Титульный лист

Лист задания

**Содержание**

Ф.И.О

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

*Яковлева Е.И.*

Пров.

Кантарович В.С.

Н. контр.

Утв.

Смелов В.В.

Содержание

Лит.

Листов

2

*74218051, 2023*

У

Николайчук А.Н.

[Введение 7](#_Toc136987313)

[1 Постановка задачи и обзор аналогичных решений 8](#_Toc136987314)

[1.1 Постановка задачи 8](#_Toc136987315)

[1.2 Обзор аналогов 9](#_Toc136987316)

[1.2.1 Приложение «Trello» 9](#_Toc136987317)

[1.2.2 Приложение «Notion» 10](#_Toc136987318)

[1.2.3 Приложение «Jira» 11](#_Toc136987319)

[1.3 Патентный поиск 12](#_Toc136987320)

[1.4 Выводы по разделу 13](#_Toc136987321)

[2 Проектирование программного средства 14](#_Toc136987322)

[2.1 Диаграмма вариантов использования 14](#_Toc136987323)

[2.2 Выбор средств реализации 15](#_Toc136987324)

[2.2.1 Основные языки программирования 15](#_Toc136987325)

[2.2.2 Фреймворки и библиотеки 16](#_Toc136987326)

[2.2.3 Система управления базами данных 17](#_Toc136987327)

[2.2.4 Вспомогательные инструменты 18](#_Toc136987328)

[2.3 Авторизация пользователей 19](#_Toc136987329)

[2.4 Контейнеризация 20](#_Toc136987330)

[2.5 Проектирование базы данных 22](#_Toc136987331)

[2.5.1 Таблица roles 23](#_Toc136987332)

[2.5.2 Таблица users 23](#_Toc136987333)

[2.5.3 Таблица employees 23](#_Toc136987334)

[2.5.4 Таблица tickets 24](#_Toc136987335)

[2.5.5 Таблица projects\_has\_employees 24](#_Toc136987336)

[2.5.6 Таблица comments 25](#_Toc136987337)

[2.5.7 Таблица logged\_time 25](#_Toc136987338)

[2.5.8 Таблица orders 26](#_Toc136987339)

[2.6 Проектирование основных алгоритмов 26](#_Toc136987340)

[2.6.1 Алгоритм перемещения задачи на «дэшборде» 26](#_Toc136987341)

[2.6.2 Алгоритм изменения прикрепленного к задаче файла 27](#_Toc136987342)

[2.6.3 Алгоритм авторизации заказчика в приложении 29](#_Toc136987343)

[2.6.4 Алгоритм изменения пароля от аккаунта 30](#_Toc136987344)

[2.7 Выводы по разделу 31](#_Toc136987345)

[3 Реализация программного средства 32](#_Toc136987346)

[3.1 Общая архитектура приложения 32](#_Toc136987347)

[3.2 Разработка серверной части 33](#_Toc136987348)

[3.2.1 Установка необходимых компонентов 33](#_Toc136987349)

[3.2.2 Выбор архитектуры 34](#_Toc136987350)

[3.2.3 Разработка контроллеров 37](#_Toc136987351)

[3.2.4 Разработка сервисов 38](#_Toc136987352)

[3.2.5 Разработка репозиториев 38](#_Toc136987353)

[3.3 Разработка клиентской части 39](#_Toc136987354)

[3.3.1 Установка необходимых компонентов 39](#_Toc136987355)

[3.3.2 Выбор архитектуры проекта 40](#_Toc136987356)

[3.3.3 Маршрутизация в приложении 41](#_Toc136987357)

[3.3.4 Валидация данных 42](#_Toc136987358)

[3.4 Выводы по разделу 43](#_Toc136987359)

[4 Тестирование программного средства 44](#_Toc136987360)

[4.1 Тестирование серверной части 44](#_Toc136987361)

[4.2 Тестирование клиентской части 49](#_Toc136987362)

[4.3 Выводы по разделу 54](#_Toc136987363)

[5 Руководство пользователя 55](#_Toc136987364)

[5.1 Развертывание приложения в среде Docker 55](#_Toc136987365)

[5.2 Руководство для сотрудника 57](#_Toc136987366)

[5.3 Руководство для менеджера 64](#_Toc136987367)

[5.4 Руководство для администратора 65](#_Toc136987368)

[5.5 Руководство для клиента 67](#_Toc136987369)

[5.6 Выводы по разделу 68](#_Toc136987370)

[6 Технико-экономическое обоснование проекта 69](#_Toc136987371)

[6.1 Общая характеристика разрабатываемого программного средства 69](#_Toc136987372)

[6.2 Исходные данные для проведения расчетов и маркетинговый анализ 70](#_Toc136987373)

[6.3 Обоснование цены программного средства 71](#_Toc136987374)

[6.3.1 Расчет затрат рабочего времени на разработку программного средства 71](#_Toc136987375)

[6.3.2 Расчет основной заработной платы 72](#_Toc136987376)

[6.3.3 Расчет дополнительной заработной платы 72](#_Toc136987377)

[6.3.4 Расчет отчислений в Фонд социальной защиты населения и по страхованию 73](#_Toc136987378)

[6.3.5 Расчет суммы прочих прямых затрат 73](#_Toc136987379)

[6.3.6 Расчет суммы накладных расходов 74](#_Toc136987380)

[6.3.7 Сумма расходов на разработку программного средства 74](#_Toc136987381)

[6.3.8 Определение цены, оценка эффективности 75](#_Toc136987382)

[6.4 Выводы по разделу 76](#_Toc136987383)

[Заключение 77](#_Toc136987384)

[Список использованных источников 78](#_Toc136987385)

[ПРИЛОЖЕНИЕ А Диаграмма вариантов использования 80](#_Toc136987386)

[ПРИЛОЖЕНИЕ Б Листинг классов 81](#_Toc136987388)

[ПРИЛОЖЕНИЕ В Логическая схема базы данных 88](#_Toc136987390)

[ПРИЛОЖЕНИЕ Г Блок-схема алгоритма перемещения задачи на «дэшборде» 89](#_Toc136987392)

[ПРИЛОЖЕНИЕ Д Блок-схема алгоритма изменения прикрепленного к задаче файла 90](#_Toc136987394)

[ПРИЛОЖЕНИЕ Е Блок-схема алгоритма авторизации заказчика в приложении 91](#_Toc136987396)

[ПРИЛОЖЕНИЕ Ж Блок-схема алгоритма изменения пароля от аккаунта 92](#_Toc136987398)

[ПРИЛОЖЕНИЕ И Структурная схема архитектуры приложения 93](#_Toc136987400)

[ПРИЛОЖЕНИЕ К Листинг классов SecurityConfig и JwtFilter 94](#_Toc136987402)

Введение

Ф.И.О

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

*Яковлева Е.И.*

Пров.

Кантарович В.С.

Н. контр.

Утв.

Смелов В.В.

Введение

Лит.

Листов

1

*74218051, 2023*

У

Николайчук А.Н.

В современном мире, где повышенная мобильность и требования к качеству жизни становятся нормой, аренда жилья превращается в сложный процесс для обеих сторон — арендаторов и арендодателей. Постоянное увеличение количества информации о доступной недвижимости, условиях съёма и предпочтениях арендаторов подчёркивает необходимость эффективных инструментов для упрощения этого процесса. Веб-приложения, разработанные для таких целей, играют ключевую роль, предлагая удобные механизмы поиска и аренды жилья.

В рамках данной курсовой работы разработано веб-приложение, ориентированное на аренду жилья. Приложение включает функционал поиска, фильтрации, визуализации объектов недвижимости и коммуникации между участниками сделки. Основная задача проекта — создание удобного и простого в использовании интерфейса, который поможет пользователям находить подходящие предложения, сравнивать их и легко взаимодействовать с арендодателями.

Такой подход обеспечивает надёжность, высокую производительность и масштабируемость решения. Принципы модульности и асинхронности, лежащие в основе архитектуры, гарантируют гибкость системы, которая может адаптироваться к изменениям и запросам рынка.

Проект охватывает полный цикл разработки: от анализа требований и проектирования структуры данных до внедрения ключевых функций и тестирования. Итоговая система станет полезным инструментом как для арендаторов, ищущих жильё, соответствующее их запросам, так и для арендодателей, стремящихся найти надёжных жильцов.

Таким образом, данное приложение представляет собой комплексное решение, призванное облегчить процесс аренды жилья, минимизирую усилия пользователей и повышая их удовлетворённость результатом.

1. Постановка задачи и обзор аналогичных решений

Ф.И.О

Подпись

Дата

Лист

1

БГТУ 01.00.ПЗ

Разраб.

*Яковлева Е.И.*

Пров.

Кантарович В.С.

Н. контр.

Утв.

Смелов В.В.

1 Постановка задачи и обзор аналогичных решений

Лит.

Листов

6

*74218051, 2023*

У

Николайчук А.Н.

Для разработки программного средства первоначально требуется проанализировать существующие аналоги, выделив их ключевые преимущества и недостатки, а также определив основные характеристики приложений данного типа. На основе этого анализа формируются задачи для создания программы. Все указанные шаги были реализованы в данной главе.

* 1. Обзор аналогов

Обзор аналогов — важный этап перед началом разработки приложения. Этот процесс позволяет выявить функции, которые приложение должно обязательно включать, определить уникальные возможности, которые станут его отличительной чертой, а также выявить недостатки, которых следует избегать при проектировании.

На рынке уже существует множество платформ для аренды жилья. Однако, многие из них имеют недостатки: ограниченная кастомизация поиска, сложность управления объявлениями для владельцев или недостаточное взаимодействие между арендаторами и арендодателями. Это открывает возможность для создания продукта, который предложит гибкий функционал, простой интерфейс и эффективное взаимодействие между пользователями.

* + 1. Приложение «Booking»

*Booking* [1] — это одна из ведущих платформ для поиска и бронирования жилья, которая предоставляет пользователям возможность находить подходящие варианты аренды, используя удобные фильтры, просматривать подробные описания объектов и бронировать жильё в режиме онлайн. Пример интерфейса данного сервиса представлен на рисунке 1.1.

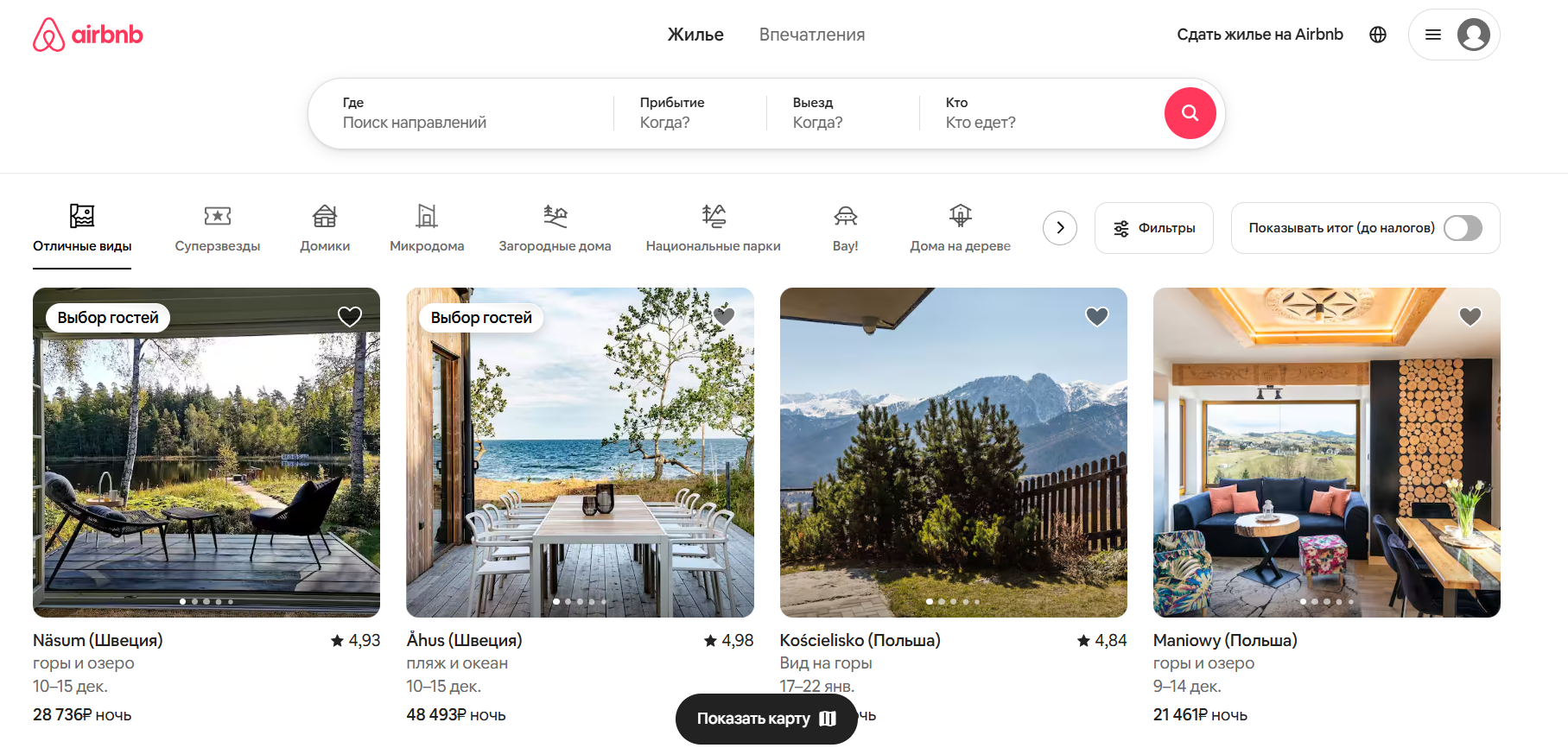


Рисунок 1.1 – Интернет-ресурс «Booking»

Одной из ключевых функций *Booking* является поиск жилья. Пользователи могут искать объекты недвижимости, применяя фильтры по местоположению, дате, цене, количеству гостей, рейтингу, удобствам и другим параметрам. Сервис также предоставляет персонализированные рекомендации на основе истории поиска и ранее забронированных объектов.

Для удобства пользователей *Booking* предлагает возможность создавать учётные записи, где можно управлять бронированиями, сохранять интересующие объекты в "Избранное" и оставлять отзывы после завершения проживания. Пользователи могут ознакомиться с отзывами других гостей, что помогает сделать более информированный выбор.

Для владельцев недвижимости *Booking* предоставляет платформу для регистрации и управления своими объектами. Они могут добавлять описания, фотографии, устанавливать цены, задавать правила отмены бронирования, а также управлять календарём доступности. Сервис предоставляет статистику по каждому объекту, включая количество просмотров, бронирований и средний рейтинг, что помогает владельцам анализировать спрос и оптимизировать свои предложения.

С технической стороны *Booking* использует сложные алгоритмы поиска и фильтрации, а также высокопроизводительные базы данных для обработки миллионов запросов пользователей ежедневно.

Эта платформа выделяется своей глобальной доступностью, поддержкой множества языков и валют, а также адаптивным интерфейсом, который обеспечивает удобство использования как на компьютерах, так и на мобильных устройствах. Благодаря широкому функционалу и пользовательскому комфорту, *Booking* остаётся одним из лидеров рынка онлайн-бронирования жилья.

* + 1. Приложение «Cian»

*Cian* [2] — это одна из крупнейших платформ для поиска и аренды недвижимости, которая предоставляет пользователям возможность находить подходящие варианты жилья, используя гибкие фильтры, просматривать подробные описания объектов и напрямую связываться с арендодателями. Пример интерфейса данного сервиса представлен на рисунке 1.2.

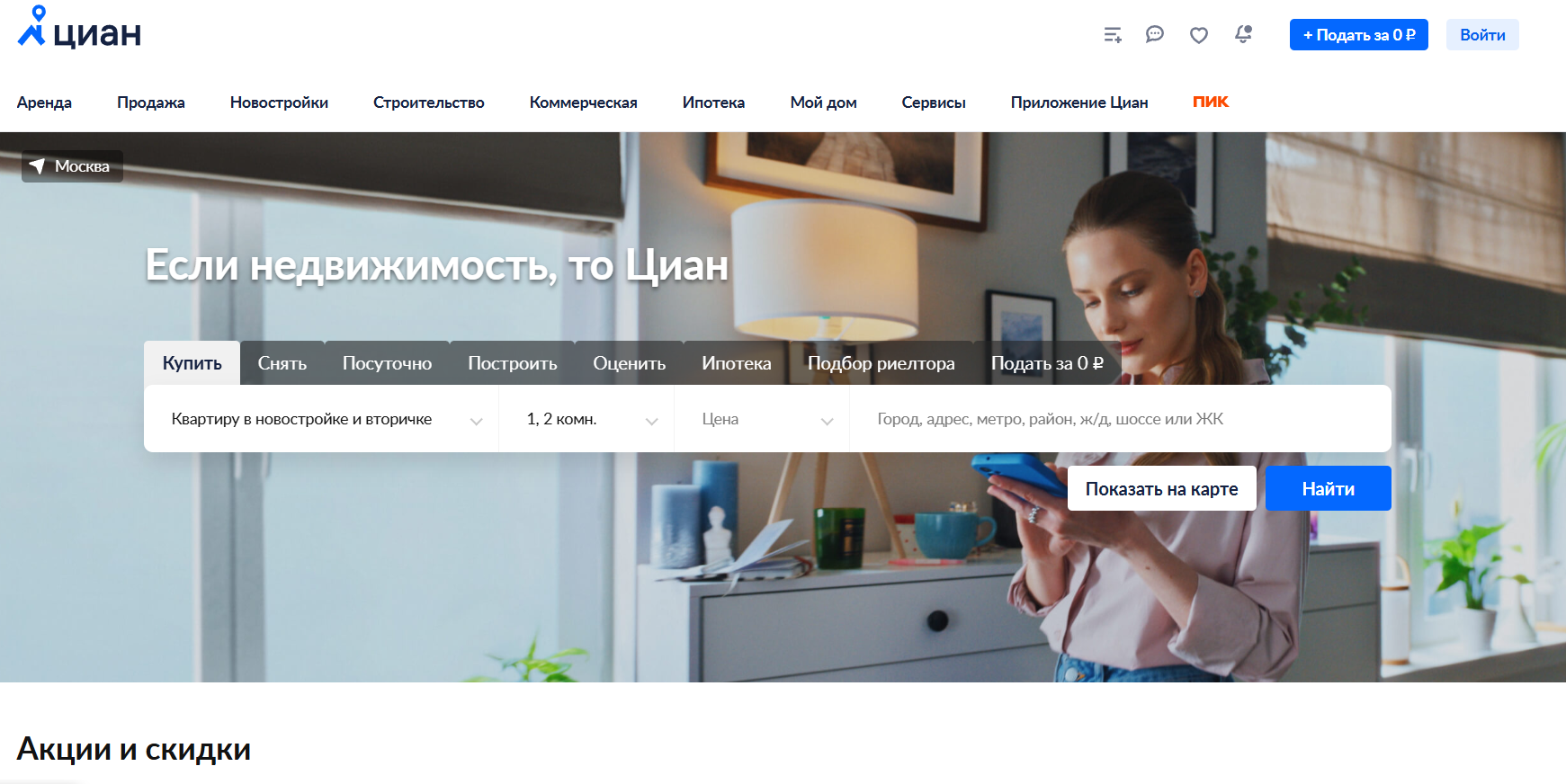


Рисунок 1.2 – Интернет-ресурс «Cian»

Ключевой функцией *Cian* является поиск недвижимости. Пользователи могут находить объекты, применяя разнообразные фильтры: по цене, району, типу жилья, площади, количеству комнат, сроку аренды и другим параметрам. Также платформа предоставляет удобную карту, где объекты отображаются с указанием точного местоположения, что упрощает выбор подходящего варианта.

Для удобства пользователей *Cian* предлагает возможность создавать личные кабинеты, где можно сохранять интересующие объявления в «Избранное», управлять своими объявлениями (для арендодателей) и отслеживать статус откликов.

Для арендодателей и продавцов недвижимости *Cian* предоставляет функционал для размещения и управления объявлениями. Владельцы могут добавлять фотографии, описания, указывать стоимость и условия аренды, а также редактировать или удалять объекты. Платформа предоставляет статистику для каждого объявления, включая количество просмотров и откликов, что помогает анализировать спрос и корректировать предложения.

С технической стороны *Cian* использует современные технологии для работы с большими объёмами данных, что обеспечивает быструю загрузку страниц, точный поиск и стабильную работу сервиса. Платформа поддерживает интеграцию с внешними системами, такими как геолокационные сервисы и платёжные системы, что делает её удобной как для арендаторов, так и для арендодателей.

Удобный и интуитивно понятный интерфейс *Cian* адаптирован для работы как на компьютерах, так и на мобильных устройствах, благодаря чему пользователи могут быстро находить и просматривать предложения в любое время. Благодаря обширной базе данных объектов, гибкости поиска и дополнительным функциям, *Cian* остаётся одним из лидеров среди платформ для аренды и покупки недвижимости.

Значительная часть объявлений на *Cian* размещается риелторскими агентствами, что иногда затрудняет поиск предложений от частных лиц. Пользователи, предпочитающие работать напрямую с владельцами, могут столкнуться с необходимостью фильтровать множество агентских объявлений.

* + 1. Приложение «Sutochno»

*Sutochno* [3] — это онлайн-платформа, специализирующаяся на краткосрочной аренде жилья. Она предоставляет пользователям удобные инструменты для поиска и бронирования квартир, домов, комнат и других объектов недвижимости по всей стране. Пример интерфейса данного сервиса представлен на рисунке 1.3.

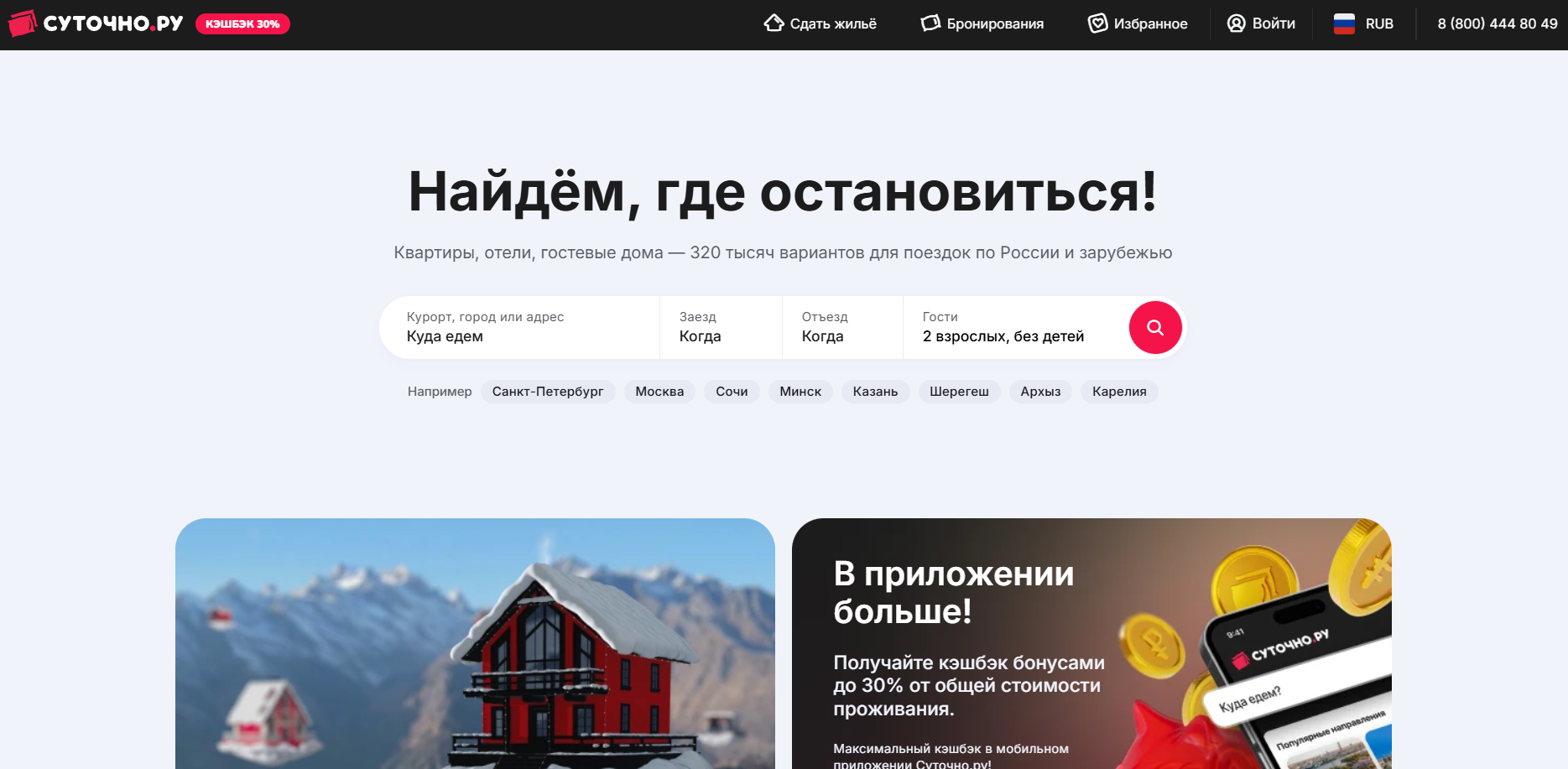


Рисунок 1.3 – Интернет-ресурс «Sutochno»

Основной функционал *Sutochno* сосредоточен на быстром поиске и аренде жилья на сутки или более короткие сроки. Пользователи могут искать объекты, применяя фильтры по цене, дате, количеству комнат, типу жилья, местоположению и другим параметрам. Система также предлагает карту для визуального поиска, на которой объекты отмечены с указанием стоимости аренды.

Платформа позволяет пользователям регистрировать личные кабинеты, где можно управлять бронированиями, сохранять интересующие объекты в «Избранное» и оставлять отзывы после завершения проживания. Пользователи могут общаться с арендодателями через встроенный чат.

Для арендодателей *Sutochno* предоставляет возможность размещать и редактировать объявления. Владельцы могут добавлять описания, фотографии, указывать правила проживания, условия отмены бронирования и доступные даты. Платформа поддерживает аналитические инструменты, позволяющие владельцам отслеживать популярность их объектов: количество просмотров и бронирований.

*Sutochno* выделяется своей ориентацией на рынок, предлагая широкий выбор объектов в крупных городах и популярных туристических направлениях. Благодаря функционалу, направленному на удобство и прозрачность для всех участников, платформа остаётся востребованным инструментом как для арендаторов, так и для арендодателей, обеспечивая комфортное взаимодействие и безопасные сделки.

* 1. Постановка задачи

Цель проекта — разработка современного веб-приложения, предоставляющего удобные инструменты для поиска, бронирования и управления недвижимостью.

Должны быть выполнены следующие требования:

* обеспечивать возможность регистрации и авторизации;
* поддерживать роли: администратор, владелец, пользователь;
* поиск, фильтрация и сортировку жилья;
* возможность управлять критериями и категориями жилья;
* возможность управлять пользователями, владельцами и их доступом;
* возможность управлять жильём в системе;
* возможность управлять статусами бронирования;
* поддержка чата между пользователем и владельцем;
* возможность просматривать информацию о жилье;
* возможность бронировать интересующее его жильё;
* возможность оставлять отзывы с оцениванием после сделки.
  1. Выводы по разделу

В данном разделе был проведён обзор аналогичных веб-приложений, таких как *Booking*, *Cian* и *Sutochno*, что позволило выявить ключевые функции, которые должны быть реализованы в нашем приложении. Эти платформы предоставляют пользователям широкий спектр возможностей: от регистрации и авторизации до поиска и бронирования жилья с использованием удобных фильтров, а также управления объектами недвижимости для владельцев.

Особое внимание уделено функционалу для различных ролей пользователей. Для владельцев объектов аренды важными элементами являются инструменты для добавления, редактирования и удаления объектов, управления бронированиями, а также ведения переписки с арендаторами. Пользователям необходимы функции просмотра и поиска жилья, бронирования, отслеживания статуса бронирований, а также возможности оставлять отзывы.

