"
password: "$2b$10$nm.40kce.fjVn4o9a9wGlOQPt8RNKb.eQ5IMquAG1Ji32sCiGbWK6"
middleName: "Иванович"
number: "898493849532"
email: "terkin@gmail.com"
avatarBase64: "/images/defaultUser.png"
favoriteProjects: Array (empty)
lastVisited: Array (empty)
__v: 0
position: "Senior-Frontend"
```


Рисунок Е28 – Пароль пользователя в БД

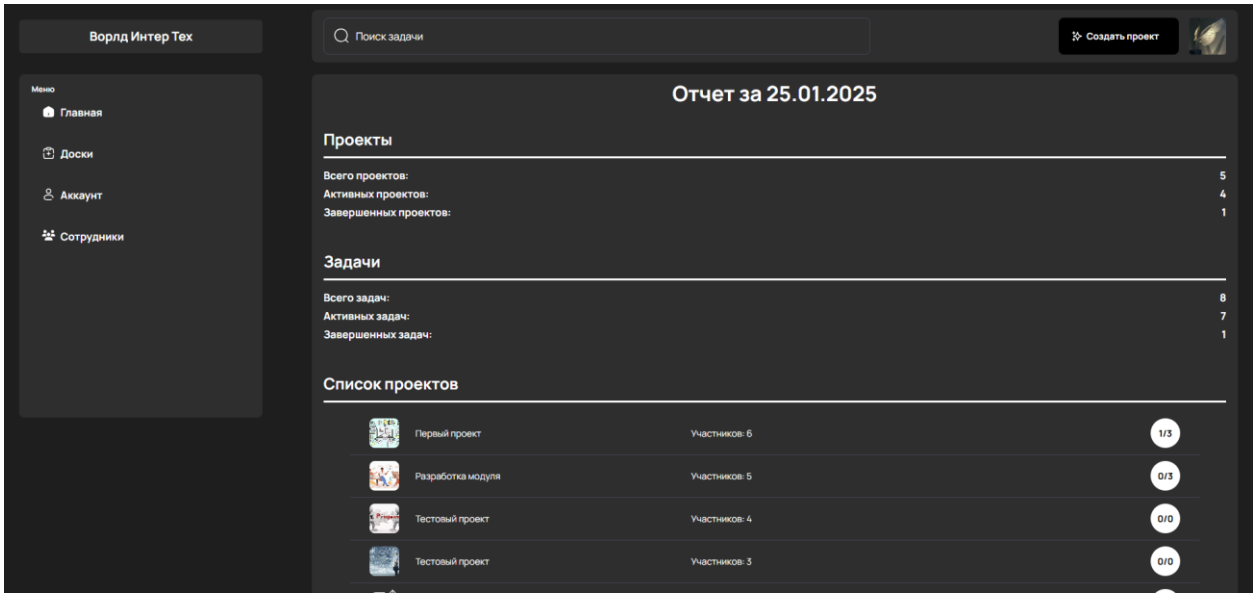

Рисунок Е29 – Создание отчёта

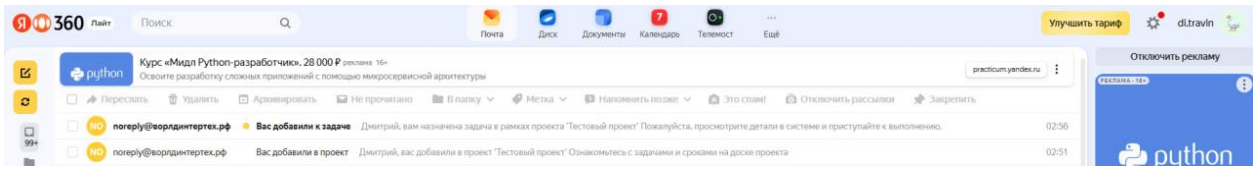

Рисунок Е30 – Уведомления на почте сотрудника