Также определены требования к удобному интерфейсу, который должен быть интуитивно понятным и обеспечивать эффективное взаимодействие между владельцами недвижимости, арендаторами и администраторами системы.

Пользовательские роли: «Администратор» (администрирование учетных записей, модерирование контента); «Владелец» (создает и поддерживает жилье, может создавать бронирования на определенные даты, отклонять или одобрять бронирования «Пользователей» на его жилье, поддерживает чат с «Пользователем»); «Пользователь» (создает бронирования на определенные даты, оставляет отзывы под жильем, поддержка чата с «Владельцем»); «Гость» (просматривает каталог жилья, выполняет поиск жилья по названию, сортирует и фильтрует жилье в каталоге, просматривает информацию о жилье).

1. Проектирование программного средства

2

Ф.И.О

Подпись

Дата

Лист

1

БГТУ 02.00.ПЗ

Разраб.

*Яковлева Е.И.*

Пров.

Кантарович В.С.

Н. контр.

Утв.

Смелов В.В.

2 Проектирование

веб-приложения

Лит.

Листов

18

*74218051, 2023*

У

Николайчук А.Н.

* 1. Разработка функциональных требований и вариантов использования

Приложение должно предоставлять функционал для регистрации и авторизации, позволяя пользователям идентифицировать себя и получать доступ к возможностям в зависимости от их роли: администратор, владелец недвижимости и обычный пользователь.

Администратор получит возможность управлять системой, включая редактирование категорий и критериев для поиска жилья, управление пользователями и их доступом, а также просмотр и анализ статистики по всем объектам и бронированиям.

Владелец недвижимости сможет добавлять, редактировать и удалять свои объекты аренды, управлять статусами бронирований, а также просматривать подробную статистику по своим объектам, включая количество просмотров, бронирований и отзывы. Для удобства взаимодействия с пользователями владельцу будет доступен встроенный чат, через который можно уточнять детали бронирования и отвечать на вопросы.

Пользователь сможет искать жильё, используя удобные фильтры и сортировку, бронировать интересующие варианты, просматривать статус своих бронирований и историю операций. Для улучшения качества сервиса пользователь сможет оставлять отзывы и оценки после завершения аренды, а также управлять своими личными данными через настройки профиля.

На рисунке 2.1 представлена диаграмма вариантов использования.

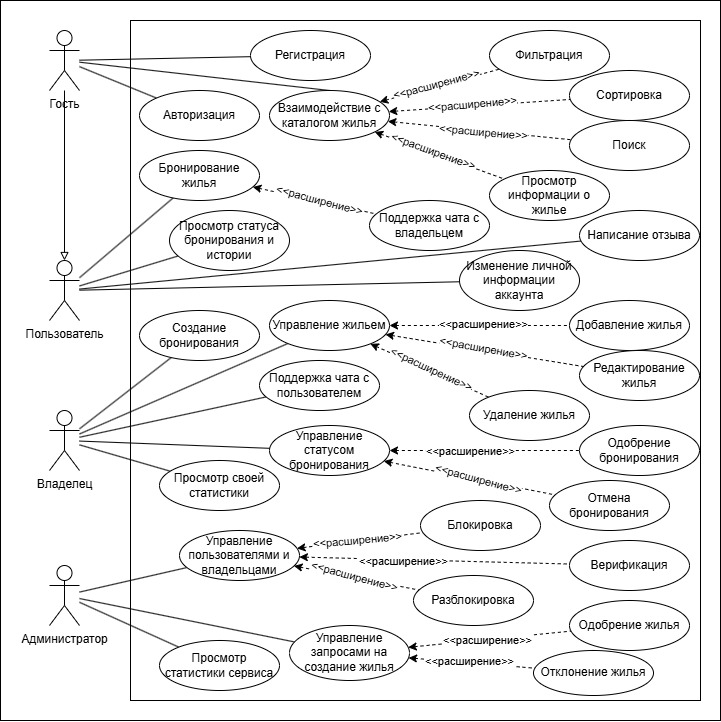


Рисунок 2.1 – Диаграмма вариантов использования

Диаграмма *UML* вариантов использования позволяет понять, что доступно каждой роли, доступной в данном веб-приложении.

* 1. Описание средств разработки

При разработке приложения необходимо выбрать языки программирования, инструменты и технологии, подходящие для реализации клиентской и серверной частей приложения. Правильный выбор средств реализации может значительно упростить и ускорить процесс разработки и реализации проекта, а также снизить риски возникновения проблем в работе готовой системы. Неправильный выбор средств может привести к проблемам в процессе разработки и реализации проекта, а также к неполадкам в работе готовой системы. В данном разделе будут рассмотрены различные библиотеки, технологии и подходы, которые будут использованы в разработке приложения.

* + 1. Visual Studio Code

*Visual Studio Code* (*VS Code*) — это популярная интегрированная среда разработки, которая завоевала доверие миллионов разработчиков по всему миру. Она поддерживает широкий спектр языков программирования, включая *JavaScript*, *TypeScript*, *Python*, *C++* и многие другие, а также такие технологии, как *React* и Node.js. Благодаря своей универсальности, *VS Code* подходит как для начинающих, так и для опытных разработчиков, работающих над проектами любой сложности.

Одной из ключевых особенностей *VS Code* является его легковесность и высокая производительность. Среда быстро запускается и работает даже на не самых мощных компьютерах, что делает её идеальным выбором для разработчиков, ценящих скорость и эффективность. При этом *VS Code* обладает гибкостью, позволяя пользователям настраивать интерфейс, горячие клавиши и функциональность под свои потребности.

*VS Code* предоставляет мощный набор встроенных инструментов для написания, отладки и тестирования кода. Встроенный терминал, поддержка систем контроля версий (например, *Git*) и интеллектуальная подсказка кода значительно упрощают процесс разработки. Кроме того, среда поддерживает отладку для множества языков, что позволяет быстро находить и исправлять ошибки в коде.

Особое внимание стоит уделить экосистеме расширений *VS Code*. В магазине расширений доступны тысячи плагинов, которые добавляют поддержку новых языков, фреймворков, баз данных и инструментов. Например, расширения для работы с *Docker*, *Kubernetes* или базами данных, такими как *PostgreSQL* и *MongoDB*, делают *VS Code* универсальным инструментом для всех этапов разработки — от написания кода до его тестирования и деплоя.

В заключение, *Visual Studio Code* — это мощная, гибкая и легковесная среда разработки, которая помогает разработчикам эффективно справляться с задачами любой сложности. Благодаря поддержке множества языков, встроенным инструментам и богатой экосистеме расширений, *VS Code* остаётся одним из лучших выборов для современных разработчиков.

* + 1. Платформа Node.js

*Node.js* — это серверная среда выполнения, которая позволяет запускать *JavaScript*-код за пределами браузера. Основанная на движке *V8*, используемом в *Google Chrome*, *Node.js* обеспечивает высокую производительность и открывает возможности для создания полноценных серверных приложений. Эта технология изменила подход к разработке, позволив использовать *JavaScript* как для клиентской, так и для серверной части приложений.

Одной из главных особенностей *Node.js* является её асинхронная, неблокирующая модель ввода-вывода, основанная на событиях. Такая архитектура делает *Node.js* идеальным выбором для создания масштабируемых веб-приложений, способных обрабатывать большое количество одновременных запросов. Это особенно важно для приложений, требующих высокой производительности, таких как чат-системы, онлайн-игры или платформы для обмена сообщениями в реальном времени.

*Node.js* также славится своей гибкостью и обширной экосистемой. Благодаря *npm* (*Node Package Manager*), разработчики имеют доступ к тысячам библиотек и модулей, которые значительно ускоряют процесс разработки. От фреймворков, таких как *Express*, до инструментов для работы с базами данных и тестирования — *npm* предоставляет всё необходимое для создания современных приложений.

В заключение, *Node.js* — это мощная и универсальная платформа, которая идеально подходит для разработки высокопроизводительных и масштабируемых веб-приложений. Благодаря поддержке асинхронной модели, богатой экосистеме npm и возможности использовать *JavaScript* на сервере, *Node.js* остаётся одним из ключевых инструментов в арсенале современных разработчиков.

* + 1. Язык программирования TypeScript

*TypeScript* — это язык программирования, представляющий собой надстройку над *JavaScript*, которая вводит статическую типизацию. Разработанный компанией *Microsoft*, *TypeScript* призван повысить предсказуемость и безопасность разработки, позволяя выявлять потенциальные ошибки на этапе компиляции. Это делает его мощным инструментом для создания надежных и масштабируемых приложений.

Основное преимущество *TypeScript* — это строгая проверка типов, которая помогает разработчикам обнаруживать ошибки еще до выполнения кода. Благодаря статической типизации код становится более читаемым и структурированным, что особенно важно для крупных проектов. *TypeScript* облегчает поддержку и масштабирование приложений, минимизируя вероятность ошибок, связанных с динамической природой *JavaScript*.

*TypeScript* полностью совместим с *JavaScript*, что позволяет использовать все его стандартные возможности. При этом *TypeScript* добавляет дополнительные функции, такие как интерфейсы, перечисления и пользовательские типы данных. Эти инструменты способствуют созданию более структурированного и понятного кода, упрощая совместную работу в командах и долгосрочное сопровождение проектов.

В заключение, *TypeScript* — это мощный язык, который расширяет возможности *JavaScript*, делая разработку более безопасной, удобной и эффективной. Благодаря поддержке статической типизации и дополнительных функций, *TypeScript* стал стандартом для разработки сложных и масштабируемых приложений, особенно в крупных командах.

* + 1. Библиотека React

*React* — это популярная *JavaScript*-библиотека, разработанная компанией *Facebook*, предназначенная для создания современных и динамичных пользовательских интерфейсов. Благодаря своей гибкости и компонентному подходу, *React* стал одним из ключевых инструментов в арсенале веб-разработчиков, особенно для построения одностраничных приложений (*SPA*).

Основой *React* является компонентный подход, при котором интерфейс разбивается на независимые, переиспользуемые компоненты. Каждый компонент управляет своим состоянием и поведением, что упрощает разработку, тестирование и поддержку кода. Такой подход особенно ценен при создании сложных интерфейсов, где требуется высокая степень модульности и масштабируемости.

Одной из главных особенностей *React* является использование виртуального *DOM*. Эта технология позволяет библиотеке минимизировать прямые манипуляции с реальным *DOM*, обновляя только те элементы интерфейса, которые действительно изменились. В результате *React* обеспечивает высокую производительность и отзывчивость приложений, улучшая пользовательский опыт даже в условиях интенсивных обновлений интерфейса.

*React* также отличается простотой интеграции с другими инструментами и библиотеками, что делает его универсальным решением для веб-разработки. Благодаря активной поддержке сообщества и богатой экосистеме, *React* предоставляет разработчикам всё необходимое для создания современных, быстрых и интерактивных веб-приложений.

В заключение, *React* — это мощная и гибкая библиотека, которая упрощает создание динамичных пользовательских интерфейсов. Компонентный подход, виртуальный *DOM* и широкая поддержка сообщества делают *React* идеальным выбором для разработки современных веб-приложений, от простых сайтов до сложных одностраничных приложений.

* + 1. PostgreSQL

*PostgreSQL* — это объектно-реляционная система управления базами данных (СУБД) с открытым исходным кодом, которая заслужила репутацию одного из самых надежных и универсальных решений для хранения и обработки данных. Благодаря своей гибкости и мощным возможностям, *PostgreSQL* широко используется в высоконагруженных веб-приложениях и корпоративных системах.

Одной из ключевых особенностей *PostgreSQL* является поддержка *ACID*-транзакций, обеспечивающих надежность и консистентность данных даже в условиях сложных операций. СУБД поддерживает сложные запросы, разнообразные типы данных, включая географические, а также предоставляет возможности для создания хранимых процедур и триггеров. Это делает *PostgreSQL* идеальным выбором для приложений, требующих высокой степени функциональности и точности.

*PostgreSQL* отличается высокой производительностью и масштабируемостью, что позволяет эффективно обрабатывать большие объемы данных. Благодаря своей архитектуре, СУБД легко справляется с нагрузками, характерными для крупных веб-приложений и аналитических систем. Кроме того, *PostgreSQL* поддерживает расширяемость, позволяя разработчикам добавлять собственные функции и модули для специфических задач.

Гибкость *PostgreSQL* делает его подходящим как для небольших стартапов, так и для крупных корпоративных решений. Активное сообщество и обширная документация обеспечивают поддержку и постоянное развитие системы, что делает её надежным выбором для современных приложений.

В заключение, *PostgreSQL* — это мощная, масштабируемая и универсальная СУБД, которая сочетает в себе надежность, производительность и гибкость. Поддержка сложных запросов, *ACID*-транзакций и расширяемость делают *PostgreSQL* незаменимым инструментом для разработки современных приложений, работающих с большими объемами данных.

* 1. Проектирование базы данных

Диаграмма базы данных таблиц (*Database Table Diagram*) – это визуальное представление структуры базы данных и отношений между таблицами, которые хранятся в этой базе данных. Диаграмма представлена на рисунке 2.2.

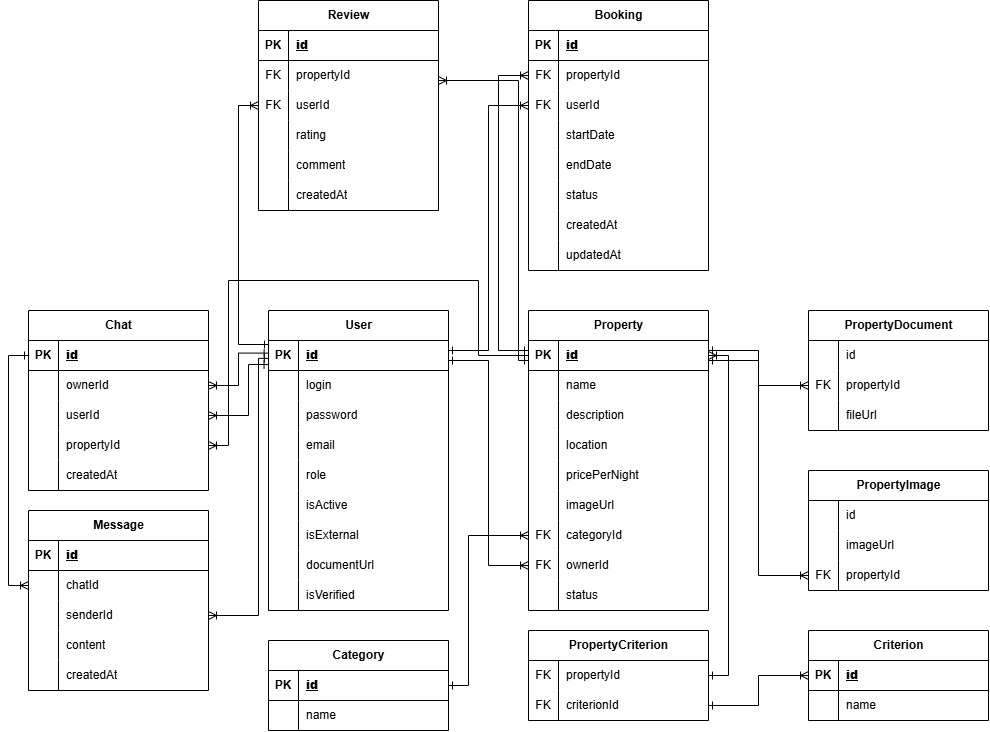


Рисунок 2.2 – Логическая схема базы данных

В этой базе данных используется сложная структура для управления пользователями, недвижимостью, бронированиями и отзывами. Модели и таблицы в базе данных связаны между собой с помощью различных связей и перечислений, что позволяет эффективно хранить и извлекать данные.

* + 1. Таблица User

Таблица 2.1 – Структура таблицы «User»

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Тип | Ограничения | Описание |
| id | INTEGER | Primary key | Идентификатор пользователя |
| login | TEXT | Not null | Логин пользователя |
| password | TEXT | Not null | Пароль пользователя |
| email | TEXT | Not null | Адрес электронной почты |
| role | TEXT | Not null | Роль пользователя |
| isActive | BOOLEAN | Not null | Статус пользователя |
| isExternal | BOOLEAN | Not null | Признак того, что пользователь создан извне |
| documentUrl | TEXT | Nullable | Прикреплённый документ пользователя |
| isVerified | BOOLEAN | Not null | Статус верификации документа |
| created\_at | DATETIME | Not null | Дата и время создания пользователя |
| updated\_at | DATETIME | Not null | Дата и время последнего обновления |

Данная таблица содержит существующих пользователей. Столбец isActive отвечает за статус пользователя, он может быть заблокирован или нет. Столбец isExternal является признаком того, что пользователь был создан владельцем жилья. Столбец role может принимать значение ADMIN, OWNER и USER*.*

* + 1. Таблица Property

Таблица 2.2 – Структура таблицы «Property»

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Тип | Ограничения | Описание |
| id | INTEGER | Primary key | Идентификатор жилья |
| name | TEXT | Not null | Название жилья |
| description | TEXT | Not null | Описание жилья |
| location | TEXT | Not null | Адрес жилья |
| pricePerNight | INTEGER | Not null | Стоимость аренды за ночь |
| ownerId | INTEGER | Foreign key | Владелец жилья |
| categoryId | INTEGER | Foreign key | Категория жилья |
| status | TEXT | Not null | Статус жилья |

Данная таблица содержит информацию о жилье. Столбец ownerId служит для связи с таблицей User многие к одному. Столбец categoryId служит для связи с таблицей Category многие к одному. Столбец status может принимать значения PENDING, APPROVED и REJECTED, по умолчанию столбец имеет значение PENDING, статус меняет администратор сервиса.

* + 1. Таблица PropertyDocument

Таблица 2.3 – Структура таблицы «PropertyDocument»

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Тип | Ограничения | Описание |
| id | INTEGER | Primary key | Идентификатор документа |
| propertyId | INTEGER | Foreign key | Идентификатор жилья |
| fileUrl | TEXT | Not null | Загруженный файл |

Данная таблица содержит документы для жилья. Столбец propertyId служит для с связи с таблицей Property многие к одному.

* + 1. Таблица Review

Таблица 2.4 – Структура таблицы «Review»

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Тип | Ограничения | Описание |
| id | INTEGER | Primary key | Идентификатор отзыва |
| propertyId | INTEGER | Foreign key | Идентификатор жилья |
| userId | INTEGER | Foreign key | Идентификатор пользователя |
| rating | INTEGER | Not null | Оценка объект |
| comment | TEXT | Nullable | Текстовый комментарий к отзыву |
| createdAt | DATETIME | Not null | Дата и время создания отзыва |

Данная таблица хранит отзывы жилья. Столбец propertyId служит для с связи с таблицей Property многие к одному. Столбец userId служит для с связи с таблицей User многие к одному.

* + 1. Таблица Booking

Таблица 2.5 – Структура таблицы «Booking»

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Тип | Ограничения | Описание |
| id | INTEGER | Primary key | Идентификатор бронирования |
| propertyId | INTEGER | Foreign key | Идентификатор жилья |
| userId | INTEGER | Foreign key | Идентификатор пользователя |
| startDate | DATETIME | Not null | Дата начала бронирования |
| endDate | DATETIME | Not null | Дата окончания бронирования |
| status | TEXT | Not null | Статус бронирования |
| createdAt | DATETIME | Not null | Дата и время создания бронирования |
| updatedAt | DATETIME | Not null | Дата и время обновления записи |

Данная таблица хранит информацию о бронировании жилья. Столбец propertyId служит для с связи с таблицей Property многие к одному. Столбец userId служит для с связи с таблицей User многие к одному. Столбец status может принимать значения PENDING, CONFIRMED и CANCELLED, по умолчанию столбец имеет значение PENDING, статус меняет владелец жилья.

* + 1. Таблица Category

Таблица 2.6 – Структура таблицы «Booking»

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Тип | Ограничения | Описание |
| id | INTEGER | Primary key | Идентификатор категории |
| name | TEXT | Not null | Название категории |

Данная таблица содержит категории жилья.

* + 1. Таблица Criterion

Таблица 2.7 – Структура таблицы «Criterion»

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Тип | Ограничения | Описание |
| id | INTEGER | Primary key | Идентификатор критерия |
| name | TEXT | Not null | Название критерия |

Данная таблица содержит критерии жилья.

* + 1. Таблица ****PropertyCriterion****

Таблица 2.8 – Структура таблицы «**PropertyCriterion**»

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Тип | Ограничения | Описание |
| propertyId | INTEGER | Foreign key | Идентификатор жилья |
| criterionId | INTEGER | Foreign key | Идентификатор критерия |

Данная таблица является промежуточной между таблицами Property и Criterion.

* + 1. Таблица Chat

Таблица 2.9 – Структура таблицы «Chat»

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Тип | Ограничения | Описание |
| id | INTEGER | Primary key | Идентификатор чата |
| ownerId | INTEGER | Foreign key | Идентификатор владельца жилья |
| userId | INTEGER | Foreign key | Идентификатор пользователя |
| propertyId | INTEGER | Foreign key | Идентификатор жилья |
| createdAt | DATETIME | Not null | Дата и время создания чата |

Данная таблица содержит информацию о чатах. Столбец ownerId служит для с связи с таблицей User многие к одному. Столбец userId служит для с связи с таблицей User многие к одному. Столбец propertyId служит для с связи с таблицей Property многие к одному.

* + 1. Таблица Message

Таблица 2.10 – Структура таблицы «Message»

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Тип | Ограничения | Описание |
| id | INTEGER | Primary key | Идентификатор сообщения |
| chatId | INTEGER | Foreign key | Идентификатор чата |
| senderId | INTEGER | Foreign key | Идентификатор отправителя |
| content | TEXT | Not null | Текстовое содержимое сообщения |
| createdAt | DATETIME | Not null | Дата и время отправки сообщения |

Данная таблица содержит информацию о сообщениях. Столбец chatId служит для с связи с таблицей Chat многие к одному. Столбец senderId служит для с связи с таблицей User многие к одному.

* + 1. Таблица PropertyImage

Таблица 2.11 – Структура таблицы «PropertyImage»

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Тип | Ограничения | Описание |
| id | INTEGER | Primary key | Идентификатор изображения |
| imageUrl | TEXT | Not null | Ссылка на изображение жилья |
| propertyId | INTEGER | Foreign key | Идентификатор жилья |

Данная таблица содержит информацию о чатах. Столбец propertyId служит для с связи с таблицей Property многие к одному.

* 1. Проектирование основных алгоритмов

В данном разделе будут описаны некоторые алгоритмы работы приложения, которые соответствуют целям дипломного проекта.

Для реализации каждого алгоритма были созданы соответствующие блок-схемы. Особенностью блок-схем является то, что они представляют собой графическую интерпретацию, позволяя сделать работу алгоритма более наглядной и упростить ее понимание. Каждый блок представляет собой некоторое действие или событие, при совершении которого начинается переход к другому действию.

Всего в ходе выполнения дипломного проекта было создано два алгоритма, каждый из которых будет описан ниже в соответствующем подразделе.

* + 1. Алгоритм добавления жилья

Для того чтобы создать жилье в сервисе нужно проделать определенные действия, которые формируют определенный алгоритм.

Данный алгоритм продемонстрирован далее на рисунке 2.3.

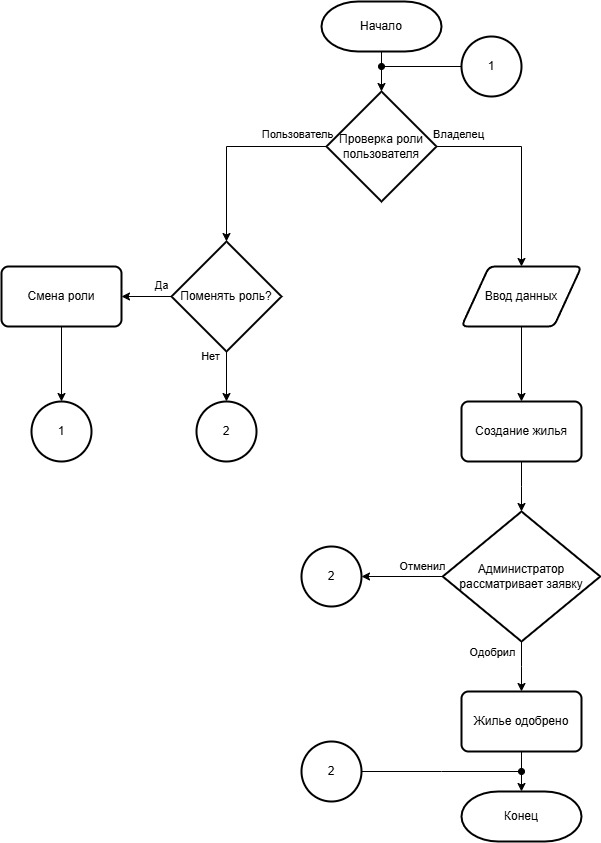


Рисунок 2.3 – Алгоритм добавления жилья

Для того чтобы создать жилье у пользователя должна быть роль владельца, таким образом сначала сервис проверяет роль пользователя. Если роль обычного пользователя, то нужно поменять роль на владельца. После смены роли появится возможность создания жилья, где нужно будет ввести определенные данные. После их ввода жилье добавится в каталоге самого владельца, но для того, чтобы оно стало доступным для всех пользователей сервиса, оно должно пройти проверку администратором. Администратор рассматривает документы, приложенные к жилью во время его создания, и принимает решение: одобрить жилье или отменить. В случае отмены жилье так и не будет показываться всем пользователям жилья, а только самому владельцу. Если же администратор жилье одобрил, то можно считать, что создание жилья в сервисе успешно завершено.

* + 1. Алгоритм бронирования жилья

При попытке забронировать жилье на сервисе происходят определенные процессы.

Данный алгоритм продемонстрирован ниже на рисунке 2.4.

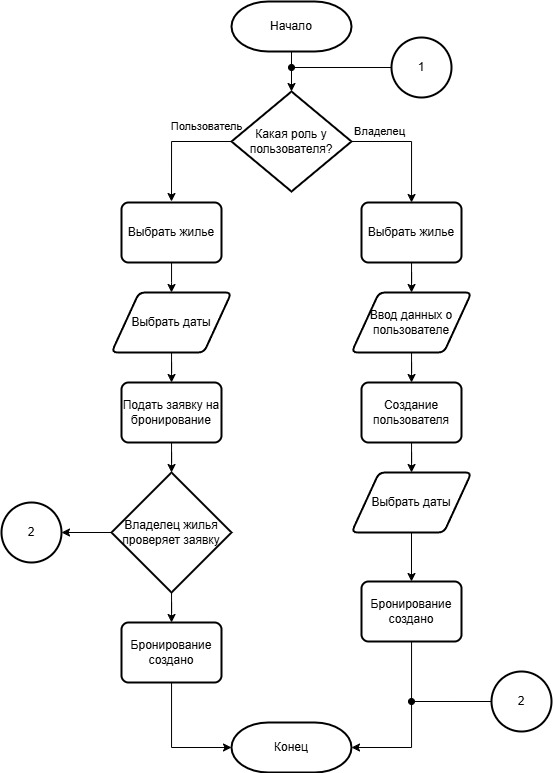


Рисунок 2.4 – Алгоритм бронирования жилья

Итак, при создании жилья сначала проверяется роль пользователя. В случае, если роль обычный пользователь, то сначала нужно выбрать жилье, которое вы хотите забронировать. Затем выбрать даты и подать заявку на бронирование. После чего уже владелец принимает решение одобрить или отклонить заявку на бронирование, с случае одобрения, то бронирование будет создано. Если же первоначально роль владелец, то сначала нужно выбрать жилье, затем ввести данные о пользователе, тем самым создаться пользователь, а затем выбрать даты для бронирования. Вследствие чего будет создано бронирование. Таким образом сервис предоставляет возможность создавать бронирования как для обычного пользователя, так и для владельца жилья.

* 1. Выводы по разделу

В разделе проектирования была построена и описана диаграмма вариантов использования. Эта диаграмма отражает основные функциональные требования к системе и демонстрирует типичные взаимодействия между пользователями и программным обеспечением.

В рамках данного раздела был произведён выбор технологий, языков программирования и инструментов, необходимых для реализации дипломного проекта. Выбор обоснован с точки зрения надёжности, производительности, расширяемости и соответствия целям проекта.

Для реализации проекта были использованы следующие технологии и инструменты:

* *Node.js 20* — как серверная платформа для обработки запросов и управления бизнес-логикой;
* *TypeScript 5* — как основной язык программирования с поддержкой типизации и улучшенной читаемостью кода;
* *Express.js 4* — как базовый серверный фреймворк для построения *REST API*;
* *React.js 18* — как библиотека для построения пользовательского интерфейса;
* *PostgreSQL 16* — как система управления базами данных, обеспечивающая реляционное хранение информации;
* *Prisma ORM* — как инструмент для работы с базой данных, предоставляющий удобный слой абстракции над *SQL* и поддержку миграций.

В проекте была выбрана реляционная СУБД *PostgreSQL*. На её основе спроектирована и реализована структура базы данных, включающая 11 таблиц, каждая из которых отвечает за хранение определённых сущностей и связей между ними. Были реализованы ограничения целостности данных, а также связи между таблицами, обеспечивающие корректную обработку и валидацию информации при взаимодействии с приложением.

Также в рамках проектирования были разработаны два ключевых алгоритма, отражающие основную бизнес-логику приложения:

* Алгоритм добавления жилья — описывает последовательность действий при создании нового объекта недвижимости, включая валидацию данных, загрузку изображений и прикрепление документов;
* Алгоритм бронирования жилья — реализует процесс создания бронирования пользователем, включая проверку доступности жилья на выбранные даты и сохранение информации о брони в базе данных.

Каждый из этих алгоритмов был подробно описан и визуализирован в виде блок-схем, что способствует их последующей реализации и тестированию.

1. Реализация программного средства

Ф.И.О

Подпись

Дата

Лист

1

БГТУ 03.00.ПЗ

Разраб.

*Яковлева Е.И.*

Пров.

Кантарович В.С.

Н. контр.

Утв.

Смелов В.В.

3 Реализация программного средства

Лит.

Листов

12

*74218051, 2023*

У

Николайчук А.Н.

Главной задачей данного дипломного проекта является создание веб-сервиса для аренды жилья. На основе анализа существующих аналогичных сервисов были сформулированы задачи: разработка программных компонентов для работы пользователей, обеспечение удобного и интуитивного интерфейса, а также соблюдение всех современных требований к выбору технологий и средств реализации.

* 1. Общая архитектура приложения

На рисунке 3.1 представлена схема архитектуры всего приложения в целом. Как можно увидеть, разрабатываемое веб-приложение состоит из трех частей: база данных, серверная часть и клиентская часть.

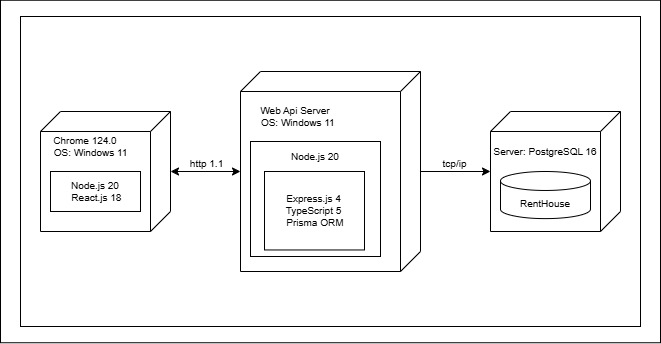


Рисунок 3.1 – Архитектура приложения

База данных приложения *RentHouse* размещается на сервере *PostgreSQL 16*. Серверная часть приложения написана на языке программирования *TypeScript* с использованием среды выполнения *Node.js 20*, фреймворка *Express.js 4* и *ORM Prisma*. Для запуска серверной части используется операционная система *Windows 11*. Клиентская часть запускается на операционной системе *Windows 11* в браузере *Google Chrome* и написана с использованием библиотеки *React.js 18*, которая работает на основе *Node.js*.

* 1. Разработка серверной части

Серверная часть приложения разработана с использованием платформы *Node.js* и языка *TypeScript*. Основной файл проекта — index.ts, который является точкой входа в приложение. В нем инициализируются все необходимые модули и настраиваются маршруты. Взаимодействие с клиентом происходит через различные маршруты, которые в свою очередь направляют запросы на соответствующие контроллеры для обработки.

Для работы с базой данных используется *Prisma*, который служит эффективным *ORM*-инструментом для взаимодействия с *PostgreSQL*. *Prisma* значительно упрощает выполнение запросов и манипуляцию данными, позволяя работать с базой через удобный *API*. На рисунке 3.2 будет представлена схема структуры данного проекта.

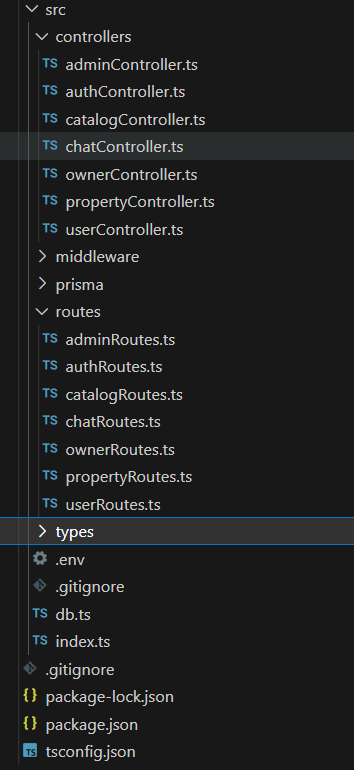


Рисунок 3.2 – Структура проекта

В приложении основная роль в обработке запросов принадлежит контроллерам и маршрутам. Контроллеры отвечают за обработку входящих *HTTP*-запросов, а маршруты направляют эти запросы на соответствующие контроллеры для выполнения нужных действий, таких как создание, получение, обновление или удаление данных. Контроллеры выполняют логику обработки запросов и вызывают соответствующие функции для работы с данными, что позволяет организовать код более структурированно.

Каждый контроллер связан с определенным функционалом приложения. Например, в проекте контроллер catalogController занимается созданием жилья с загрузкой изображений, а другие контроллеры, такие как authRoutes, userRoutes, propertyRoutes и adminRoutes, обрабатывают различные функциональные блоки, такие как авторизация, работа с пользователями, управление недвижимостью и администрирование.

Маршруты (routes) служат связующим звеном между входящими запросами и контроллерами. В отличие от контроллеров, маршруты не содержат бизнес-логику, а просто перенаправляют запросы на соответствующие функции в контроллерах. В коде маршруты для авторизации, пользователей, каталога, недвижимости и администрирования подключаются в файле index.ts, который является точкой входа в приложение.

Файл index.ts является главным файлом сервера, где настраиваются маршруты и запускается сервер. В этом файле создается экземпляр приложения *Express*, подключаются все маршруты через app.use(), а также настраиваются middleware для обработки запросов, таких как CORS, парсинг JSON и загрузка файлов через multer. В конце происходит запуск сервера и подключение к базе данных через *Prisma*. Таким образом, index.ts является основным файлом, где происходит настройка всего приложения, подключение всех необходимых компонентов и их взаимодействие.

Структура проекта организована таким образом, чтобы контроллеры и маршруты могли легко взаимодействовать друг с другом, при этом каждая часть приложения отвечает только за свою область ответственности. Файл index.ts представлен в листинге 3.1.

const app = express();

const PORT = process.env.PORT || 3000;

app.use(cors({

  origin: 'http://localhost:3001',

}));

app.use(express.json());

const storage = multer.diskStorage({

  destination: (req, file, cb) => {

    cb(null, 'images/');

  },

  filename: (req, file, cb) => {

    const uniqueSuffix = Date.now() + '-' + Math.round(Math.random() \* 1e9);

    cb(null, file.fieldname + '-' + uniqueSuffix + path.extname(file.originalname));

  },

});

const upload = multer({ storage });

app.post('/housing', upload.single('image'), createHousing);

app.use('/images', express.static(path.join(\_\_dirname, '../images')));

app.use('/auth', authRoutes);

app.use('/users', userRoutes);

app.use('/catalog', catalogRoutes);

app.use('/property', propertyRoutes);

app.use('/admin', adminRoutes);

app.listen(PORT, async () => {

  await prisma.$connect();

  console.log(`Сервер запущен на http://localhost:${PORT}`);

});

Листинг 3.1 – Файл index.ts

Файл tsconfig.json в проекте на *TypeScript* играет ключевую роль в настройке компилятора и определении того, как будет происходить преобразование *TypeScript* в *JavaScript*. В вашем случае файл настроен для проекта, использующего *Node.js* и *Express*, и включает ряд параметров, которые важны для корректной работы приложения. Данный файл представлен в листинге 3.2.

{

  "compilerOptions": {

    "target": "ES6",

    "module": "commonjs",

    "esModuleInterop": true,

    "strict": true,

    "skipLibCheck": true,

    "forceConsistentCasingInFileNames": true,

    "outDir": "./dist",

    "typeRoots": [

      "./node\_modules/@types",

      "./src/types"],

    "types": ["node", "express"]

  },

  "include": [

    "src/\*\*/\*.ts",

    "src/\*\*/\*.tsx"

  ],

  "exclude": [

    "node\_modules"

  ]

}

Листинг 3.2 – Файл tsconfig.json

target: устанавливает версию *ECMAScript*, в которую будет компилироваться *TypeScript*. В вашем случае это *ES6*, что означает, что компилятор будет генерировать код, соответствующий стандарту *ECMAScript 2015* (*ES6*), включая поддержку классов, стрелочных функций, промисов и других функций этого стандарта.

module: указывает, какой модульный формат будет использоваться. В вашем случае это commonjs, который является стандартом для *Node.js*. Это означает, что компилятор будет генерировать код с использованием require() и module.exports для импорта и экспорта модулей.

esModuleInterop: включает поддержку взаимодействия с модулями, экспортируемыми по стандарту ES*,* и позволяет использовать их в стиле CommonJS. Это упрощает работу с модулями, которые используют разные стандарты, и позволяет делать импорты в стиле import \* as X from 'module'.

strict: включает строгий режим, который активирует множество проверок типов, таких как noImplicitAny, strictNullChecks и другие, что помогает предотвратить потенциальные ошибки и улучшить качество кода.

skipLibCheck: Отключает проверку типов в файлах .d.ts (определения типов), что может ускорить компиляцию, особенно в больших проектах с множеством зависимостей.

forceConsistentCasingInFileNames: устанавливает требование использовать согласованный регистр в именах файлов, предотвращая проблемы с кроссплатформенной совместимостью (например, различия между *Windows* и *Linux*).

outDir: указывает, куда будет скомпилирован результат работы компилятора. В вашем случае это папка ./dist, куда будут выводиться все скомпилированные .js файлы.

typeRoots: определяет директории, в которых компилятор будет искать типы. В вашем случае компилятор будет искать типы сначала в папке node\_modules/@types (стандартные типы, установленные через *npm*), а затем в src/types (пользовательские типы, которые вы можете создать).

types: определяет, какие типы следует подключить. В вашем случае это типы для *Node.js* и *Express*, что позволяет использовать их в проекте для автодополнения и проверки типов.

include: указывает, какие файлы *TypeScript* должны быть включены в проект. В вашем случае это все файлы с расширениями .ts и .tsx в папке src, что означает, что компилятор будет обрабатывать все файлы *TypeScript* в этой директории.

exclude: определяет, какие файлы или директории должны быть исключены из процесса компиляции. В вашем случае это папка node\_modules, которая не должна быть скомпилирована, так как она уже содержит скомпилированный код и типы для всех установленных зависимостей.

В рамках данного проекта будут рассматриваться следующие основные функции:

* регистрация;
* авторизация;
* управление жильём в системе;
* бронирование жилья;
* оставление отзывов с оцениванием после сделки.
  + 1. Регистрация

В любом приложении одной из ключевых задач является правильная и безопасная регистрация и авторизация пользователей. Эти процессы должны быть спроектированы таким образом, чтобы гарантировать защиту данных пользователей, обеспечивать удобство использования и беспрепятственную интеграцию с другими частями системы. В данном разделе будет рассмотрен процесс регистрации пользователей. Для реализации регистрации в приложении используется контроллер authController, который управляет обработкой запроса на регистрацию. В запросе от клиента передаются такие данные, как email, пароль и роль пользователя, при этом роль пользователя указывается на стороне клиента. Код, реализующий логику регистрации, будет представлен в листинге 3.3.

export const register = async (req: Request, res: Response): Promise<void> => {

  const { login, password, email } = req.body;

    if (existingUser) {

      res.status(400).json({ message: 'Пользователь с таким логином или email уже существует.' });

      return;

    }

    const hashedPassword = await bcrypt.hash(password, 10);

    const newUser = await prisma.user.create({

      data: {

        login,

        password: hashedPassword,

        email,

        role: 'USER',

      },

    });

    const token = jwt.sign(

      { userId: newUser.id, role: newUser.role },

      JWT\_SECRET,

      { expiresIn: '72h' }

    );

    res.status(201).json({

      message: 'Пользователь успешно зарегистрирован',

      token,

    });

  }

};

Листинг 3.3 – Код регистрации

Код представляет собой обработчик маршрута для регистрации пользователя в приложении. Когда клиент отправляет запрос на регистрацию с данными пользователя, такими как логин, пароль и email, сервер выполняет несколько шагов для создания нового пользователя. Сначала данные из запроса извлекаются и проверяется, существует ли уже пользователь с таким логином или email в базе данных. Если такой пользователь найден, сервер возвращает ошибку, уведомляя, что логин или email уже заняты. Если же такого пользователя нет, то сервер переходит к следующему шагу — хешированию пароля. Для этого используется библиотека bcrypt, которая безопасно преобразует исходный пароль в хеш, чтобы защитить его от несанкционированного доступа. После хеширования пароля создается новый пользователь в базе данных с предоставленными данными. Роль пользователя по умолчанию устанавливается как 'USER'. Также, в этот момент создается *JWT* токен. Этот токен содержит идентификатор пользователя и его роль, а также срок действия — токен будет действителен в течение 72 часов. *JWT* токен используется для аутентификации пользователя в дальнейшем. Если процесс регистрации проходит успешно, сервер отправляет клиенту ответ, содержащий сообщение об успешной регистрации и сгенерированный токен. В случае возникновения ошибки в процессе регистрации (например, проблемы с базой данных или хешированием пароля), сервер отправляет ошибку с описанием проблемы. Этот процесс позволяет обеспечить надежную защиту данных пользователей, безопасность паролей и удобное управление сессиями с помощью токенов.

* + 1. Авторизация

Авторизация — это ключевая часть безопасности приложения, которая предоставляет пользователям доступ к ограниченным ресурсам после успешной проверки их личности. Этот процесс включает в себя подтверждение данных пользователя и создание сессии, что позволяет контролировать доступ к защищенным частям системы. В данном случае процесс авторизации осуществляется через проверку учетных данных пользователя, таких как логин и пароль, с последующей выдачей токена для дальнейшего взаимодействия с системой. Логика авторизации обеспечит безопасность и управление правами пользователей в системе. Код, реализующий процесс авторизации, будет представлен в листинге 3.4.

export const login = async (req: Request, res: Response) : Promise<void> => {

  const { login, password } = req.body;

  try {

    const user = await prisma.user.findUnique({

      where: { login },

    });

    if (!user) {

      res.status(400).json({ message: 'Неверный логин или пароль' });

      return;

    }

    const isPasswordValid = await bcrypt.compare(password, user.password);

    if (!isPasswordValid) {

      res.status(400).json({ message: 'Неверный логин или пароль' });

      return;

    }

    const token = jwt.sign({ userId: user.id, role: user.role }, JWT\_SECRET, {

      expiresIn: '72h',

    });

    res.status(200).json({ message: 'Авторизация успешна', token });

  } catch (error) {

    res.status(500).json({ message: 'Ошибка при авторизации', error });

  }

};

Листинг 3.4 – Код авторизации

Этот контроллер реализует процесс авторизации пользователя. Когда пользователь пытается войти в систему, отправляется запрос с логином и паролем. Контроллер извлекает эти данные из тела запроса и ищет пользователя в базе данных, используя логин. Если пользователь с таким логином не найден, сервер отправляет ошибку с сообщением, что логин или пароль неверные. Если пользователь найден, следующим шагом идет проверка пароля. Для этого используется метод bcrypt, который сравнивает введенный пароль с хешированным паролем, хранящимся в базе данных. Если пароли не совпадают, также возвращается ошибка с аналогичным сообщением. Если логин и пароль верны, генерируется *JWT* токен, который содержит информацию о пользователе (его *ID* и роль). Этот токен используется для авторизации и доступа к защищенным частям приложения. Токен отправляется пользователю в ответе, и с его помощью он может выполнять дальнейшие запросы в систему без необходимости вводить логин и пароль повторно. Если во время выполнения процесса авторизации возникает ошибка, например, при работе с базой данных или другим компонентом, сервер возвращает сообщение об ошибке с соответствующим кодом состояния.

* + 1. Управление жильём в системе

Управление жильем в системе — это процесс, который включает в себя создание, изменение и удаление информации о жилье, которое доступно на платформе для бронирования. В данной части приложения пользователи (в частности, владельцы жилья) могут добавлять новое жилье в систему, редактировать его характеристики или удалять, если оно больше не доступно. Управление жильем помогает поддерживать актуальность информации на платформе и предоставляет возможность пользователям искать жилье по определенным критериям. В листинге 3.5 будет представлен код для создания жилья.

export const createHousing = async (req: Request, res: Response): Promise<void> => {

  const { name, description, location, pricePerNight, ownerId, categoryId, criteria } = req.body;

  const imageUrl = req.file ? `/images/${req.file.filename}` : null;

  if (!name || !description || !location || !pricePerNight || !ownerId || !categoryId) {

    res.status(400).json({ message: 'Все поля, кроме изображения, обязательны' });

    return;

  }

  try {

    const newProperty = await prisma.property.create({

      data: {

        name,

        description,

        location,

        pricePerNight: parseFloat(pricePerNight),

        owner: {

          connect: { id: parseInt(ownerId) },

        },

        category: {

          connect: { id: parseInt(categoryId) },

        },

        imageUrl,

      },

    });

};

Листинг 3.5 – Код для создания жилья

Контроллер createHousing отвечает за процесс создания нового жилья в системе. Вначале он извлекает данные из тела запроса, такие как название жилья, описание, местоположение, цена за ночь, ID владельца, ID категории и критерии. Также, если изображение жилья было загружено, извлекается путь к файлу. После этого контроллер проверяет, что все обязательные поля присутствуют в запросе. Если какое-то поле отсутствует (кроме изображения), возвращается ошибка с кодом 400 и соответствующим сообщением. Это позволяет предотвратить создание записи с недостаточным набором данных. Если все поля в порядке, контроллер использует Prisma для создания записи о жилье в базе данных. Он связывает новое жилье с владельцем и категорией через их ID, передаваемые в запросе. В случае наличия изображения, сохраняется путь к этому изображению. Если в запросе также указаны критерии для жилья (например, наличие бассейна или Wi-Fi), контроллер связывает эти критерии с новым жильем через промежуточную таблицу, добавляя соответствующие записи в базу данных. По завершении всех операций контроллер возвращает клиенту ответ с кодом 201 (успешно создано), включая данные нового жилья. В случае ошибки (например, при взаимодействии с базой данных) контроллер перехватывает исключение и отправляет ошибку с кодом 500, информируя клиента о проблемах на сервере. Таким образом, контроллер эффективно управляет процессом добавления нового жилья в систему, обеспечивая валидацию данных и корректное взаимодействие с базой данных.

В листинге 3.6 будет представлен код для удаления жилья.

export const deleteHousing = async (req: Request, res: Response): Promise<void> => {

  try {

    const property = await prisma.property.findUnique({

      where: { id: parseInt(id) },

      include: { bookings: true, criteria: true

      },

    });

    await prisma.booking.deleteMany({

      where: {propertyId: parseInt(id),

      },

    });

    await prisma.propertyCriterion.deleteMany({

      where: {propertyId: parseInt(id),

      },

    });

    await prisma.property.delete({

      where: { id: parseInt(id) },

    });

  }

};

Листинг 3.6 – Код для удаления жилья

Контроллер deleteHousing отвечает за удаление жилья из системы. Когда поступает запрос на удаление, контроллер извлекает ID жилья из параметров запроса. Также он получает данные о текущем пользователе, которые предположительно были добавлены в объект req через middleware, например, для проверки авторизации. На первом этапе контроллер проверяет, авторизован ли пользователь. Если данных о пользователе нет (например, он не прошел аутентификацию), возвращается ошибка с кодом 401 (неавторизован) и сообщением, что нужно авторизоваться. Затем контроллер пытается найти жилье в базе данных по ID, который был передан в запросе. Вместе с данными о жилье он также запрашивает связанные с ним бронирования и критерии (если они есть), чтобы удостовериться, что все связанные сущности будут корректно обработаны. Если жилье не найдено, возвращается ошибка с кодом 404 (не найдено), и процесс удаления не продолжается. После того как жилье найдено, контроллер проверяет, имеет ли пользователь право на его удаление. Это право есть только у администратора или владельца этого жилья. Если пользователь не является владельцем жилья или администратором, возвращается ошибка с кодом 403 (Forbidden), информируя, что у пользователя нет прав на выполнение данного действия. Если проверки прошли успешно, контроллер сначала удаляет все бронирования, связанные с данным жильем, через метод deleteMany. Затем он удаляет все связи этого жилья с критериями через таблицу propertyCriterion, которая является промежуточной. Эти операции необходимы для того, чтобы в базе данных не оставалось зависимых данных, которые могут нарушить целостность данных после удаления самого жилья. После того как все зависимые сущности были удалены, контроллер удаляет само жилье через метод delete, и отправляет пользователю ответ с кодом 200 (OK) и сообщением о том, что жилье и все связанные с ним данные были успешно удалены. Если в процессе удаления возникает ошибка, она перехватывается, и пользователю отправляется ответ с кодом 500 (Internal Server Error), а также логируется информация о произошедшей ошибке для дальнейшего анализа. Таким образом, данный контроллер обеспечивает не только удаление самого жилья, но и гарантирует, что все связанные данные, такие как бронирования и критерии, будут корректно удалены, предотвращая проблемы с целостностью базы данных.

В листинге 3.7 будет представлен код для редактирования жилья.

export const updateProperty = async (req: Request, res: Response): Promise<void> => {

  try {

    const updatedProperty = await prisma.property.update({

      where: { id: Number(id) },

      data: {

        name,

        description,

        location,

        pricePerNight: parseFloat(pricePerNight),

        category: categoryId ? { connect: { id: parseInt(categoryId) } } : undefined,

        ...(imageUrl && { imageUrl }),

      },

    });

    if (validCriteria.length > 0) {

      await prisma.propertyCriterion.deleteMany({

        where: { propertyId: Number(id) },

      });

      const criterionConnections = validCriteria.map((criterionId: number) => ({

        propertyId: updatedProperty.id,

        criterionId,

      }));

      await prisma.propertyCriterion.createMany({

        data: criterionConnections,

      });

    }

};

Листинг 3.7 – Код для редактирования жилья

Контроллер updateProperty отвечает за обновление информации о жилье. Сначала он извлекает ID жилья из параметров запроса и данные для обновления из тела запроса. Если переданы данные для загрузки нового изображения, контроллер сохраняет путь к файлу. Если передается критерий (criteria), контроллер проверяет, является ли он строкой или массивом. Если это строка, она преобразуется в массив с проверкой на корректность значений. Если это уже массив, фильтруются только валидные числовые значения. Затем контроллер пытается найти жилье в базе данных по ID. Если жилье не найдено, возвращается ошибка с кодом 404 (не найдено). Если жилье найдено, контроллер обновляет его данные в базе, включая основную информацию (название, описание, местоположение, цена, категория) и, при наличии, новое изображение. Если были переданы корректные критерии, сначала удаляются старые связи этого жилья с критериями, а затем создаются новые, используя промежуточную таблицу для связи жилья и критериев. Наконец, обновленные данные о жилье возвращаются в ответе с кодом 200 (OK). Если возникает ошибка, она логируется, и пользователю отправляется сообщение об ошибке с кодом 500 (внутренняя ошибка сервера).

* + 1. Бронирование жилья

Бронирование жилья — это процесс, позволяющий пользователям резервировать выбранное жилье на определенные даты. В системе бронирования важно учитывать доступность жилья на заданный период, чтобы избежать пересечений с уже подтвержденными бронированиями. Этот функционал обычно включает проверку на доступность, создание заявки на бронирование и управление статусами бронирований (ожидание, подтверждение или отмена). Реализация бронирования будет показана в листинге 3.8.

export const createBooking = async (req: Request, res: Response): Promise<void> => {

  const { propertyId, userId, startDate, endDate } = req.body;

  try {

    const existingBookings = await prisma.booking.findMany({

        where: {

          propertyId: propertyId,

          status: 'CONFIRMED',

          OR: [{

              startDate: { gte: new Date(startDate) },

              endDate: { lte: new Date(endDate) }

            },

            {

              startDate: { lte: new Date(endDate) },

              endDate: { gte: new Date(startDate) }

            }

          ],

        },

      });

    const newBooking = await prisma.booking.create({

      data: {

        userId,

        propertyId,

        startDate: new Date(startDate),

        endDate: new Date(endDate),

        status: 'PENDING',

      },

    });

};

Листинг 3.8 – Создание бронирования жилья

Контроллер createBooking используется для обработки запросов на создание бронирования жилья. Основная задача этого контроллера — гарантировать, что новое бронирование не пересекается с уже существующими, и создать запись о бронировании в базе данных. Когда клиент отправляет запрос, содержащий propertyId (идентификатор жилья), userId (идентификатор пользователя), а также даты начала (startDate) и окончания (endDate) бронирования, контроллер начинает выполнение с проверки доступности жилья на указанный период. Первый шаг — это поиск существующих подтвержденных бронирований (status: 'CONFIRMED') для данного жилья, которые пересекаются с указанным диапазоном дат. Если хотя бы одно пересечение найдено, контроллер возвращает ответ с кодом состояния 400 и сообщает, что указанный период занят. Если никаких пересечений не обнаружено, система переходит к созданию нового бронирования. Для этого в базу данных добавляется запись с переданными userId, propertyId, датами начала и окончания. Новое бронирование изначально получает статус PENDING, что означает, что оно ожидает подтверждения. Это может быть полезно, если система предполагает ручное или автоматическое подтверждение бронирований на следующем этапе. В случае успешного создания бронирования контроллер возвращает ответ с кодом состояния 201, содержащий данные о созданной записи. Если в процессе выполнения возникает ошибка (например, из-за проблем с базой данных), контроллер обрабатывает её и возвращает ответ с кодом состояния 500, указывая, что произошла ошибка на сервере. Этот подход обеспечивает целостность данных и предотвращает конфликты бронирований, сохраняя удобство для пользователей и стабильность работы системы.

* + 1. Оставление отзывов с оцениванием после сделки

Оставление отзывов и оценивание после сделки — это важный механизм, который позволяет пользователям делиться своим опытом взаимодействия с системой или другими пользователями. В контексте аренды жилья отзывы служат инструментом обратной связи, где арендаторы могут оценивать жилье, делиться своими впечатлениями о его состоянии, удобствах и соответствии ожиданиям. Оценки помогают новым пользователям принимать более осознанные решения, а также мотивируют владельцев жилья поддерживать высокий уровень сервиса. Логика работы этого функционала будет представлена в листинге 3.9.

export const addReview = async (req: Request, res: Response): Promise<void> => {

  try {

    if (existingReview) {

      const newReview = await prisma.review.create({

        data: {

          propertyId: Number(propertyId),

          userId: Number(userId),

          rating,

          comment,

        },

        include: { user: true }, // Подключаем данные пользователя

      });

      res.status(201).json(newReview);  // Отправляем новый отзыв

    }

  }

};

Листинг 3.9 – Код для добавления отзыва

Контроллер addReview отвечает за добавление или обновление отзыва пользователя для конкретного жилья. Его задача — обработать данные, отправленные клиентом, проверить существование отзыва и в зависимости от результата либо создать новый, либо обновить уже существующий. Когда запрос поступает, контроллер сначала извлекает из тела запроса следующие данные: идентификатор жилья (propertyId), идентификатор пользователя (userId), оценку (rating) и комментарий (comment). Эти данные нужны, чтобы связать отзыв с конкретным жильем и пользователем. Затем контроллер проверяет, существует ли уже отзыв, оставленный этим пользователем для указанного жилья. Для этого в базе данных выполняется поиск отзыва по уникальной комбинации propertyId и userId. Если такой отзыв находится, он обновляется: заменяются старая оценка и комментарий на новые данные, предоставленные клиентом. После обновления контроллер возвращает обновленный отзыв с кодом успешного выполнения 200. Если отзыв не найден, контроллер создает новый. Он записывает в базу данных все предоставленные данные, включая идентификатор жилья, пользователя, оценку и комментарий. Новый отзыв возвращается клиенту с кодом успешного создания 201. На каждом этапе проверяются возможные ошибки. Если происходит сбой, например, при работе с базой данных, контроллер регистрирует ошибку в журнале и возвращает клиенту сообщение с кодом состояния 500, указывающим на ошибку сервера. Таким образом, этот контроллер реализует удобный механизм, который автоматически определяет, нужно ли обновить существующий отзыв или создать новый, упрощая взаимодействие пользователя с системой.

* 1. Разработка клиентской части

Клиентская часть веб-приложения была разработана с учетом принципов кроссплатформенности и работает в браузере. Все данные обновляются динамически, благодаря чему взаимодействие с интерфейсом становится плавным и удобным.

Для обмена данными с сервером клиентская часть использует встроенный API fetch. Эта технология позволяет отправлять запросы к серверу и получать ответы в асинхронном режиме, что предотвращает задержки в работе интерфейса. Запросы отправляются с использованием различных HTTP-методов, таких как GET, POST, PUT и DELETE, на заранее определённые пути.

Подход с использованием fetch обеспечивает простоту реализации и достаточную гибкость для большинства задач. Это позволяет динамически обновлять данные на страницах, не прерывая работу приложения. Благодаря этому интерфейс остаётся отзывчивым, а взаимодействие пользователя с приложением — интуитивно понятным.

На рисунке 3.3 представлена структура проекта.

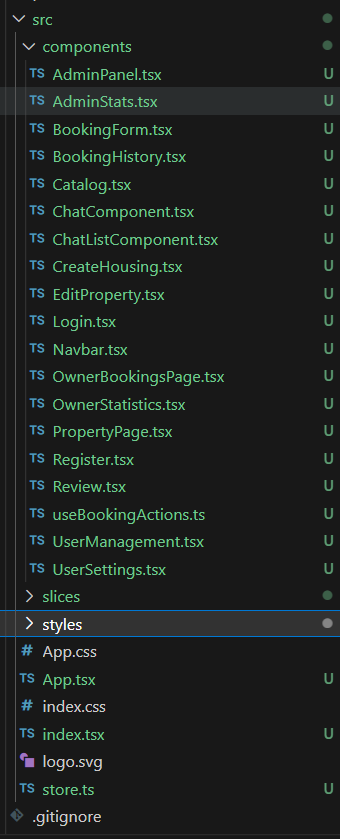


Рисунок 3.3 – Структура проекта

Директория components содержит отдельные файлы компонентов приложения, организованных по функциональному назначению. Это основные части интерфейса. Папка slices содержится userSlice.ts, App.tsx – главный компонент, в котором определены маршруты приложения и подключение основных компонентов, index.tsx – точка входа в клиентское приложение, store.ts – содержит настройку Redux-хранилища для глобального управления состоянием, package.json – файл конфигурации npm, где указаны зависимости, скрипты и метаинформация о проекте, tsconfig.json –конфигурационный файл TypeScript, задающий правила компиляции и обработки кода.

В листинге 3.10 будет представлен слайс, который управляет состоянием.

import { createSlice, PayloadAction } from '@reduxjs/toolkit';

const initialState: UserState = {

  role: localStorage.getItem('role') || null,

  userName: localStorage.getItem('userName') || null,

  userId: localStorage.getItem('userId') || null,

};

const userSlice = createSlice({

  name: 'user',

  initialState,

  reducers: {

    setUserRole(state, action: PayloadAction<string>) {

      state.role = action.payload;

      localStorage.setItem('role', action.payload); // Сохраняем роль в localStorage

    },

    setUserName(state, action: PayloadAction<string>) {

      state.userName = action.payload;

      localStorage.setItem('userName', action.payload);

    },

    setUserId(state, action: PayloadAction<string>) {

      state.userId = action.payload;

      localStorage.setItem('userId', action.payload);

    },

    logout(state) {

      state.role = null;

      state.userName = null;

      state.userId = null;

      localStorage.removeItem('role');

      localStorage.removeItem('userName');

      localStorage.removeItem('userId');

    },

  },

});

Листинг 3.10 – Слайс для хранения состояния

userSlice представляет собой слайс состояния для управления данными о пользователе в Redux. В нем хранятся три основных свойства: роль пользователя (role), имя пользователя (userName) и идентификатор пользователя (userId). Эти данные сохраняются в localStorage, чтобы оставаться доступными при перезагрузке страницы. Инициализация состояния происходит с извлечением этих значений из localStorage, если они там есть.

Слайс включает несколько редьюсеров: setUserRole обновляет роль пользователя в состоянии и сохраняет её в localStorage, setUserName обновляет имя пользователя и также сохраняет его в localStorage, setUserId обновляет идентификатор пользователя и сохраняет его в localStorage, logout сбрасывает данные о пользователе в состоянии и удаляет информацию из localStorage, что позволяет реализовать выход из системы.

Состояние хранится в Redux, и данные о пользователе могут быть легко обновлены с помощью этих редьюсеров.

store будет представлен в листинге 3.11.

import { configureStore } from '@reduxjs/toolkit';

import userReducer from './slices/userSlice';

export const store = configureStore({

  reducer: {user: userReducer,},});

export type RootState = ReturnType<typeof store.getState>;

export type AppDispatch = typeof store.dispatch;

Листинг 3.11 – Код хранилища

store — это конфигурация Redux-хранилища, которое управляет глобальным состоянием приложения. Хранилище настраивается с использованием configureStore из библиотеки @reduxjs/toolkit. В данном случае, хранилище включает редьюсер пользователя (userReducer), который управляет состоянием, связанным с пользователем.

Для удобства работы с состоянием и диспетчером в коде также определены типы: RootState — тип, описывающий всё состояние приложения, который позволяет использовать типизацию при доступе к состоянию, AppDispatch — тип диспетчера, который используется для отправки экшенов в *Redux*.

Таким образом, store централизует управление состоянием приложения и позволяет эффективно работать с данными о пользователе.

* 1. Выводы по разделу

В данном разделе рассмотрены структура и технологии как серверной, так и клиентской частей приложения. Серверная часть основана на *Node.js*, который реализует регистрацию, авторизацию, управлением жильём, бронирование жилья и оставление отзывов. Клиентская часть построена на *React* с *React-Redux* для управления состоянием, а маршрутизация и асинхронная загрузка страниц реализованы с использованием react-router-dom.

1. Тестирование программного средства

Ф.И.О

Подпись

Дата

Лист

1

БГТУ 04.00.ПЗ

Разраб.

*Яковлева Е.И.*

Пров.

Кантарович В.С.

Н. контр.

Утв.

Смелов В.В.

4 Тестирование программного средства

Лит.

Листов

11

*74218051, 2023*

У

Николайчук А.Н.

Процесс тестирования является не менее важным, чем процесс разработки приложения, поскольку на данном этапе можно найти недочеты в работе приложения и исправить их. В данной главе будут описаны проведения тестирования на серверной и клиентской частях дипломного проекта.

Необходимо протестировать корректность функционирования базовых операций, доступных пользователям. Также стоит удостовериться, что при заведомо неверных действиях система предотвратит их и оповестит об ошибке. Для исключения возможности возникновения ошибок необходимо протестировать каждый из компонентов системы по отдельности.

* 1. Тестирование серверной части

Для тестирования серверной части была использована программа Postman для проверки корректности полученных ответов. На рисунке 4.1 изображен ответ от сервера, а именно получение 401 ошибки, если пользователь ввел неверный пароль при авторизации. В случае если пользователь введет неверный логин, то он получит сообщение, что такого пользователя не существует в системе.

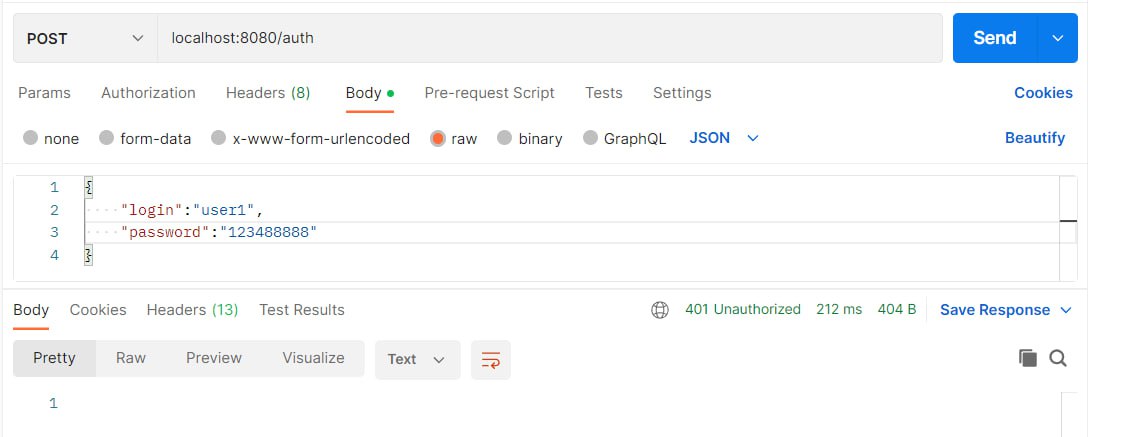


Рисунок 4.1 – Попытка входа с неверным паролем

В случае если пользователь ввел верные данные при авторизации, ему будет отправлен JWT-токен, с помощью которого он получит доступ к возможностям приложения. Теперь пользователь может осуществлять все действия, доступные для его роли. В случае отправления запроса без токена, в ответ будет отправлена статус ошибки 401, который свидетельствует о том, запрос не был применен, поскольку ему не хватает действительных учетных данных для целевого ресурса [18].

Пример успешной авторизации продемонстрирован на рисунке 4.2.

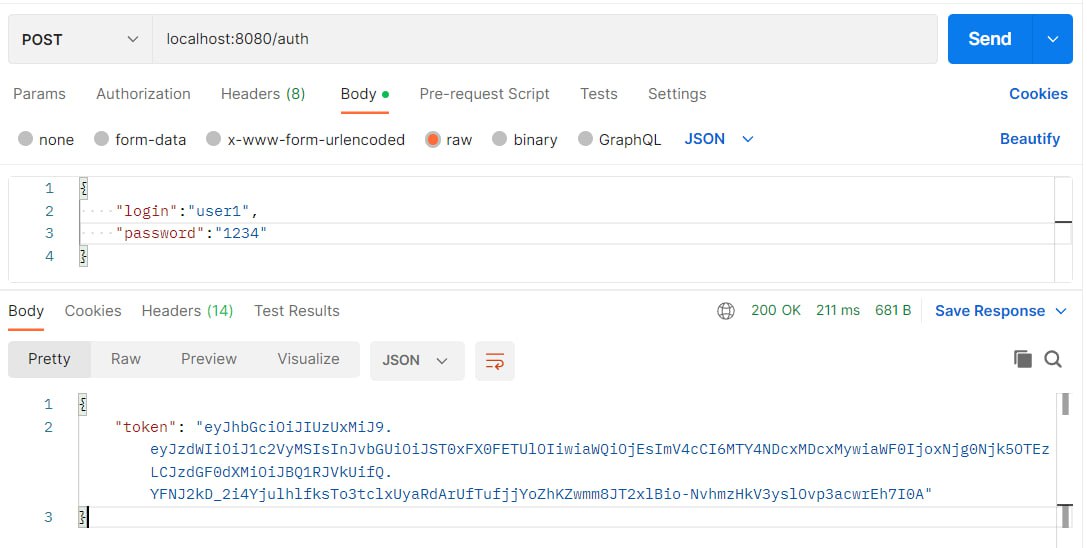


Рисунок 4.2 – Успешная авторизация пользователя

На рисунке 4.3 продемонстрировано получение данных профиля всех сотрудников. Для осуществления авторизации в Postman токен необходимо вставить во вкладке Authorization. Аналогичные проверки были произведены и для других ресурсов, доступ к которым пользователь не должен иметь без статического токена.

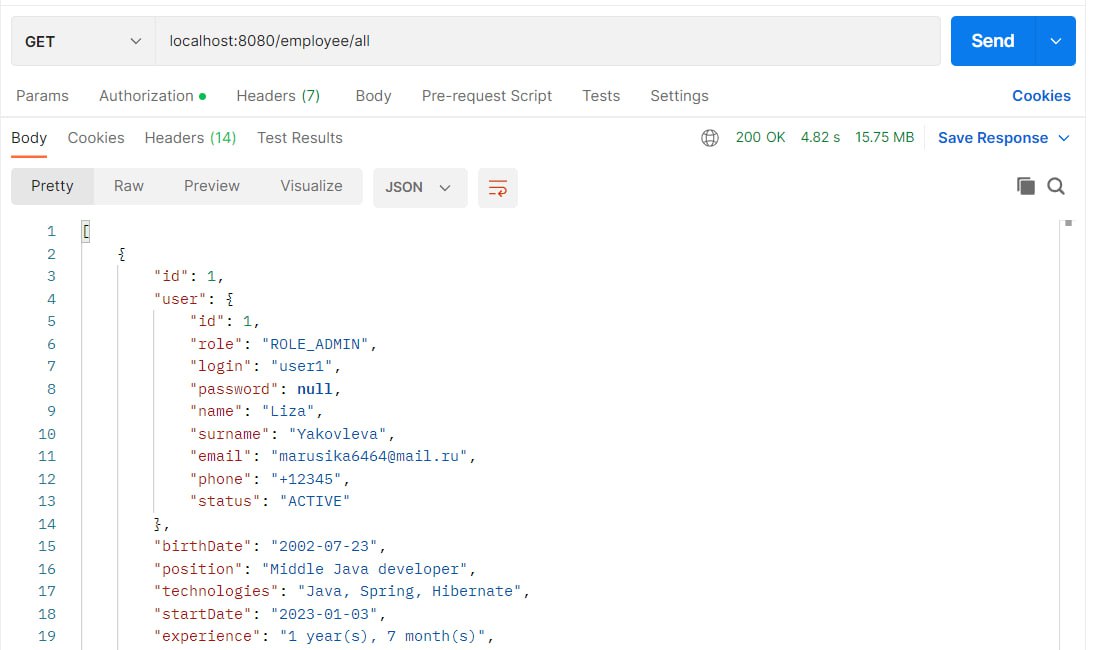


Рисунок 4.3 – Получение данных всех сотрудников

На рисунке 4.4 показана попытка получения данных о всех сотрудниках без прикрепления к запросу токена. Как и ожидалось, получена 401 ошибка. Все запросы к приложению, за исключением авторизационного, нуждаются в наличии токена.

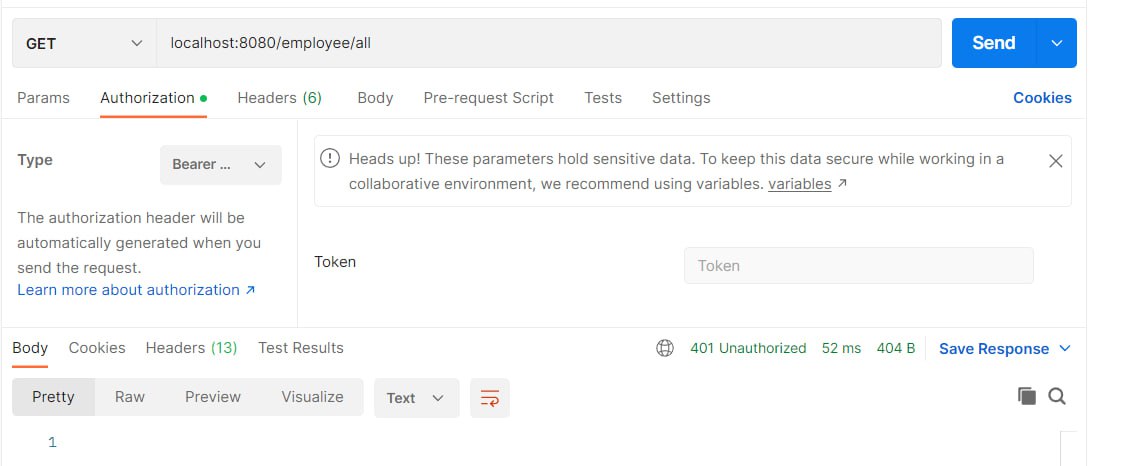


Рисунок 4.4 – Попытка получения данных без токена

Также необходимо удостовериться, что даже при наличии аутентификационного токена, пользователь не сможет успешно выполнить запросы, на которые у него нет прав. Например, на рисунке 4.5 продемонстрировано, как пользователь с ролью сотрудника пытается получить список всех заказов, что доступно только пользователю с ролью менеджера или администратора.

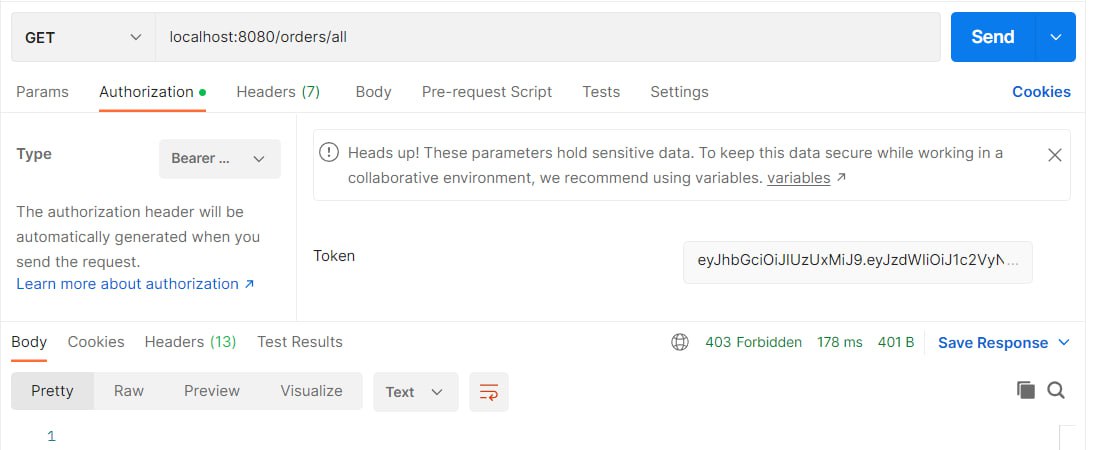


Рисунок 4.5 – Попытка получения данных с недостаточными правами

Как можно увидеть, пользователю была отправлена ошибка со статусом 403, которая означает ограничение или отсутствие доступа к данным. Такая ошибка будет отправлена во всех идентичных ситуациях. Аналогичные проверки были произведены и для других ресурсов, доступ к которым пользователь должен иметь только при наличии соответствующих прав.

Далее необходимо проверить осуществление валидации при POST- и PUT-запросах. Сервер должен вернуть понятное сообщение, чтобы пользователь мог исправить свою ошибку. Например, на рисунке 4.6 приведена попытка редактирования задачи, при которой имя задачи не заполнено. В ответ пользователю ясно сказано, что имя задачи не может быть пустым.

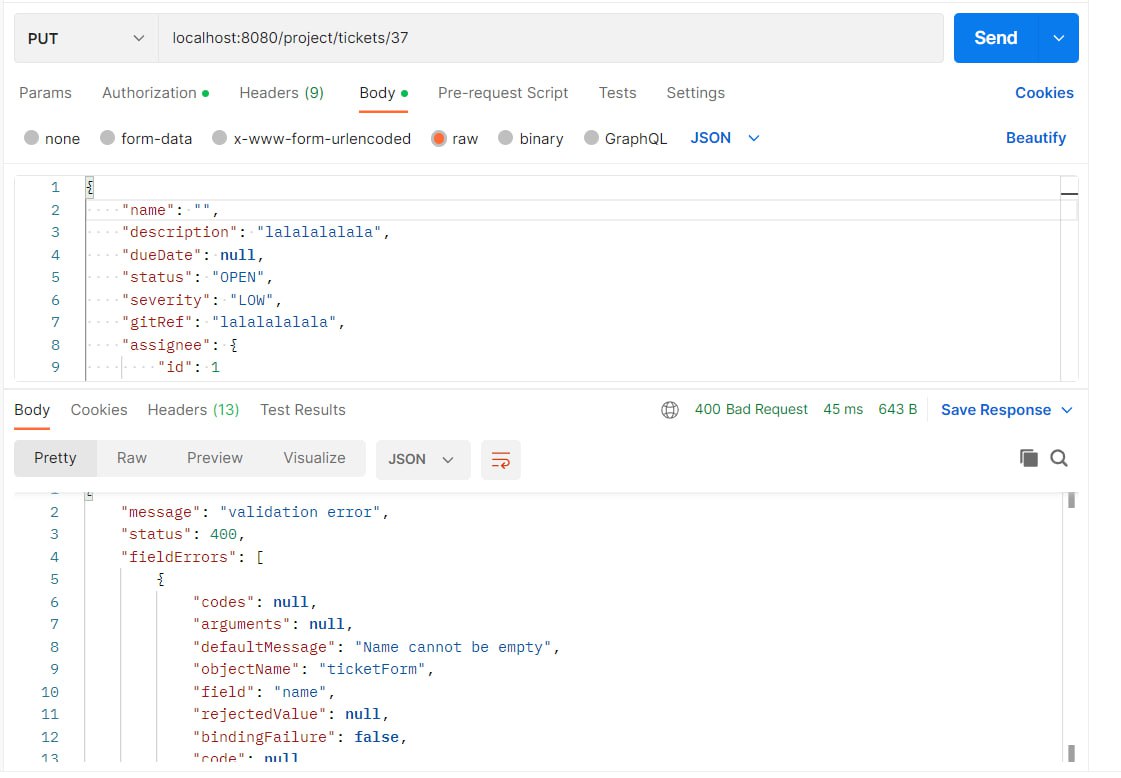


Рисунок 4.6 – Ошибка валидации при редактировании задачи

В случае, если пользователь корректно заполнит все данные и повторит свой запрос, то в ответ будет получен статус 200, информирующий о корректном завершении запроса. Пример приведен на рисунке 4.7.

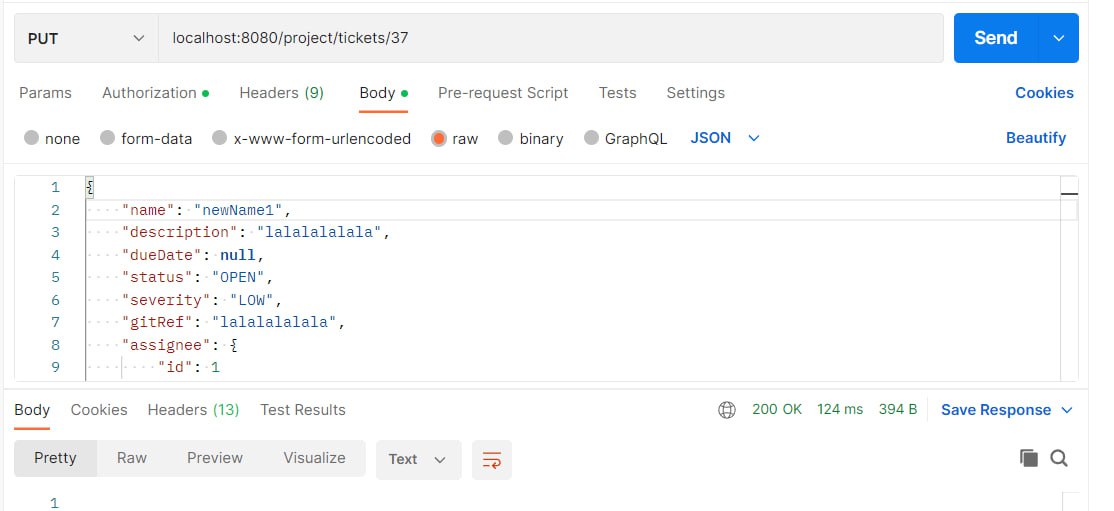


Рисунок 4.7 – Успешное редактирование задачи

Дополнительно проверим успешность редактирования, посмотрев, сохранился ли результат в базе данных. Для этого воспользуемся инструментом для визуализации базы данных MySQL Workbench. На рисунке 4.8 продемонстрировано, что действительно задаче было присвоено новое имя.

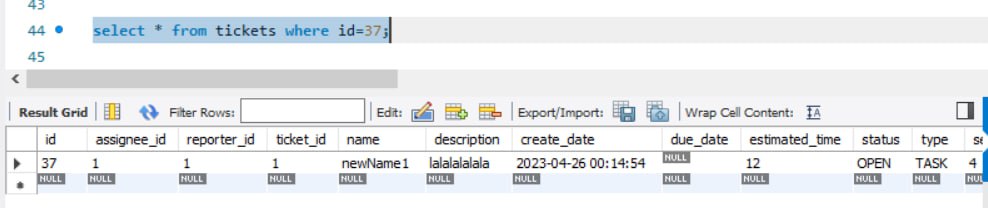


Рисунок 4.8 – Измененная задача в MySQL Workbench

Также важно осуществить проверку на ответ сервера при попытке доступа к ресурсу, которого не существует, поскольку важно, чтобы не возникло исключительной ситуации, приводящей к необработанной ошибке, а было отправлено понятное пользователю сообщение. Например, на рисунке 4.9 происходит попытка получения данных задачи, которой не существует. На это пользователю отображен ответ, информирующий пользователя о данной ситуации.

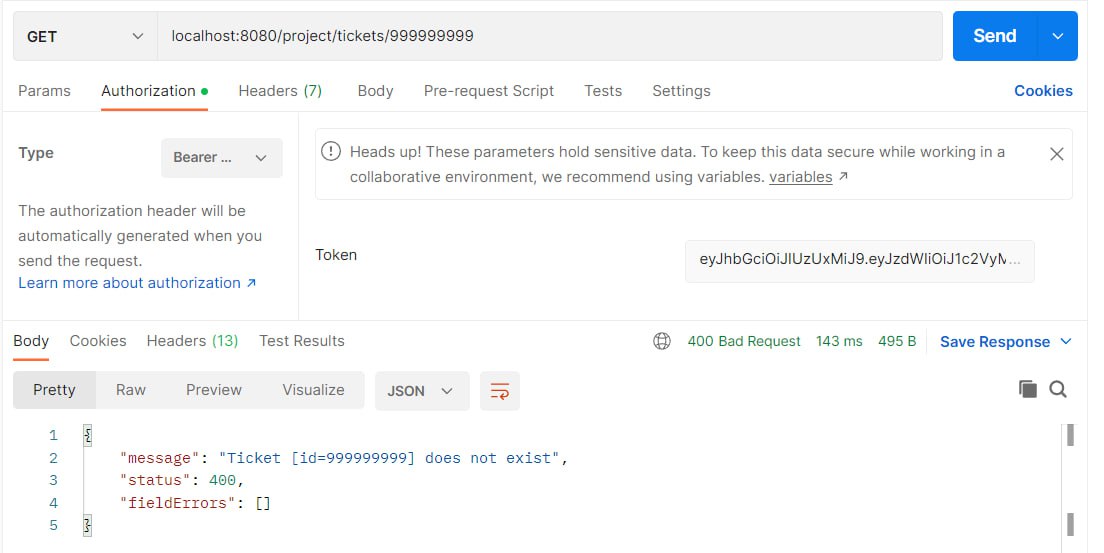


Рисунок 4.9 – Получения данных несуществующего ресурса

Идентичная обработка несуществующего ресурса будет происходит и при попытке получения данных несуществующего пользователя, и при попытке получения данных несуществующей задачи и так далее.

Если аккаунт пользователя по какой-либо причине был заблокирован администратором, даже при вводе правильных логина и пароля заблокированный пользователь ни в коем случае не должен быть допущен в приложение, поэтому важно осуществить проверку данной ситуации. На рисунке 4.10 видно, что при попытке заблокированного пользователя зайти в приложение, была получена ошибка со статусом 423. В данном случае эта ошибка отображает, что аккаунт пользователя был заблокирован и теперь он не имеет доступа к приложению.

В случае если администратор разблокирует пользователя, то ему будет возвращен доступ к приложению, и авторизация пройдет успешно.

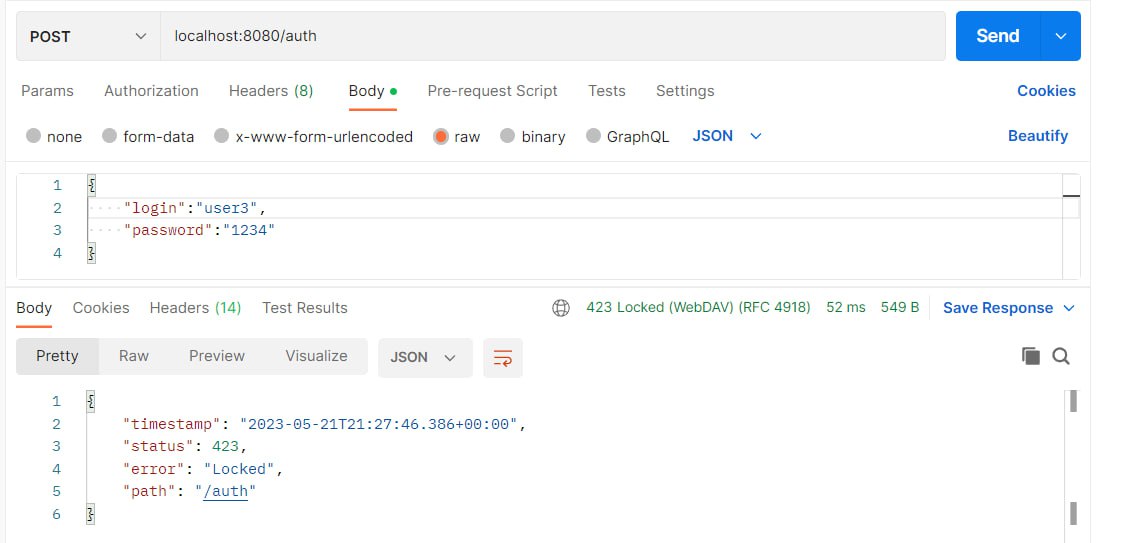


Рисунок 4.10 – Авторизация заблокированного пользователя

Таким образом было проведено ручное тестирование серверной части. Как можно увидеть, все тесты успешно прошли работу, что означает корректную работу модулей всего сервера. Исключительных ситуаций, которые могли привести к нестабильной работе приложения, найдено не было.

* 1. Тестирование клиентской части

Перейдем к тестированию клиентской части приложения и выполним аналогичные действия со стороны клиента и проверим правильность отображаемых данных. Сообщения об ошибках, отправляемые серверной частью, должны корректно отображаться и в клиентском интерфейсе.

Для начала перейдем к странице авторизации. Если пользователь ввел неверный пароль, ему будет отображено сообщение об этом, как показано на рисунке 4.11.

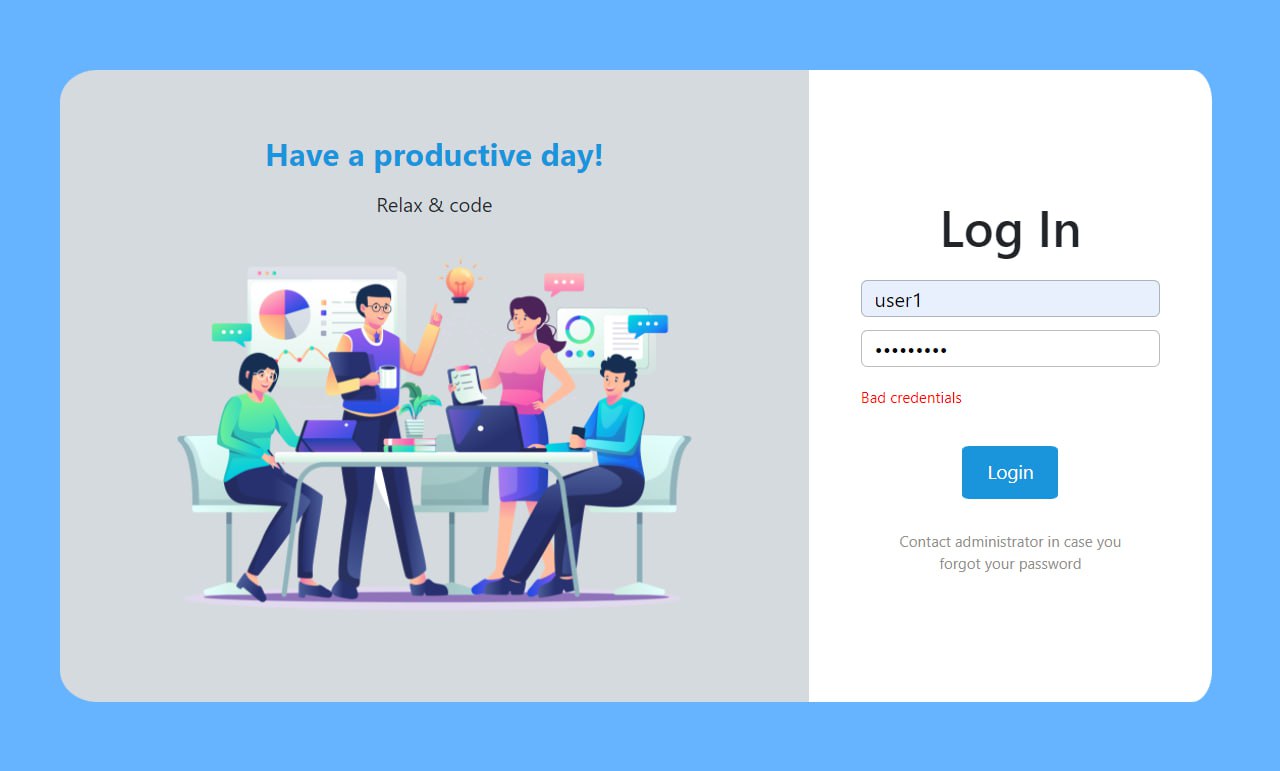


Рисунок 4.11 – Ввод неверного пароля при авторизации

Если заблокированный пользователь совершает попытку авторизации в приложение, то ему будет отображено сообщение об ошибке, как показано на рисунке 4.12.

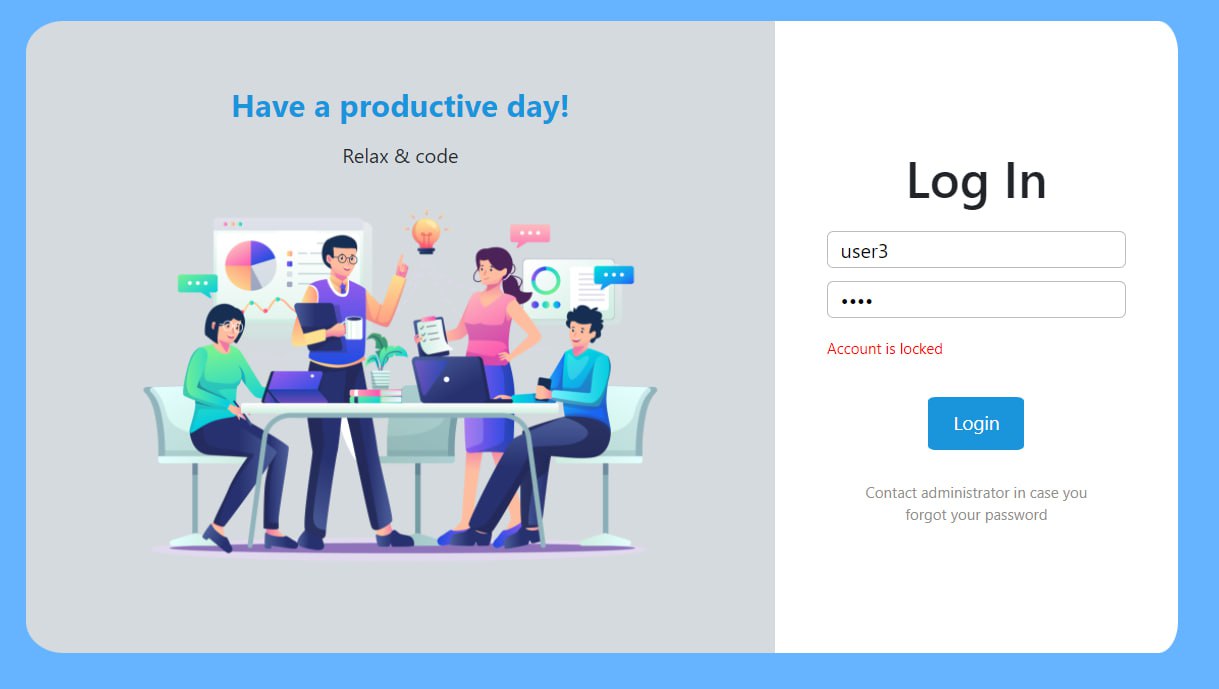


Рисунок 4.12 – Попытка входа заблокированного пользователя

На рисунке 4.13 продемонстрирована попытка создания задачи с некоторыми незаполненными обязательными полями, что нарушает правила валидации. В ответ пользователю отображено сообщение об ошибке, перечисляющее сразу все поля, которые обязательны и не были заполнены при отправке запроса.

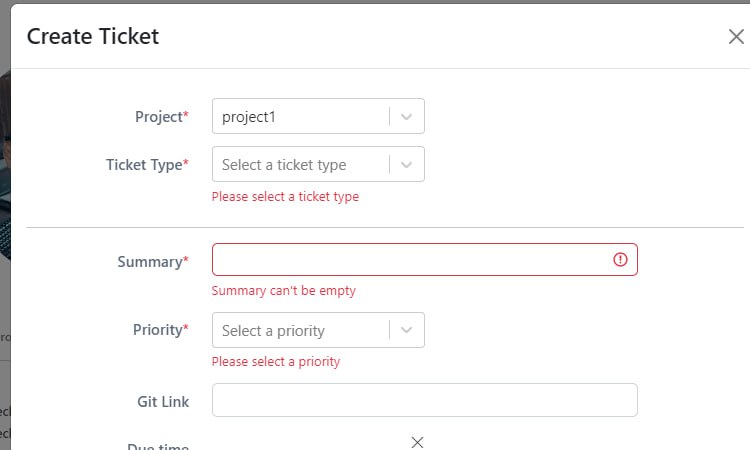


Рисунок 4.13 – Нарушения валидации при создании задачи

Если сотрудник попытается внести изменения на дэшборде чужого проекта, то появится сообщение, уведомляющее о запрете на данное действие, как показано на рисунке 4.14, и внесенные изменения будут отменены. Так же будет и в случае попытки клиента перетащить задачу на дэшборде даже своего проекта.

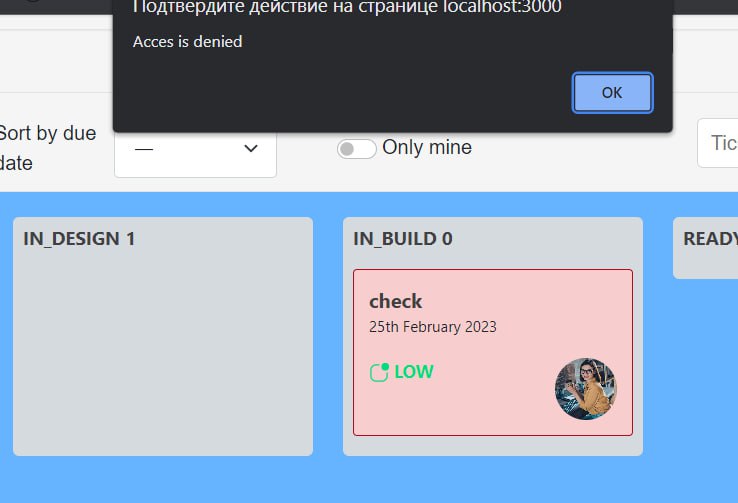


Рисунок 4.14 – Попытка изменения чужого дэшборда

Также было проверено, что при попытке сотрудника или клиента попасть на страницы, доступные только менеджерам и администратору, все работает корректно, и они действительно будут перенаправлены на доступные их ролям страницы.

Также была протестирована безопасность при использовании куков. Если, например, клиент путем изменения в куках номера проекта попытается получить доступ к дэшборду чужого проекта, он сразу будет переадресован на страницу с дэшбордом его проекта, а куки также автоматически будут заменены на корректные. Такие же операции будут произведены при попытке клиента получить доступ к детальной информации чужого проекта или задачи.

На каждой странице для каждой роли было проверено, что содержатся только те функции и действия, на которые у данной роли есть права. В частности, на рисунке 4.15 видно, что сотрудник может изменять и удалять только личные комментарии.



Рисунок 4.15 – Комментарии к задаче

Когда администратор решает деактивировать или заблокировать пользователя, для подтверждения действия ему отображается соответствующее подтверждающее всплывающее окно, как показано на рисунке 4.16.

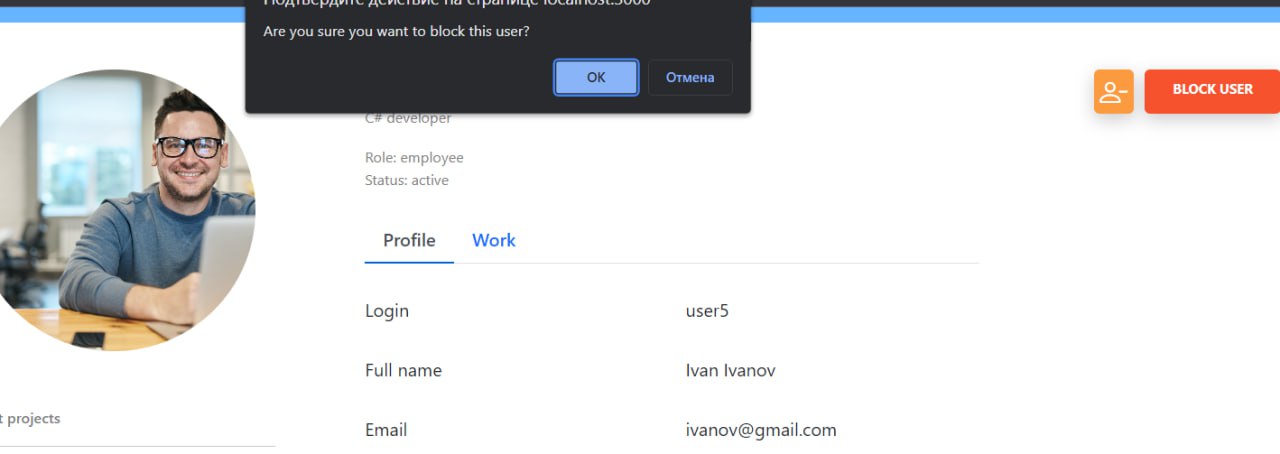


Рисунок 4.16 – Подтверждающее всплывающее окно

После того, как администратор деактивировал пользователя, данному пользователю на электронную почту приходит письмо с новым временным паролем для авторизации. Пример приведен на рисунке 4.17.

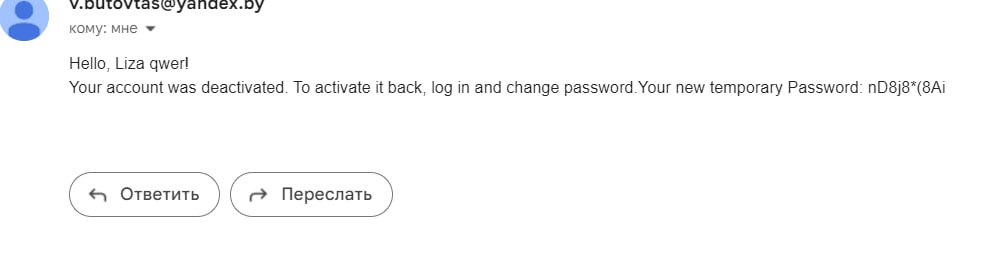


Рисунок 4.17 – Электронное письмо для деактивированного пользователя

Теперь при первом после деактивации входе этого пользователя в приложение, ему будет отображено модальное окно для смены пароля. Такое окно продемонстрировано на рисунке 4.18. В нем необходимо ввести текущий временный пароль, а также новый и повторить новый пароль. Если новый пароль в двух полях не совпадает, то будет отображена ошибка, а кнопка подтверждения будет недоступной.

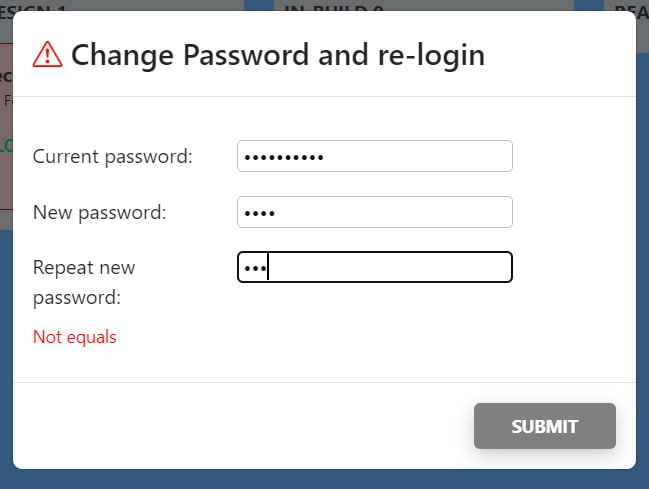


Рисунок 4.18 – Модальное окно для смены пароля

При редактировании задачи было проверена возможность перетаскивания файла в область для прикрепленного файла с целью замены. Как можно увидеть на рисунке 4.19, данная функция работает исправно.

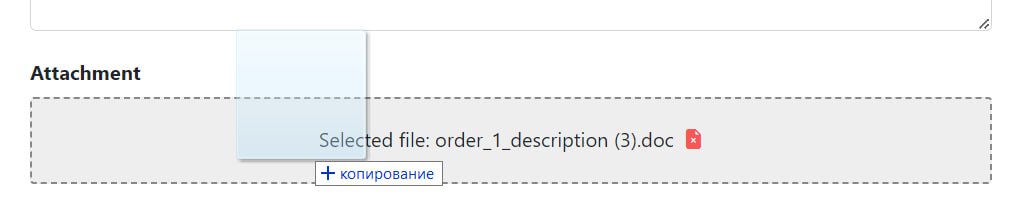


Рисунок 4.19 – Замена прикрепленного к задаче файла

На рисунке 4.20 показано, что при нажатии на файл, он был успешно скачан в папку загрузки на локальную машину.

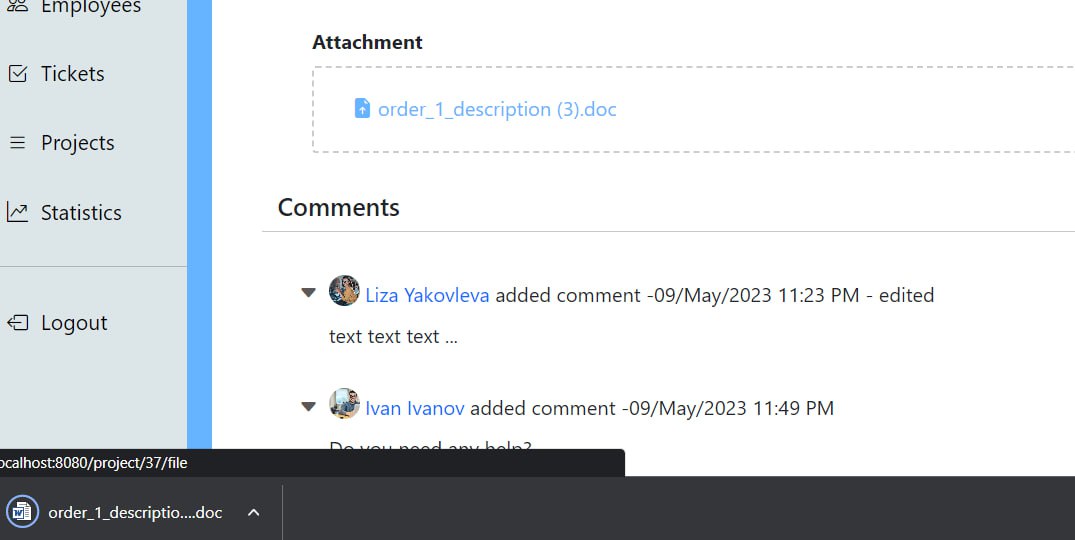


Рисунок 4.20 – Скачивание файла

Весь интерфейс приложения написан на английского языке. Это объясняется тем, что делается расчет на то, что все пользователи владеют английским языком, поскольку в настоящее время это одно из самых важных требований в IT-сфере. А также важным фактором является возможность работы с клиентами со всего мира за счет отсутствия языкового барьера в приложении. Но тем не менее, хоть в приложении и нет встроенной смены языка, при необходимости пользователь может перевести любую страницу приложения, воспользовавшись встроенным в браузер инструментом, таким как Google Translate, благодаря которому приложение может быть переведено буквально на любой язык мира. Пример перевода страницы профиля сотрудника с использованием данного средства представлен на рисунке 4.21.

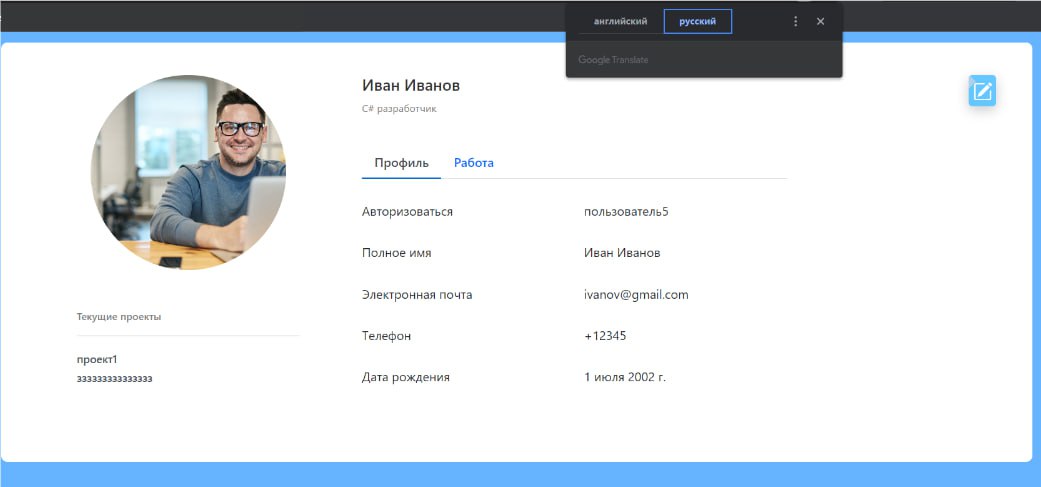


Рисунок 4.21 – Перевод страницы с использованием Google Translate

Тестирование клиентской части приложения можно считать успешно завершенным. Проблем выявлено не было.

* 1. Выводы по разделу

Основной целью данного раздела являлось тестирование всего дипломного для гарантии его целостной и стабильной работы. В качестве варианта проведения тестирования было выбрано ручное тестирование. Оно позволяет точно проверить отдельные элементы всего приложения в различных ситуациях. Были проверены как «позитивные», так и «негативные» тестовые случаи.

Для тестирования серверной части использовалось приложение Postman, которое позволяет формировать запросы, наполнять их данными и отправлять на сторону сервера. В ходе тестирования сервера были проверены такие основные ситуации, как получение данных без авторизации, отправка невалидных данных, проверка прав на доступ к данным. В ходе тестирования были проверены такие действия, как: авторизация, получение данных о всех сотрудниках, редактирование задачи.

Для тестирования клиентской части проекта был использован браузер Google Chrome. В ходе тестирования было проверено поведение клиентской части при пропуске необходимых полей, проверена работоспособность автоматической отправки электронных писем. На стороне клиента были протестированы такие действия как: авторизация, перетаскивание задач на дэшборде, редактирование файла задачи, деактивация пользователя, перевод страниц.

В ходе проведения тестирования всего проекта были обнаружены и устранены все уязвимости, которые могли повлечь за собой нарушение стабильности работы всего приложения и привести к потере пользовательских данных.

1. Руководство пользователя

Ф.И.О

Подпись

Дата

Лист

1

БГТУ 05.00.ПЗ

Разраб.

*Яковлева Е.И.*

Пров.

Кантарович В.С.

Н. контр.

Утв.

Смелов В.В.

5 Руководство пользователя

Лит.

Листов

14

*74218051, 2023*

У

Николайчук А.Н.

* 1. Развертывание приложения в среде Docker

Для проверки работоспособности приложения прежде всего необходимо установить на компьютере Docker. Никаких дополнительных экземпляров базы данных или, например, платформ как Java, NodeJS устанавливать не требуется, так как все образы будут скачаны через зависимости в конфигурационных файлах Docker из общедоступного централизованного хранилища. Таким образом, проект не требует жесткой привязки к какой-либо системе и может быстро и легко разворачиваться на любой операционной системе. Для создания самих контейнеров Docker нужно выполнить команду «docker compose up» в терминале, открытом в директории, где находится файл docker-compose. После этого все указанные в файле контейнеры будут автоматически созданы и запущены, и можно начать работать с приложением. Для этого нужно сделать переход по следующему URL в браузере: «http://localhost:3000».

Пример опубликованного в Docker с использованием Docker Compose приложения выглядит следующим образом, как показано ниже на рисунке 5.1.

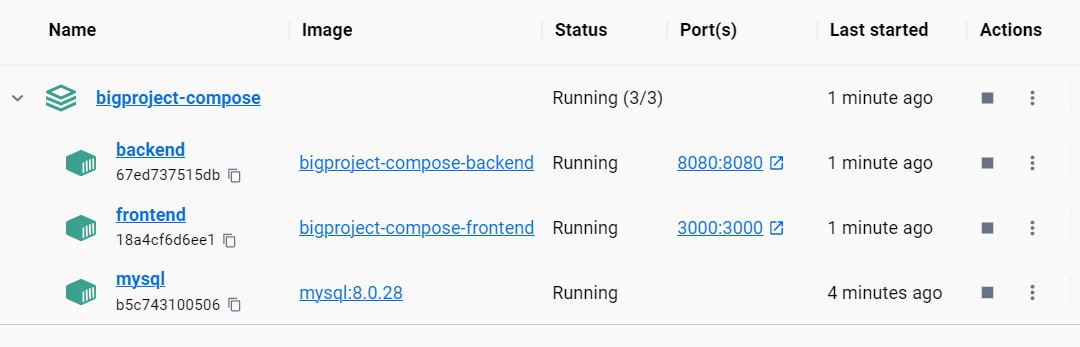


Рисунок 5.1 – Развернутое на Docker приложение

Чтобы проверить статус контейнеров можно выполнить команду как на рисунке 5.2, которая отобразит все существующие контейнеры и их состояния.



Рисунок 5.2 – Просмотр всех контейнеров в Docker

В данном случае запуск происходил на основе операционной системы Windows. Чтобы запустить разработанное приложение на другой машине, необходимо либо скопировать на нее все Docker-файлы, включая docker-compose, а самое главное код приложения, либо после создания образов, можно загрузить их в облачное хранилище, так называемое docker registry. В случае второго варианта будет возможность запустить приложение где угодно и когда угодно, не перенося на машину само приложение и конфигурационные файлы. Конечно, для запуска контейнеров необходим docker-compose файл, но имея готовые образы, его можно быстро и легко написать самостоятельно. При загрузке образов в репозиторий хранилища важно сделать репозиторий приватным, чтобы только определенные лица имели к нему доступ. Однако, чтобы иметь возможность создавать более одного приватного репозитория, будет необходимо приобрести подписку.

Образ приложения, загруженного в docker registry представлен на рисунке 5.3. В данном случае используется DockerHub. Как можно увидеть, один из образов был сделан приватным в целях безопасности.

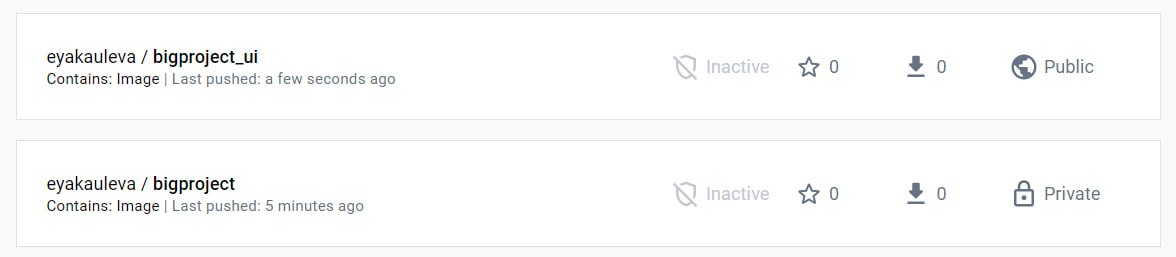


Рисунок 5.3 – Загруженные в docker registry образы приложения

Теперь для демонстрации можно легко запустить весь проект на другой машине с иной операционной системой, например, MacOS. Для этого будет использован идентичный ранее использованному docker-compose файл, за исключением того, что в данном случае вместо указания путей к Dockerfile файлам, будет указано полное название образов из хранилища, как это уже сделано в контейнере для базы данных MySQL, который использует готовый образ из хранилища. После запуска файла, будет получен результат, представленный на рисунке 5.4. Как можно увидеть, все контейнеры были успешно созданы и запущены.

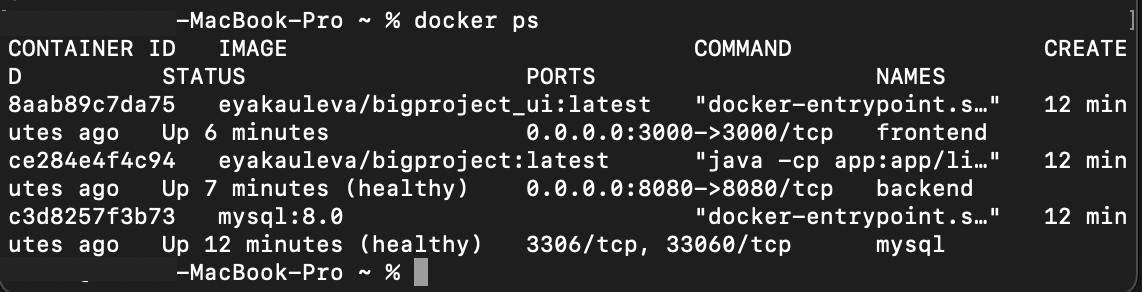


Рисунок 5.4 – Развернутое на Docker приложение на ОС MacOS

Таким образом, любая операционная система, имеющая поддержку контейнеризации или предустановленное средство контейнеризации Docker, может без всяких проблем запускать данное приложение и выполнять его стабильно без надобности установки дополнительных инструментов.

* 1. Руководство для сотрудника

Роль сотрудник предназначена для пользователя, являющегося сотрудником компании без особых прав. У него есть доступ ко всем базовым возможностям.

Для того чтобы начать работу с приложением, необходимо авторизоваться путем ввода своих логина и пароля. Страница авторизации приведена на рисунке 5.5.

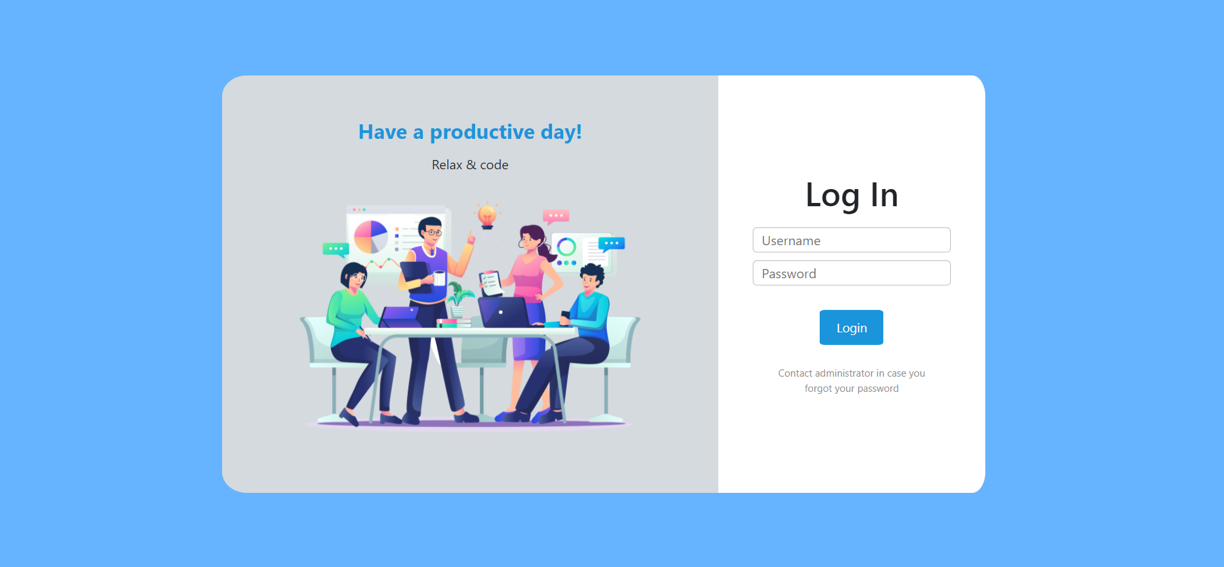


Рисунок 5.5 – Страница авторизации

После прохождения авторизации любой пользователь-сотрудник будет перенаправлен на страницу с его профилем. Пример страницы пользователя представлен на рисунке 5.6. Здесь во вкладке Profile отображена общая информация о пользователе, а во вкладке Work данные о профессиональной деятельности сотрудника. Пользователь может редактировать данные, нажав на кнопку редактирования, находящуюся в правом верхнем углу. Под фотографией находится список проектов, к которым закреплен сотрудник и которые находятся в разработке.

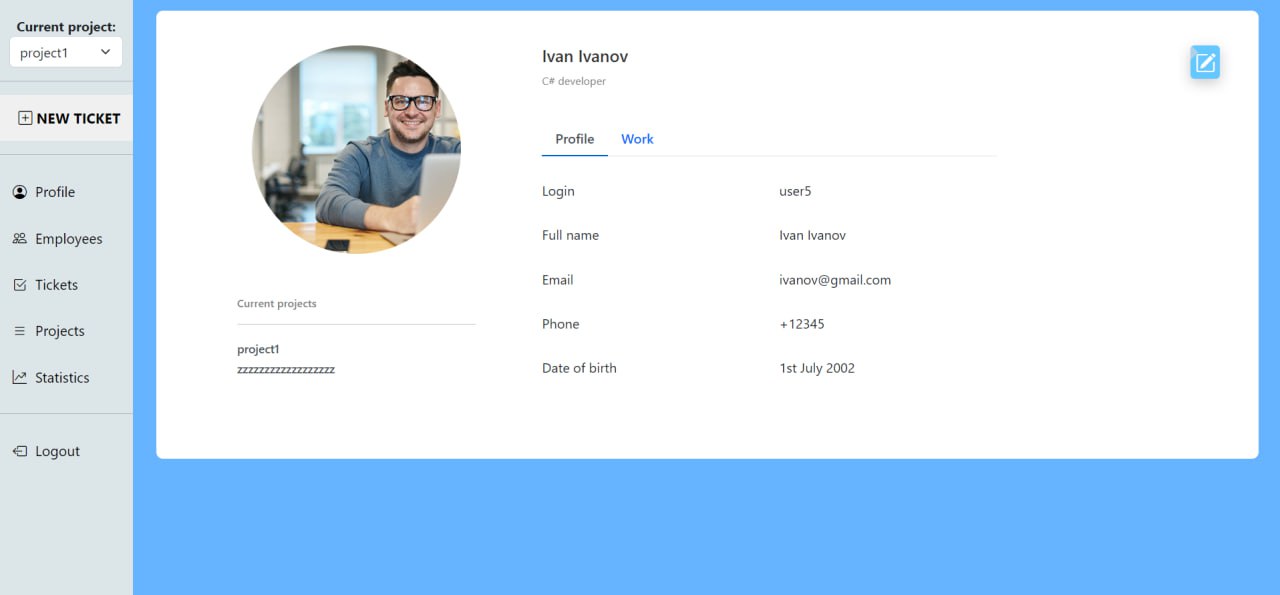


Рисунок 5.6 – Страница пользователя

Навигацию по приложению можно осуществлять, используя меню, находящееся с левой стороны. Вверху меню расположен выпадающий список, где сотрудник может переключать проекта, с которым он работает в данный момент. Ниже находится кнопка для создания задачи. Нажав на нее, появится модальное окно для создания задачи, представленное на рисунке 5.7. Поля, помеченные звездочкой, обязательны к заполнению. По умолчанию в поле Project будет выбран проект, выбранный как текущий в меню, но его можно изменить, нажав на поле, после чего будет отображен выдающий список с проектами для выбора. Для прикрепления файла к задаче можно либо кликнуть по полю Attachment, либо перетащить файл прямо в поле.



Рисунок 5.7 – Модальное окно для создания задачи

Далее в меню находится клавиша Profile, при нажатии направляющая пользователя на страницу с его профилем как на рисунке 5.6. Кнопка Employees в меню отображает пользователю всех сотрудников компании. Данная страница представлена на рисунке 5.8. Можно осуществлять поиск по имени и/или фамилии, используя поисковое поле в правом верхнем углу.

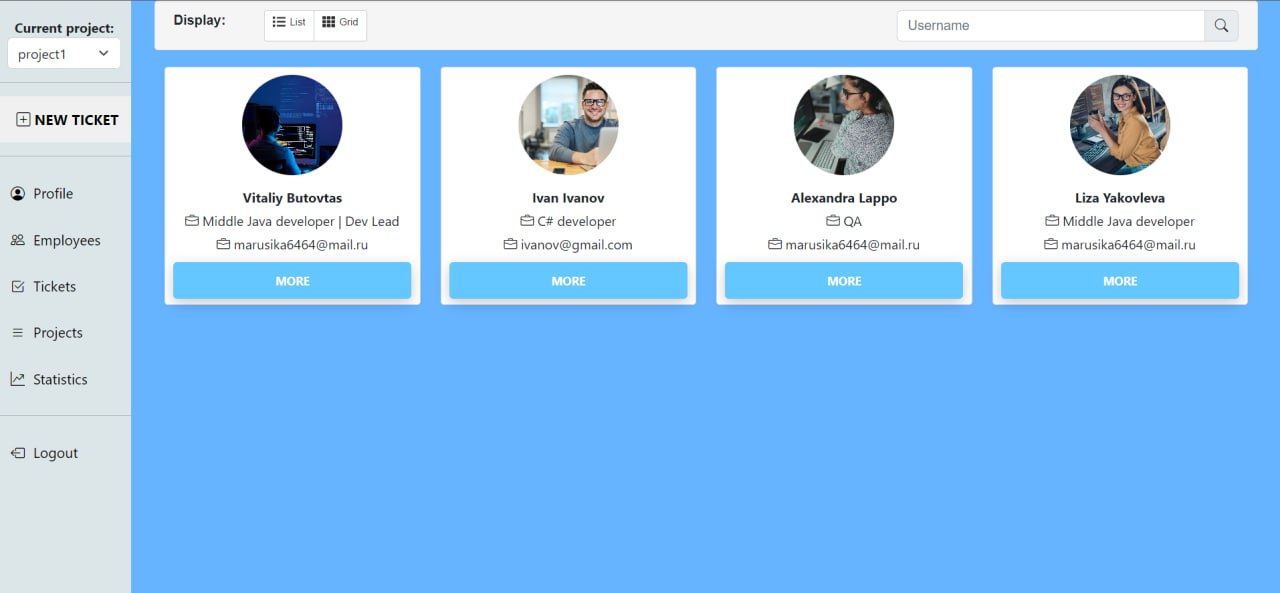


Рисунок 5.8 – Страница со всеми сотрудниками

Также есть возможность представления в виде сетки, как на рисунке 5.8, и в виде раскрывающегося списка, как на рисунке 5.9. Чтобы перейти на профиль сотрудника необходимо нажать на кнопку More.

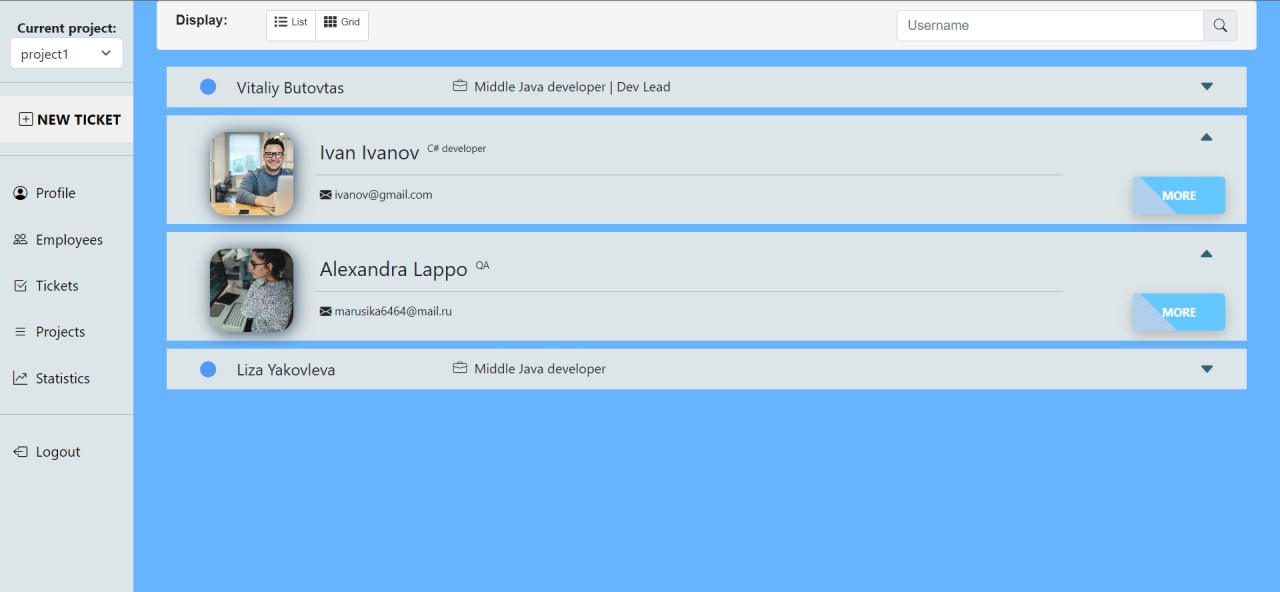


Рисунок 5.9 – Страница со всеми сотрудниками в виде списка

Следующая клавиша меню это Tickets. При нажатии на нее пользователь будет перенаправлен на страницу с дэшбордом, представленную на рисунке 5.10. Дэшборд на ней будет представлен для проекта, выбранного в меню в качестве текущего. При изменении текущего проекта в меню дэшборд будет перезагружен. В самом верху странице слева дополнительно указано имя проекта дэшборда.

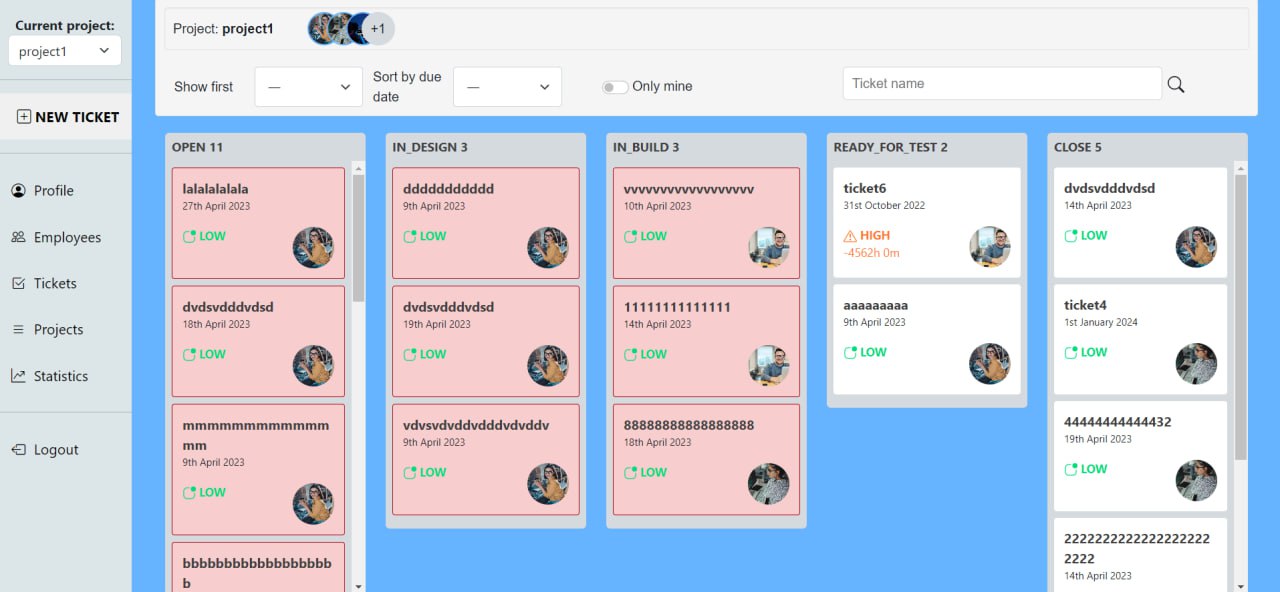


Рисунок 5.10 – Страница с дэшбордом

Справа от названия проекта изображены иконки с фотографиями нескольких сотрудников, которые работают на проекте. При нажатии на них откроет модальное окно с полноценным списком всех сотрудников, занятых на проекте. По списку можно осуществлять поиск по имени и/или фамилии. Пример такого окна будет представлен далее при рассмотрении страницы проекта.

На панель ниже представлены инструменты для произведения поиска, фильтрации и сортировки по задачам. Поиск осуществляется на основе названия задачи, с помощью фильтрация можно отобразить только те задачи, которые закреплены за вами, а сортировку можно выполнить либо по приоритету задачи, тогда задачи с выбранным статусом будут отображены первыми, либо по сроку выполнения задачи по убыванию или по возрастанию. Также в полях сортировки есть возможность вернуть порядок задачи к исходному, выбрав в них пустое значение.

В центре страницы расположены сами задачи, разделенные по их статусам. Красным цветом отмечены задачи, в которых просрочен срок выполнения. Задачи можно перемещать курсором внутри колонки, а также в другие колонки. Чтобы перейти на страницу задачи нажмите на название задачи. Пример страницы для конкретной задачи представлен на рисунке 5.11.

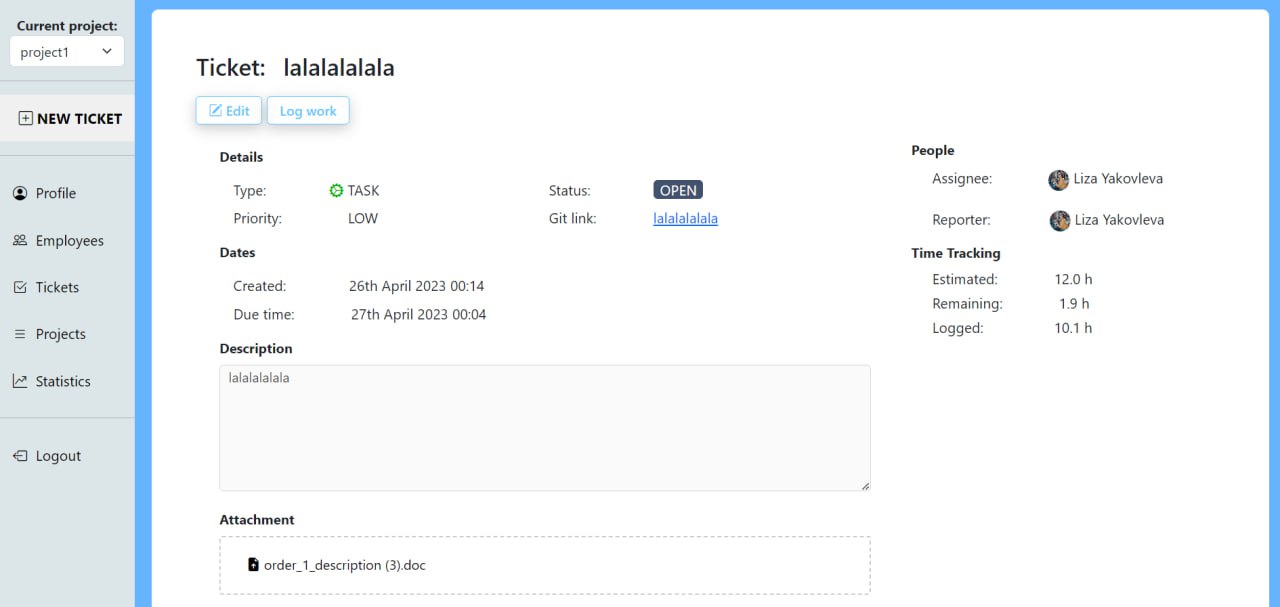


Рисунок 5.11 – Страница задачи

На страницы представлена вся детальная информация о задаче. Редактировать задачу может любой сотрудник, который работает на проекте, которому принадлежит задача, а также любой администратор. Чтобы отметить время, затраченное на выполнение задачи, то есть залогировать его, необходимо нажать на соответствующую кнопку на страницу, после чего появится модальное окно, представленное на рисунке 5.12. Здесь можно указать количество часов, потраченных на выполнение задачи в текущий день, а также изменить оценочное время на выполнение задачи. Если оценочное время будет меньше, чем уже затраченное, то на странице задачи в графе с оценочным временем будет указано затраченное время. Логирование времени могут выполнять те же лица, что и изменение задачи.

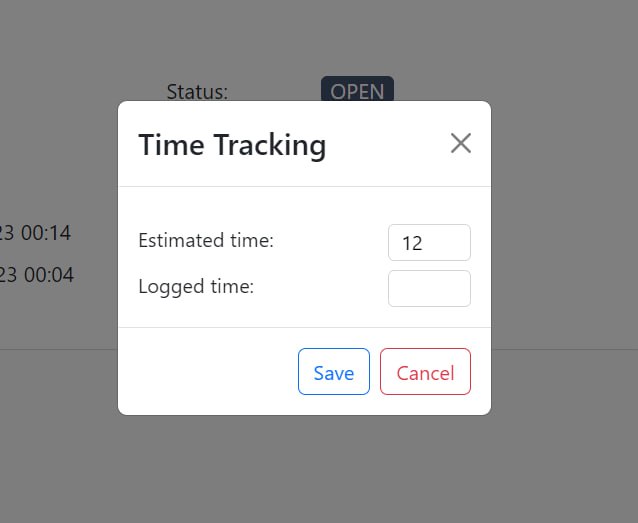


Рисунок 5.12 – Окно для логирования времени

Также под задачей можно оставлять комментарии. Они все также находятся на странице с задачей в самом ее низу и представлены на рисунке 5.13. Есть возможность сворачивать комментарии. Оставлять комментарии под задачей может любой сотрудник. Пользователи могут редактировать и удалять свои комментарии, а администратор может редактировать и удалять любые комментарии.

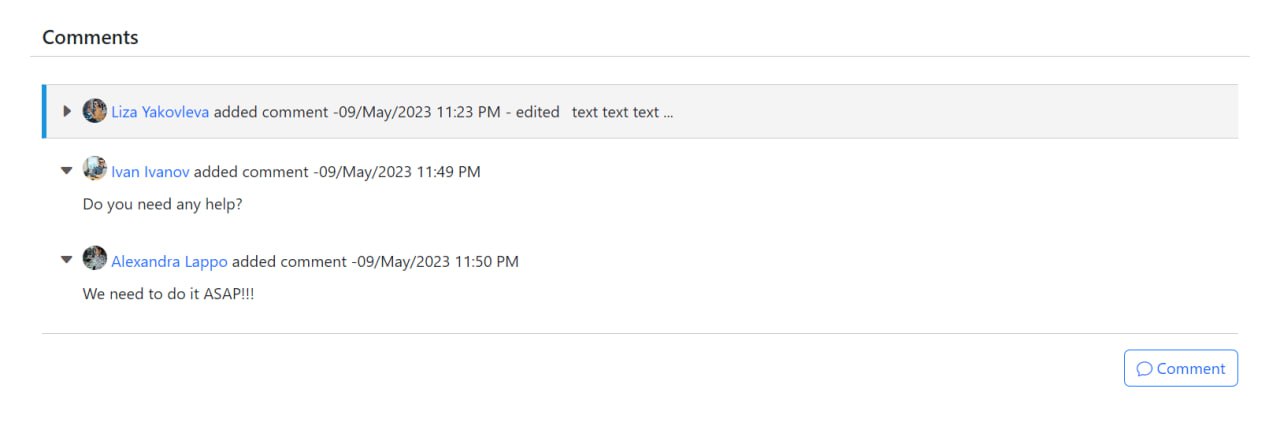


Рисунок 5.13 – Комментарии к задаче

Далее рассмотрим страницу с проектами. Она изображена на рисунке 5.14. На ней представлены все существующие проекты компании.

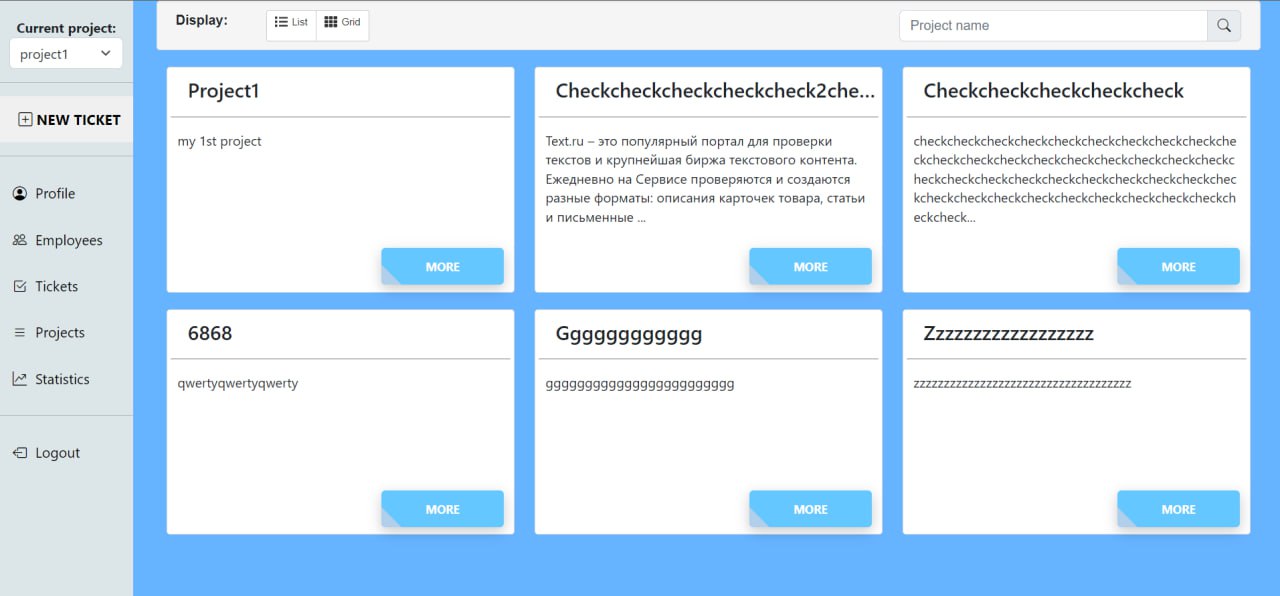


Рисунок 5.14 – Страницы со всеми проектами

Можно отображать данные в виде сетки или списка, а также осуществлять поиск по проектам по названию. Чтобы перейти на страницу проекта необходимо нажать на кнопку More. Пример страницы проекта представлен на рисунке 5.15. В целом она выглядит идентично странице задачи, за исключением того, что тут отсутствует возможность логировать время и внизу страницы отображены все задачи, относящиеся к данному проекту.

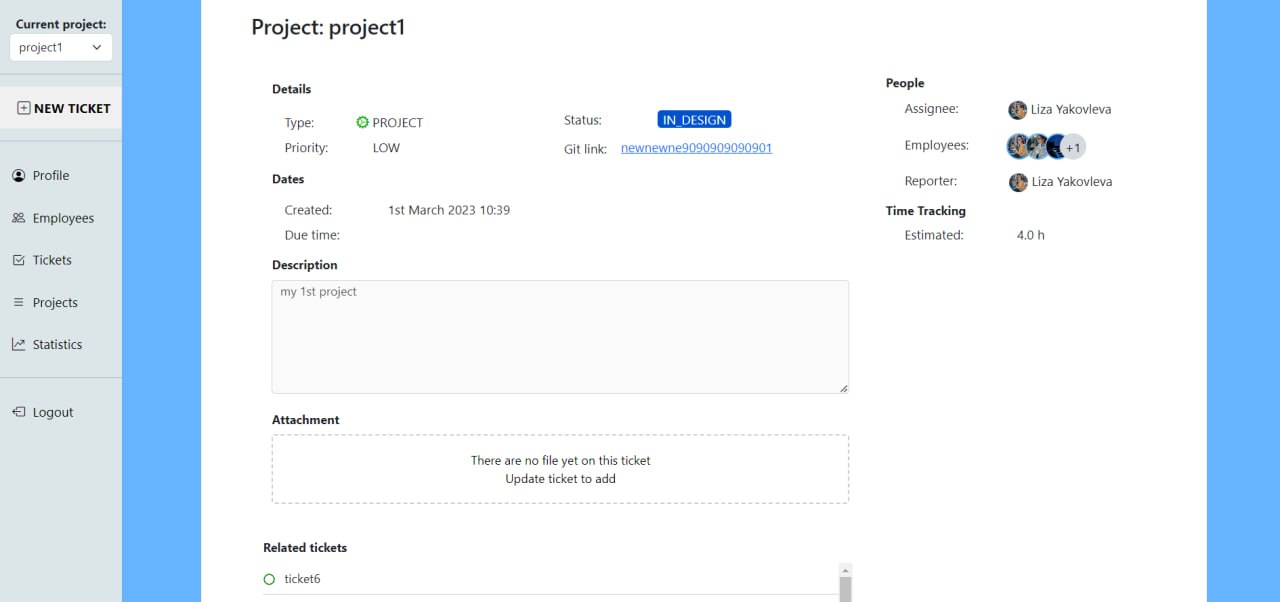


Рисунок 5.15 – Страница проекта

В правой части экрана отображено поле, отображающее сотрудников, работающих на данном проекте. При нажатии на него будет отображено модальное окно, содержащее уже полный список участников проекта, где можно осуществлять среди них поиск по имени и/или фамилии. Пример этого окна представлен на рисунке 5.16.

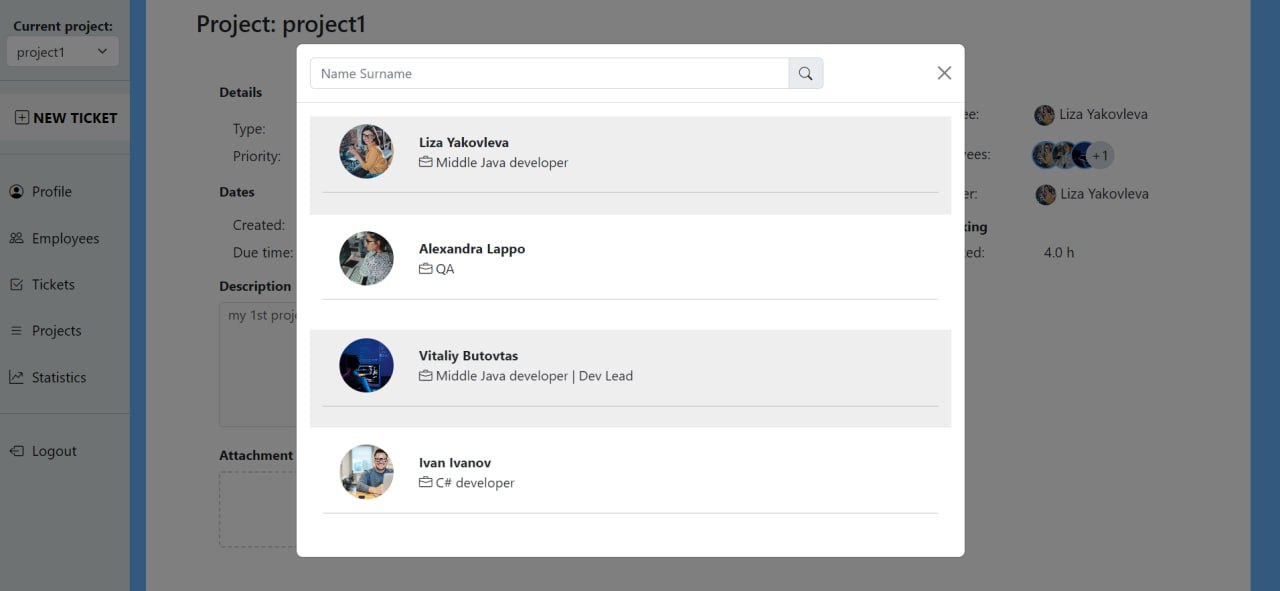


Рисунок 5.16 – Модальное окно со всеми сотрудниками проекта

Следующий пункт навигационного меню – это страница со статистикой. Для сотрудников здесь представлена как личная статистика, пример которой продемонстрирован на рисунке 5.17, так и отдельная статистика по проектам.

На странице с личной статистикой приведены следующие показатели. Слева изображена столбчатая диаграмма, отображающая количество залогированных пользователем часов по дням. Для переключения между неделями используются стрелки над диаграммой. Справа отображено сколько у пользователя на данный момент незакрытых задач, а также количество задач, у которых превышен установленный срок выполнения. Цвет данных кружков зависит от количества задач. В лучшем случае они будут зеленого цвета, в случае средней – ситуации желтого, а в большее худших вариантах – красного цвета. Под ними находится список задач, у которых вышло время завершения или закончится сегодня или завтра.

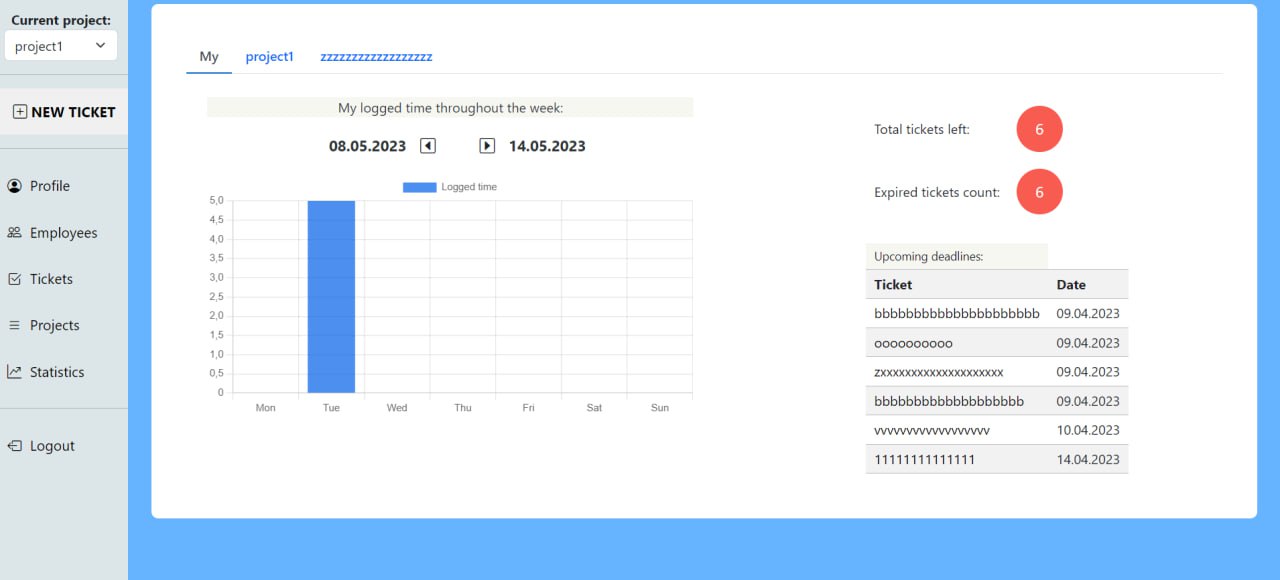


Рисунок 5.17 – Пример личной статистики сотрудника

На рисунке 5.18 представлен пример страницы со статистикой для отдельного проекта. Слева находится круговая диаграмма, отображающая количество задач по каждому статусу на данный момент. При наведении на каждую из секций будет показано точное число. Справа находится график, отображающий по дням количество завершенных задач в течение конкретной недели.



Рисунок 5.18 – Пример статистики по проекту

Крайней кнопкой в навигационном меню является выход из приложения. Она очистит все куки в браузере и направит пользователя на страницу для авторизации.

* 1. Руководство для менеджера

Менеджер – это сотрудник, но с расширенными правами. Он отвечает за работу с проектами и клиентами. Менеджер может выполнять все действия, что и сотрудник, а также специфичные действия, доступные только ему, которые рассмотрим далее.

Навигационное меню для менеджера представлено на рисунке 5.19. Как можно заметить, здесь добавились разделы Clients и Orders.

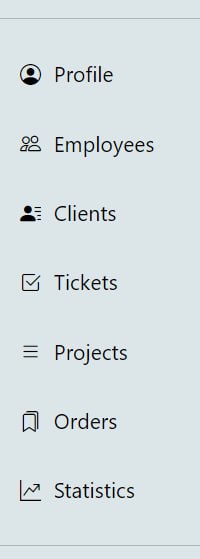


Рисунок 5.19 – Навигационное меню менеджера

При переходе на страницу Clients пользователь-менеджер может просмотреть список всех клиентов компании, а также осуществлять по ним поиск по имени и/или фамилии. Пример данной страницы можно будет просмотреть в руководстве для администратора на рисунке 5.23, но важно отметить, что для менеджера отсутствуют кнопки деактивации и блокировки пользователя.

Пример страницы Orders также представлен на рисунке 5.20. На данной странице представлены все заказы компании. Менеджер может блокировать заказы, однако после этого они не будут ему видны.



Рисунок 5.20 – Пример страницы с заказами

Если проект на основе какого-либо заказа уже существует, то можно нажать по соответствующей кнопке интерфейса и просмотреть его. Если проекта еще нет, то менеджер может создать его на основе выбранного заказа. Пример страницы для создания проекта представлен на рисунке 5.21.

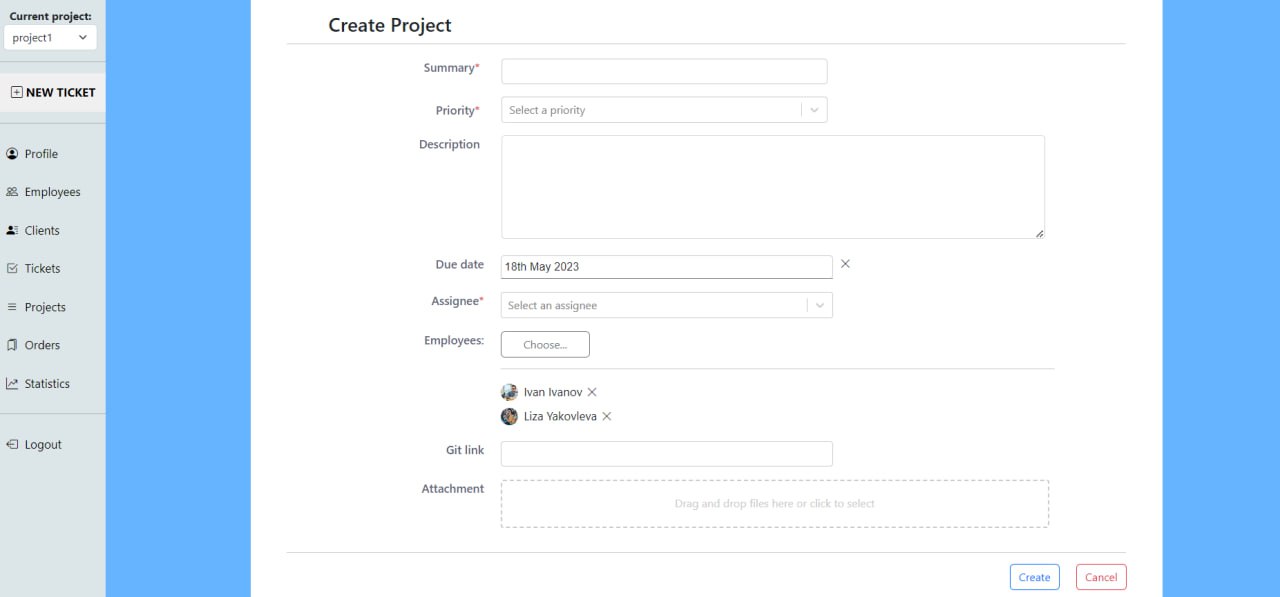


Рисунок 5.21 – Пример страницы для создания проекта

Для прикрепления сотрудников к проекту менеджер может выбрать их в специальном модальном окне, нажав на соответствующую кнопку.

Таким образом, только менеджер может создать проект. Редактировать проект может только тот менеджер, который его и создал, а также администратор.

* 1. Руководство для администратора

Администратор обладает неограниченными правами в приложении. Ему доступно все, что доступно любому сотруднику или менеджеру, а также некоторые расширенные возможности. Навигационное меню администратора идентично меню менеджера. Администратор – единственный, кто может создавать пользователей-сотрудников. Для этого необходимо перейти на страницу Users, где только для администратора будет отображена специальная кнопка для добавления пользователя. После нажатия на экране появится модальное окно, представленное на рисунке 5.22.



Рисунок 5.22 – Модальное окно для создания сотрудника

Пример отображения для администратора страницы со всеми клиентами представлен на далее рисунке 5.23.

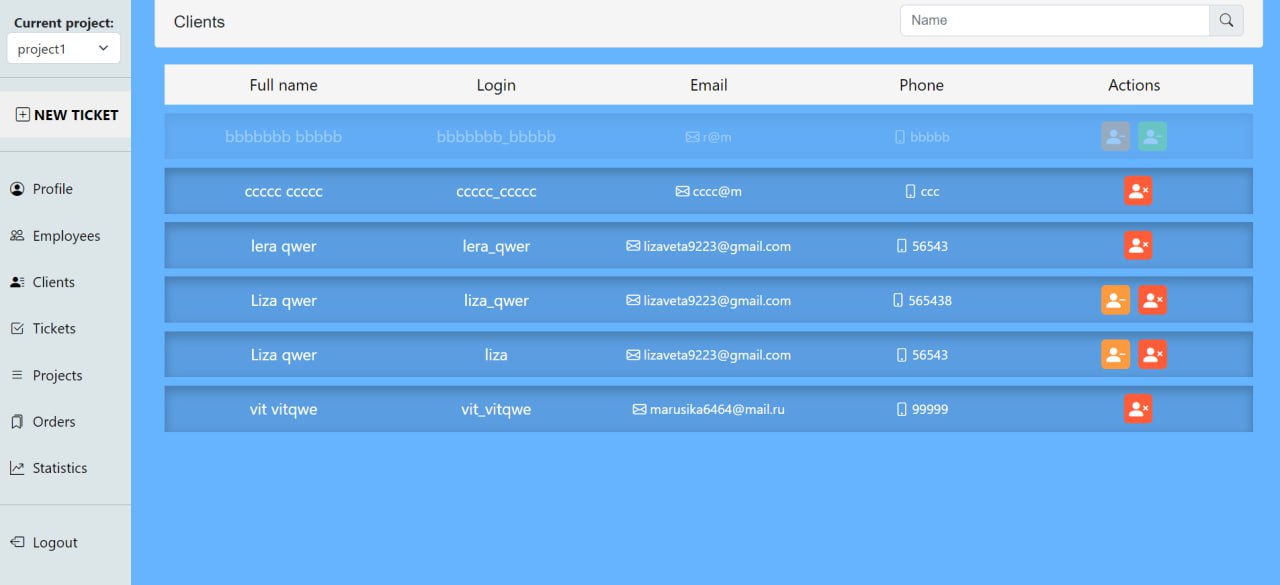


Рисунок 5.23 – Отображение для администратора страницы со всеми клиентами

Как можно увидеть, администратор обладает возможностью деактивации, блокировки и разблокировки клиентом. Аналогичное он может осуществлять и с пользователями-сотрудниками. Пример отображения страницы сотрудника для администратора представлен на рисунке 5.24. Как можно увидеть, администратору на ней доступны кнопки деактивации и блокировки.

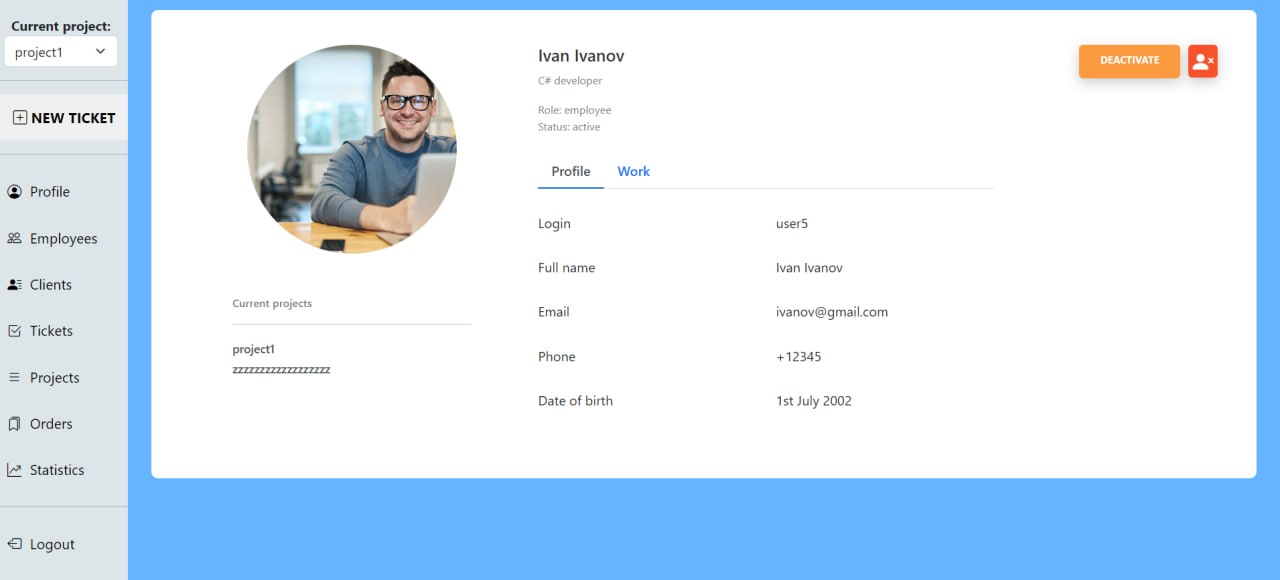


Рисунок 5.24 – Отображение страницы сотрудника для администратора

В случае деактивации пользователя, ему на электронную почту будет отправлено письмо с новым временным сгенерированным паролем, и затем при первом входе в приложение после деактивации, пользователю будет необходимо сменить временный пароль. Также администратор может видеть заблокированных сотрудников и клиентов на страницах со всеми сотрудниками/клиентами. Пример отображения представлен далее на рисунке 5.25.

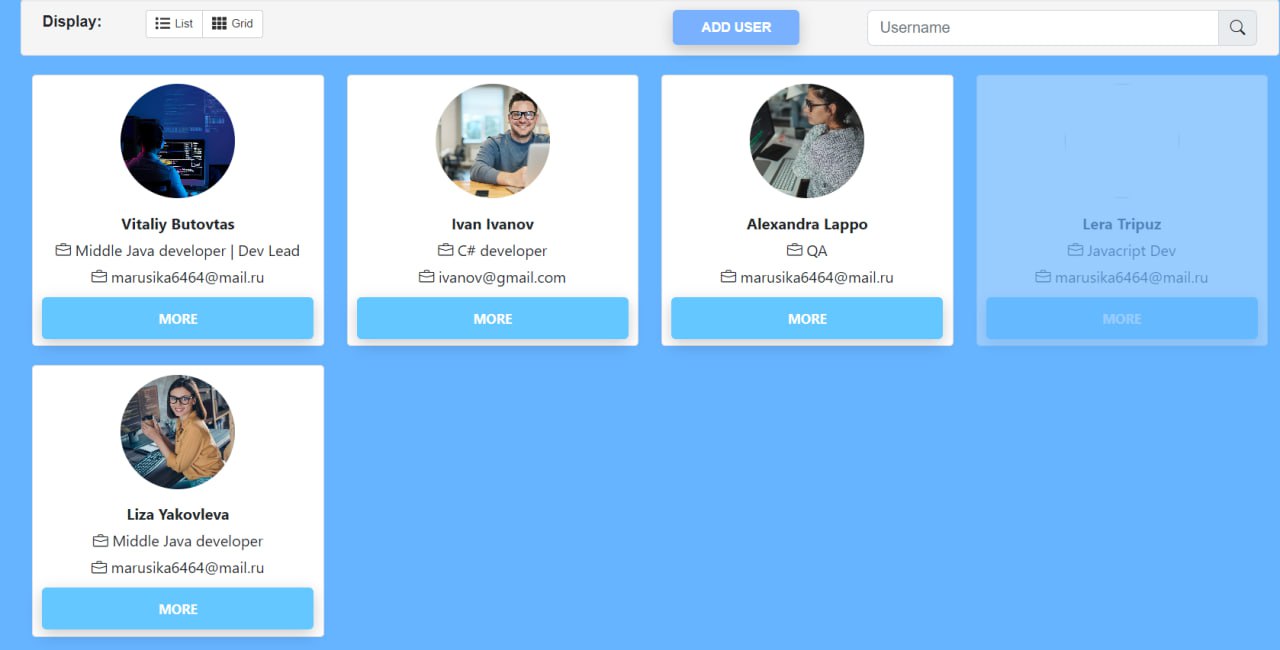


Рисунок 5.25 – Отображение заблокированных сотрудников

Как можно увидеть, заблокированные пользователи тут отображаются с приглушенным цветом. Также только администратор на странице Orders может видеть заблокированные заказы в приложении.

* 1. Руководство для клиента

Роль клиента создана для клиентов компании. Клиент может создавать заявки на выполнение проекта. Для этого ему необходимо перейти на общедоступный лендинг-сайт и заполнить там форму. Пример такой формы приведен на рисунке 5.26.



Рисунок 5.26 – Пример клиентской формы

После этого будет создана заявка клиента и автоматически будет сгенерирован профиль в приложении для него, а на почту будут высланы логин и пароль для входа. Однако пока у клиента не будет ни одного созданного проекта, у него не будет возможности войти в приложении. Профиль клиента представлен в виде модального окна, пример которого изображен на рисунке 5.27.

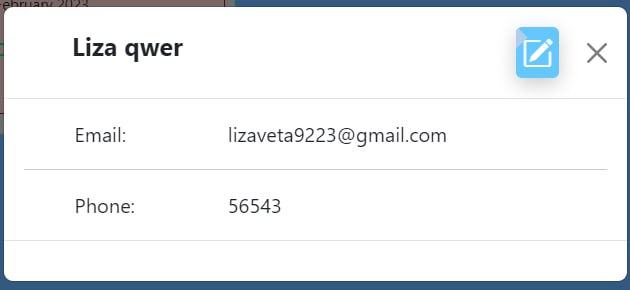


Рисунок 5.27 – Профиль пользователя-клиента

В приложении клиент может следить за выполнением своего проекта или проектов, если их несколько. Он может просматривать дэшборд по его проекту, детальную информацию по задаче этого проектам или по самому проекту. Клиент не может ничего редактировать или комментировать, а также просматривать чужие проекты или задачи, не относящиеся к его проекту. Клиент может просматривать статистику по своему проекту, однако у клиента соответственно нету личной статистики.

* 1. Выводы по разделу

Данный раздел предоставляет пояснения по работе пользователя с веб-приложением. Так как интерфейс приложения прост и интуитивно понятен, то ожидается, что у пользователя не должно возникнуть трудностей с его эксплуатацией.

Руководство содержит описание всех основных функций и возможностей приложения с приведением графических иллюстраций.

Данное приложение имеет 4 типа пользователя: сотрудник, менеджер, администратор и клиент. В данном разделе были описаны основные возможности каждой из ролей, цели создания этих ролей, а также методы их получения.

Помимо руководства по работе с приложением данный раздел включает в себя дополнительное пояснение по развертыванию приложения в среде Docker, а также инструкции по запуске на операционных системах Windows и MacOS.

Архитектуру приложения и большие функциональные возможности, предоставляемые пользователю, получилось ограничить небольшим количеством экранов, что является прямым доказательством хорошо спроектированного и простого пользовательского опыта в данном приложении. Но несмотря на это, руководство пользователя необходимо создавать всегда, независимо от сложности приложения, поскольку это не исключает возникновения недопонимания о пользователей.

1. Технико-экономическое обоснование проекта

Ф.И.О

Подпись

Дата

Лист

1

БГТУ 06.00.ПЗ

Разраб.

*Яковлева Е.И.*

Пров.

Кантарович В.С.

Н. контр.

Семенова Л. С.

Утв.

Смелов В.В.

6 Технико-экономическое обоснование проекта

Лит.

Листов

8

*74218051, 2023*

У

Консульт.

Николайчук А.Н.

* 1. Общая характеристика разрабатываемого программного средства

При выполнении данного проекта было разработано программное средство, предназначенное для управления IT-проектами. Цель приложения – это обеспечить удобный менеджмент проектов и задач внутри IT-компании для ее сотрудников, а также клиентов. В приложении сотрудники в зависимости от своей должности могут создавать проекты, прикреплять к ним сотрудников, создавать и редактировать для проектов задачи, просматривать задачи в виде Kanban-доски и осуществлять поиск и фильтрацию по ним, комментировать задачи, логировать время, потраченное на выполнение задачи, осуществлять поиск среди пользователей, осуществлять поиск среди проектов компании, просматривать и редактировать свой профиль, просматривать личную статистику и статистику по конкретному проекту. Клиенты компании могут оставлять заявки на выполнение проектов и после начала разработки проектов следить за происходящим процессом.

Программное средство создано при помощи среды разработки IntelliJ IDEA и Visual Studio Code и языков программирования для серверной части Java и для клиентской части JavaScript. Для реализации функционала были использованы фреймворк Spring и библиотека React. В качестве базы данных использовалась СУБД MySQL. Для контейнеризации приложения был использован Docker. Для тестирования серверной части использовался Postman.

Разработанное программное решение имеет следующие преимущества перед рассмотренными в главе 1 аналогичными образами:

* возможность оставлять заявки на создание проектов для заказчиков;
* наличие профилей у пользователей, которые содержат полноценную расширенную информацию о сотруднике;
* осуществление поиска среди сотрудников;
* осуществление поиска среди проектов;
* логирование времени выполнения задач;
* возможность у заказчиков следить за ходом создания проекта;
* просмотр личной статистики и отдельно по проекту.

Приложение было создано на заказ и в последующем будет передано заказчику. Отпускная цена равняется среднерыночной стоимости аналогов.

Далее в этом разделе будет произведен маркетинговый анализ, расчет затрат на всех стадиях разработки программного средства и его определена себестоимость, то есть будет произведено полноценное технико-экономическое обоснование проекта.

* 1. Исходные данные для проведения расчетов и маркетинговый анализ

Источниками исходных данных для данных расчетов выступают действующие нормативные правовые акты. Исходные данные для расчета приведены в таблице 6.1.

Таблица 6.1 – Исходные данные для расчета

|  |  |  |
| --- | --- | --- |
| Наименование показателя | Условные обозначения | Норматив |
| Численность разработчиков, чел | Чр | 1 |
| Норматив дополнительной заработной платы, % | Ндз | 15 |
| Ставка отчислений в Фонд социальной защиты населения, % | Нфсзн | 34 |
| Ставка отчислений по обязательному страхованию в БРУСП «Белгосстрах», % | Нбгс | 0,6 |
| Норматив накладных расходов, % | Нобп, обх | 55 |
| Ставка НДС, % | ННДС | 0 |
| Налог на прибыль, % | Нп | 0 |

Ставка НДС и налог на прибыль равны нулю, так как компания является резидентом ПВТ. Далее эти данные понадобятся для расчета всех необходимых параметров в процессе экономического анализа приложения.

Для оценки стоимости разработки необходимо проанализировать приложения, которые обладают аналогичным функционалом. Однако официальной информации о стоимости разработки рассмотренных в главе 1 аналогов компаниями не предоставлялось, и приблизительных данных также не найдено. Поэтому, чтобы рассчитать примерную стоимость аналогичных приложений, были использованы специализированные сайты для расчета ориентировочной стоимости программного обеспечения. Подробные данные приведены ниже в таблице 6.2.

Таблица 6.2 – Исходные данные аналогов

|  |  |  |
| --- | --- | --- |
| Источник | Стоимость, бел. руб. (без НДС) | Примечание |
| https://estimatemyapp.com/ | 15 000 | При оценке стоимостей были выбраны следующие параметры: это веб-приложение, средних габаритов, которое необходимо разработать с нуля, где пользователи могут авторизовываться, просматривать профили, просматривать дэшборд, осуществлять поиск, загружать файлы, оставлять комментарии, с наличием администраторских возможностей. |
| https://thebestapp.ru/calculator | 13 900 |
| https://digitalya.co/app-cost-calculator/estimate/ | 14 200 |

Таким образом, средняя цена разработки аналогичного продукта составляет 14366,67 белорусских рубля без учета НДС.

* 1. Обоснование цены программного средства

Широкое применение вычислительных технологий требует постоянного обновления и совершенствования ПС. Выбор эффективных проектов ПС связан с их экономической оценкой и расчетом экономического эффекта, который может определяться как для разработчика, так и для пользователя.

У разработчика экономический эффект выступает в виде чистой прибыли от реализации ПС, остающейся в распоряжении организации, а у пользователя – в виде экономии трудовых, материальных и финансовых ресурсов, получаемой за счет:

* снижение таких показателей, как трудоемкость расчетов, алгоритмизация, программирование и отладка программ;
* сокращения расходов на оплату машинного времени, а также других ресурсов на отладку программных средств;
* снижения расходов на материалы;
* оптимизации программных средств;
* улучшения показателей основной деятельности в результате использования передовых программных средств.

Стоимостная оценка программных средств у разработчиков предполагает определение затрат, что включает следующие статьи:

* заработная плата исполнителей – основная и дополнительная;
* отчисления в фонд социальной защиты населения;
* отчисления по обязательному страхованию от несчастных случаев на производстве и профессиональных заболеваний;
* прочие прямые затраты;
* накладные расходы.

На основании данных затрат рассчитывается себестоимость и отпускная цена конечного программного средства.

* + 1. Расчет затрат рабочего времени на разработку программного средства

В таблице 6.3 в укрупненном виде указаны все работы, выполненные для создания указанного в дипломной работе программного средства и количество рабочих часов, потраченных на выполнение этих работ.

Таблица 6.3 – Затраты рабочего времени на разработку ПС

|  |  |
| --- | --- |
| Содержание работ | Затраты рабочего времени, часов |
| 1 | 2 |
| Написание требований | 8 |
| Разработка дизайна интерфейса | 100 |
| Разработка архитектуры базы данных | 12 |
| Установка необходимых компонентов | 8 |

Продолжение таблицы 6.3

|  |  |
| --- | --- |
| 1 | 2 |
| Разработка архитектуры проекта | 8 |
| Разработка репозиториев серверной части | 20 |
| Разработка сервисов серверной части | 50 |
| Разработка контроллеров серверной части | 30 |
| Разработка страниц интерфейса | 80 |
| Разработка запросов между серверной и клиентской частями | 10 |
| Контейнеризация приложения | 10 |
| Программное тестирование | 20 |
| Ручное тестирование | 15 |
| Доработка после тестирования | 30 |
| Написание документации | 20 |
| **Всего** | **421** |

Итоговый результат по данной таблице будет использован далее для расчетов.

* + 1. Расчет основной заработной платы

Для определения величины основной заработной платы, было проведено исследование величин заработной платы full-stack разработчиков на Java. В итоге было установлено, что средняя месячная заработная плата на позиции junior+ составляет примерно 2100 рублей для Java full-stack разработчика, а часовая ставка составляет соответственно примерно 12,50 руб/час. Далее необходимо определить основную заработную плату исходя из трудозатрат и ЗП. Согласно таблице 6.3, проект разрабатывался одним специалистом на протяжении 421 час. Таким образом, основная заработная плата будет рассчитываться по формуле 6.1.

|  |  |  |
| --- | --- | --- |
|  | , | (6.1) |

где Соз – основная заработная плата, руб.;

Траз – время раработки (часов);

Сзп – средняя часовая ставка руб./час;

Краз – количество разработчиков, человек.

|  |  |  |
| --- | --- | --- |
|  | руб. |  |

В дальнейшем для других расчетов используется основная заработная плата, рассчитанная по указанной выше методике.

* + 1. Расчет дополнительной заработной платы

Дополнительная заработная плата представляет собой выплаты, предусмотренные законодательством о труде, включает компенсирующие выплаты (например, доплаты за работу в сверхурочное время, в государственные праздники, праздничные и выходные дни) и определяется по нормативу в процентах к основной заработной плате по формуле 6.2.

|  |  |  |  |
| --- | --- | --- | --- |
|  | , | (6.2) | (6.4) |

где Соз – основная заработная плата, руб.;

Ндз – норматив дополнительной заработной платы, %.

руб.

Таким образом, дополнительная заработная плата составляет 789,38 руб.

* + 1. Расчет отчислений в Фонд социальной защиты населения и по страхованию

Отчисления в Фонд социальной защиты населения (ФСЗН) и по обязательному страхованию от несчастных случаев на производстве и профессиональных заболеваний в БРУСП «Белгосстрах» определяются в соответствии с действующими законодательными актами по нормативу в процентном отношении к фонду основной и дополнительной зарплаты исполнителей. Отчисления в Фонд социальной защиты населения вычисляются по формуле 6.3.

|  |  |  |
| --- | --- | --- |
|  |  | (6.3) |

где – основная заработная плата, руб.;

– дополнительная заработная плата на конкретное ПС, руб.;

– норматив отчислений в Фонд социальной защиты населения, %.

Отчисления в БРУСП «Белгосстрах» вычисляются по формуле 6.4.

|  |  |  |
| --- | --- | --- |
|  |  | (6.4) |

Таким образом, общие отчисления в БРУСП «Белгосстрах» составили 36,31 руб., а в фонд социальной защиты населения – 2057,64 руб.

* + 1. Расчет суммы прочих прямых затрат

В таблице 6.4 приведены прочие затраты, которые были необходимы в ходе разработке дипломного проекта. Стоимость затрат указана за период разработки.

Таблица 6.4 – Прочие прямые затраты

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Статья затрат | | Стоимость, руб. |
| 1 | Среда разработки Intellij Idea Ultimate | | 526,53 |
| 2 | Доменное имя (.com) | | 14,5 |
| 3 | Веб-хостинг | | 19,2 |
| 4 | Редактор диаграмм Visio | | 14 |
|  | | **Всего** | **574,23** |

Таким образом, общая сумма прочих затрат составила 574,23 рубля.

* + 1. Расчет суммы накладных расходов

Сумма накладных расходов Собп,обх – произведение основной заработной платы исполнителей на конкретное программное средство Соз на норматив накладных расходов в целом по организации Нобп,обх, рассчитывается по формуле 6.5.

|  |  |  |
| --- | --- | --- |
|  |  | (6.5) |

Сумма накладных расходов составит:

= 5262,50 ⋅ 55 / 100 = 2894,38 руб*.*

Таким образом накладные расходы будут равны 2894,38 руб.

* + 1. Сумма расходов на разработку программного средства

Сумма расходов на разработку программного средства Ср определяется как сумма основной и дополнительной заработных плат исполнителей на конкретное программное средство, отчислений на социальные нужды, суммы прочих затрат и суммы накладных расходов, по формуле 6.6.

|  |  |  |
| --- | --- | --- |
|  | Ср = Соз + Сдз + Сфсзн + Сбгс + Спз + Собп,обх. | (6.6) |

Все данные необходимые для вычисления есть, поэтому можно определить сумму расходов на разработку программного средства.

Ср = 5262,50 + 789,38 + + 36,31 + 574,23 + 2894,38 = 11614,43 руб.

Сумма расходов на разработку программного средства была вычислена на основе данных, рассчитанных ранее в данном разделе, и составила 11614,43 рублей.

Поскольку расходов на реализацию в данном случае нет, они будут соответственно равны нулю, и таким образом полная себестоимость проекта равна сумме расходов на разработку программного средства, то есть 11614,43 рублей.

* + 1. Определение цены, оценка эффективности

В данном разделе будет произведен расчет прибыли, которая будет получена после продажи приложения, а также будет оценена экономическая эффективность разработки данного проекта. Это итоговый и самый важный показатель, который отобразит целесообразность создания продукта.

Расчет будет отличаться в зависимости от целей разработки и способа монетизации. В данном случае продукт разрабатывается на заказ и передается заказчику.Эффект определяем по чистой прибыли, которая будет получена организацией-разработчиком при реализации продукта.

Расчета отпускной цены будет проводится по средней рыночной цене аналогов. Прибыль от реализации программного средства рассчитывается по формуле 6.7. Среднерыночная цена аналогов составляет примерно 14366,67 руб. без НДС. Прибыль от реализации программного средства рассчитывается как разница между отпускной ценой и полной себестоимостью продукта.

|  |  |
| --- | --- |
| , | (6.7) |
| ППС = 14366,67 – 11614,43 = 2752,24 руб. | |

Итого прибыль составила 2752,24 руб. при отпускной цене 14366,67 руб. без учета НДС и себестоимости равно 11614,43 руб.

Выше рассчитанное значение является величиной чистой прибыли, так как для резидентов ПВТ ставка налога на прибыль равна нулю. Для расчета уровня рентабельности продукта Урент воспользуемся формулой 6.8.

|  |  |  |
| --- | --- | --- |
|  |  | (6.8) |

|  |
| --- |
| У рент = 2752,24⋅ 100 / 11614,43 = 23,7 % |

Рентабельность рассчитывается как частное между прибыль от реализации программного средства и полной себестоимостью продукта, умноженное на сто процентов. Таким образом, как и значение чистой прибыли, уровень рентабельность является очень важным показателем и определяет экономическую эффективность проекта. В данном случае уровень рентабельности соствил 23,7% при прибыли в 2752,24 руб. и отпускной цене 11614,43 руб.

На данный момент, анализируя текущую ситуацию на IT-рынке, можно считать приемлемым уровень рентабельности примерно от 10 до 20%.

Исходя из этого, разработанное программное средство можно считать экономически эффективным, а его разработку соответственно целесообразной, учитывая все рассчитанные в данной главе экономические показатели.

* 1. Выводы по разделу

В таблице 6.5 представлены результаты расчетов для основных показателей данной главы в краткой форме.

Таблица 6.5 – Результаты расчетов

|  |  |
| --- | --- |
| Наименование показателя | Значение |
| Время разработки, ч. | 421 |
| Количество разработчиков, чел. | 1 |
| Основная заработная плата, руб. | 5262,50 |
| Дополнительная заработная плата, руб. | 789,38 |
| Отчисления в Фонд социальной защиты населения и БРУСП «Белгосстрах», %, руб. | 2093,95 |
| Прочие прямые затраты, руб | 574,23 |
| Накладные расходы, руб | 2894,38 |
| Себестоимость разработки программного средства, руб. | 11614,43 |
| Полная себестоимость, руб. | 11614,43 |
| Чистая прибыль, руб | 2752,24 |
| Уровень рентабельности, % | 23,70 |

Поскольку величина рентабельности разработки программного средства, является положительным числом, то разработку программного продукта можно считать экономически целесообразной. Стоит отметить, что посчитать показатели прибыли и рентабельности для подобного продукта достаточно сложно, так как цены на подобные разработки не регламентируются и, наиболее оптимальным вариантом было использовать для расчетов среднюю цену на подобные разработки.

Разработка программного средства, осуществляемая одним программистом в течение двух с половиной месяцев при заданных условиях, обойдется компании в 11614,43 рублей. Реализация данного программного средства при среднерыночной цене в 14366,67 рублей принесет прибыль компании в размере суммы 2752,24 рубля. Уровень рентабельности составил 23,7%.

Благодаря этому приложению IT-компания может давать своим сотрудниками возможность эффективно и удобно заниматься управлением своих проектов.

Заключение

Ф.И.О

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

*Яковлева Е.И.*

Пров.

Кантарович В.С.

Н. контр.

Утв.

Смелов В.В.

Заключение

Лит.

Листов

1

*74218051, 2023*

У

Николайчук А.Н.

В ходе выполнения дипломного проекта были выполнены следующие изначально поставленные задачи для реализации функционала проекта:

* вся пользовательская рабочая информация cохраняется в базе данных;
* наличие возможности создания пользователей с заданными ролями;
* наличие возможности создания новых проектов;
* наличие возможности создания задач и их распределение по исполнителям;
* наличие возможности комментирования задач;
* наличие возможности просмотра личной статистики и статистики проекта.

Сначала были проанализированы схожие аналоги веб-приложения для управления IT-проектами, выявлены их положительные и отрицательные стороны. Результаты данного анализа были учтены при создании проекта.

Была спроектирована и разработана реляционная база данных для приложения, использующая экземпляр СУБД MySQL, имеющая в своем наборе 8 таблиц.

Для использования функционала созданной базы данных, был создан веб-сервис, написанный на языке программирования Java с использованием Spring Framework. Данный сервис содержит в себе все необходимые контроллеры позволяющими обслуживать пользовательские запросы, а также обеспечивает безопасность приложения благодаря авторизации пользователей на основе JWT-токенов.

Клиентская часть была разработана с использованием популярной библиотеки React, позволяющей создавать гибкую и удобную HTML-верстку, использовать различные библиотеки, а также легко взаимодействовать с AJAX-запросами.

Важным плюсом данного проекта является простота распространения, поскольку была добавлена возможность его распространения и запуска с использованием среды виртуализации Docker и Docker Registry.

Было проведено тестирование приложения на корректность работы по положительным и отрицательным сценариям, в ходе чего неисправностей выявлено не было.

Для конечных пользователей было написано руководство использования с подробным и наглядным описанием всех возможностей и функций.

Итоговым шагом было проведение экономического обоснования проекта, подтвердившее экономическую выгоду его разработки.

Таким образом, было создано веб-приложение, которое будет очень удобно для использования при менеджменте IT-проектов, а также легко для внедрения.

Список использованных источников

Ф.И.О

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

*Яковлева Е.И.*

Пров.

Кантарович В.С.

Н. контр.

Утв.

Смелов В.В.

Список использованной литературы

Лит.

Листов

2

*74218051, 2023*

У

Николайчук А.Н.

1. Trello [Электронный ресурс] / Atlassian – Режим доступа: https://trello.com/ – Дата доступа: 07.04.2023.
2. Notion [Электронный ресурс] / Notion – Режим доступа: https://www.notion.so/ – Дата доступа: 07.04.2023.
3. Jira [Электронный ресурс] / Atlassian – Режим доступа: https://www.atlassian.com/software/jira – Дата доступа: 07.04.2023.
4. Dynamic digital calendar day Kanban board: пат. 10496962 B1 США / Bryan Gregory Bartels, Patrick Humpal; заявитель Bryan Gregory Bartels, Patrick Humpal –   
   № : 15/995,156; заявл. 01.06.18; опубл. 03.12.19.
5. Kanban-based work processing device and work processing method: пат. 2022/0327486 A1 США / Jae Ho Choi; заявитель Jae Ho Choi, DELTA PDS CO. LTD – № : 17 / 333,734; заявл. 28.05.21; опубл. 13.10.22.
6. Встраиваемые системы и интернет вещей: почему надо учить Java в 2023 году [Электронный ресурс] / Synergy Times: медиа об образовании и карьере – Режим доступа: https://synergytimes.ru/evolve/vstraivaemye-sistemy-i-internet-veshchey-poche  
   mu-nado-uchit-java-v-2023-godu – Дата доступа: 08.04.2023.
7. Фреймворки в веб-разработке [Электронный ресурс] / Web creator: сложные IT-проекты и автоматизация бизнеса – Режим доступа: https://web-creator.ru/articles/about\_frameworks – Дата доступа: 10.04.2023.
8. Spring Boot 101: Введение в создание веб-приложений [Электронный ресурс] / VC.RU – Режим доступа: https://vc.ru/u/1389654-machine-learning/586955-spring-boot-101-vvedenie-v-sozdanie-veb-prilozheniy – Дата доступа: 10.04.2023.
9. Что такое реляционная база данных? [Электронный ресурс] / AWS – Режим доступа: https://aws.amazon.com/ru/relational-database/ – Дата доступа: 10.04.2023.
10. Идентификация, аутентификация и авторизация [Электронный ресурс] / Kaspersky Daily – Режим доступа: https://www.kaspersky.ru/blog/identification-authentication-authorization-difference/29123/ – Дата доступа: 10.04.2023.
11. Что такое контейнеризация? [Электронный ресурс] / Корпоративный облачный провайдер – Режим доступа: https://www.cloud4y.ru/blog/overview-of-containerization-technology/ – Дата доступа: 10.04.2023.
12. Развертывание нескольких контейнеров с помощью Docker Compose [Электронный ресурс] / Microsoft Build – Режим доступа: https://learn.microsoft.com/ru-ru/azure/cognitive-services/containers/docker-compose-recipe – Дата доступа: 12.04.2023.
13. Java Downloads [Электронный ресурс] / Oracle – Режим доступа: https://www.oracle.com/java/technologies/downloads/#jdk20-windows – Дата доступа: 17.04.2023.
14. Download IntelliJ IDEA [Электронный ресурс] / IntelliJ IDEA – Режим доступа: https://www.jetbrains.com/idea/download/#section=mac – Дата доступа: 17.04.2023.
15. Maven Repository [Электронный ресурс] – Режим доступа: https://mvnrepository.com/ – Дата доступа: 17.04.2023.
16. Что такое RESTful API? [Электронный ресурс] / AWS – Режим доступа: https://aws.amazon.com/ru/what-is/restful-api/ – Дата доступа: 19.04.2023.
17. 401 Unauthorized [Электронный ресурс] / Developer Mozilla – Режим доступа: https://developer.mozilla.org/ru/docs/Web/HTTP/Status/401 – Дата доступа: 26.04.2023.

ПРИЛОЖЕНИЕ А

Диаграмма вариантов использования

ПРИЛОЖЕНИЕ Б

Листинг классов

services:

backend:

build: ../backend/

ports:

- 8080:8080

container\_name: backend

healthcheck:

test: ["CMD", "curl", "-f", "http://backend:8080/"]

interval: 20s

timeout: 10s

retries: 10

start\_period: 60s

depends\_on:

mysql:

condition: service\_healthy

frontend:

build: ../bigproject\_UI/

ports:

- 3000:3000

container\_name: frontend

depends\_on:

backend:

condition: service\_healthy

mysql:

image: mysql:8.0.28

container\_name: mysql

volumes:

- "../backend/db.sql:/docker-entrypoint-initdb.d/1.sql"

- "../backend/DML.sql:/docker-entrypoint-initdb.d/2.sql"

- dbdata:/var/lib/mysql

environment:

MYSQL\_ROOT\_PASSWORD: system

healthcheck:

test: ["CMD", 'mysqladmin', 'ping', '-h', 'localhost', '-u', 'root', '-p$MYSQL\_ROOT\_PASSWORD']

interval: 20s

timeout: 10s

retries: 10

start\_period: 30s

volumes:

dbdata:

package com.project.integration.web.controller;

import com.project.integration.serv.services.LoggedTimeService;

import com.project.integration.web.convertor.PojoConverter;

import com.project.integration.web.formmodel.TimeForm;

import lombok.RequiredArgsConstructor;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import java.time.LocalDate;

@RestController

@RequestMapping(value = "/time")

@RequiredArgsConstructor

public class LoggedTimeController {

private final LoggedTimeService loggedTimeService;

@GetMapping(value = "/ticket/{ticketId}")

public Float getTicketLoggedTime(@PathVariable("ticketId") Integer ticketId) {

return loggedTimeService.getTicketLoggedTime(ticketId);

}

@PostMapping(value = "/{employeeId}")

public ResponseEntity<Void> updateTicketTime(@PathVariable("employeeId") Integer employeeId, @RequestBody TimeForm timeForm) {

loggedTimeService.logTime(employeeId, PojoConverter.convertTimePojoToDto(timeForm));

return new ResponseEntity<>(HttpStatus.OK);

}

@GetMapping(value = "/employee/{employeeId}")

public ResponseEntity<Float[]> getWeekLoggedTime(@PathVariable("employeeId") Integer employeeId, @RequestParam(name = "weekFirstDate") String weekFirstDate, @RequestParam(name = "weekLastDate") String weekLastDate) {

return new ResponseEntity<>(loggedTimeService.getWeekLoggedTime(

employeeId,

LocalDate.parse(weekFirstDate),

LocalDate.parse(weekLastDate)

), HttpStatus.OK);

}

}

package com.project.integration.serv.services;

import com.project.integration.dao.entity.Employee;

import com.project.integration.dao.entity.LoggedTime;

import com.project.integration.dao.entity.Ticket;

import com.project.integration.dao.repos.EmployeeRepository;

import com.project.integration.dao.repos.LoggedTimeRepository;

import com.project.integration.serv.dto.TicketDto;

import com.project.integration.serv.dto.TimeDto;

import com.project.integration.serv.exception.ServiceException;

import com.project.integration.serv.mapper.LoggedTimeMapper;

import lombok.RequiredArgsConstructor;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.stereotype.Service;

import java.sql.Date;

import java.time.LocalDate;

import java.time.LocalDateTime;

import java.util.List;

import java.util.Map;

import java.util.Objects;

import java.util.stream.Collectors;

@Service

@ComponentScan("com.project.integration.serv")

@RequiredArgsConstructor

public class LoggedTimeService {

private final LoggedTimeRepository loggedTimeRepository;

private final TicketService ticketService;

private final EmployeeRepository employeeRepository;

private final LoggedTimeMapper loggedTimeMapper;

public Float getTicketLoggedTime(Integer ticketId) {

return loggedTimeRepository.getTicketLoggedTime(ticketId);

}

public void logTime(Integer employeeId, TimeDto timeDto) {

Employee employee = employeeRepository.findById(employeeId).orElse(null);

if (Objects.isNull(employee)) {

throw new ServiceException("Employee [id=" + employeeId + "] does not exist");

}

TicketDto ticketDto = ticketService.getTicket(timeDto.getTicketId());

TicketDto updatedTicketDto = new TicketDto();

updatedTicketDto.setEstimatedTime(timeDto.getEstimatedTime());

ticketService.updateTicket(updatedTicketDto, ticketDto.getId(), false);

if (Objects.nonNull(timeDto.getLoggedTime())) {

LoggedTime loggedTime = new LoggedTime();

loggedTime.setTicket(new Ticket(timeDto.getTicketId()));

loggedTime.setEmployee(new Employee(employeeId));

loggedTime.setLoggedTime(timeDto.getLoggedTime());

loggedTime.setCreatedAt(LocalDateTime.now());

loggedTimeRepository.save(loggedTime);

}

}

public Float[] getWeekLoggedTime(Integer employeeId, LocalDate weekFirstDate, LocalDate weekLastDate) {

Map<LocalDate, Double> loggedTimeMap = loggedTimeRepository.getWeekLoggedTime(employeeId, weekFirstDate, weekLastDate)

.stream()

.collect(Collectors.toMap(s -> ((Date) s.get("date")).toLocalDate(), s -> (Double) s.get("time")));

Float[] loggedTime = new Float[7];

int idx = 0;

for (LocalDate i = weekFirstDate; weekLastDate.isAfter(i) || weekLastDate.equals(i); i = i.plusDays(1), idx++) {

Double value = loggedTimeMap.get(i);

if(Objects.nonNull(value)){

loggedTime[idx] = value.floatValue();

} else {

loggedTime[idx] = 0f;

}

}

return loggedTime;

}

}

package com.project.integration.dao.repos;

import com.project.integration.dao.entity.LoggedTime;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.data.jpa.repository.Query;

import org.springframework.stereotype.Repository;

import java.time.LocalDate;

import java.util.List;

import java.util.Map;

@Repository

@ComponentScan("com.project.integration")

public interface LoggedTimeRepository extends JpaRepository<LoggedTime, Integer> {

@Query(nativeQuery = true, value = "select sum(logged\_time.logged\_time) from logged\_time where logged\_time.ticket\_id = ?1")

Float getTicketLoggedTime(Integer ticketId);

@Query(nativeQuery = true, value = "select date(logged\_time.created\_at) as date, TRUNCATE(sum(logged\_time.logged\_time),1) as time " +

"from logged\_time where logged\_time.employee\_id = ?1 " +

"and logged\_time.created\_at >= ?2 and logged\_time.created\_at <= ?3 group by date(logged\_time.created\_at)")

List<Map<String, Object>> getWeekLoggedTime(Integer employeeId, LocalDate weekFirstDate, LocalDate weekLastDate);

}

const validateTicket = (event) => {

let isValidated = false;

if(selectedTtOption === null ||

selectedPriorityOption === null ||

selectedTtOption === null){

isValidated = true;

}

if (ticketName === '') {

isValidated = true;

setTicketNameErrMessage("Summary can't be empty");

} else if (ticketName.length > 100 ){

isValidated = true;

setTicketNameErrMessage("Summary max length is 100 symbols");

}

if((!gitLink && gitLink.length > 200) ||

!description && description.length > 1000){

isValidated = true;

}

if(isValidated){

event.stopPropagation();

setValidated(true);

}

return isValidated;

}

import React from "react";

import { Routes, Route, Navigate, useNavigate} from 'react-router-dom';

import { useCookies } from 'react-cookie';

import jwt\_decode from "jwt-decode";

import Authorization from './Authorization';

import Main from './Main';

import 'bootstrap/dist/css/bootstrap.min.css';

const isUserLoggedIn = (token) => {

if (token==null || token==undefined || token ==="undefined" || token === '') {

return false;

} else{

try{

let decodedToken = jwt\_decode(token);

if (decodedToken.exp > new Date().getTime()) {

return false;

}

else return true;

} catch(error){

return false;

}

}

};

const ProtectedRoute = ({ redirectPath = '/login', children }) => {

const [cookies] = useCookies(["token"]);

document.cookie = "url=" + window.location.pathname + "; path=/";

if(isUserLoggedIn(cookies.token)) return children;

else{

document.cookie = "token=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/";

document.cookie = "project=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/";

document.cookie = "employeeId=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/";

return <Navigate to={redirectPath} replace />;

}

};

const GoToLogin = ({ children }) => {

const [cookies] = useCookies(["token", "url"]);

if(isUserLoggedIn(cookies.token))

return <Navigate to={cookies.url} replace />;

else return children;

};

function App(){

const navigate = useNavigate();

const [cookies] = useCookies(["token"]);

return(

<Routes>

<Route path="/\*" element={<Navigate to={isUserLoggedIn(cookies.token) ? "/app/\*" : "/login"} />} />

<Route path="/login"

element={

<GoToLogin>

<Authorization navigate={navigate} />

</GoToLogin>

}/>

<Route path="/app/\*"

element={

<ProtectedRoute>

<Main navigate={navigate} />

</ProtectedRoute>

}/>

</Routes>

);

}

const OnlyManagerAdminRoute = ({ redirectPath = '/', children }) => {

const [cookies] = useCookies(["token"]);

let decodedToken = jwt\_decode(cookies.token);

if(decodedToken.role==="ROLE\_MANAGER" || decodedToken.role==="ROLE\_ADMIN") return children;

else return <Navigate to={redirectPath} replace />;

};

export default function Main(){

return(

<div className="main">

<Routes>

<Route path="/\*" element={<Navigate to={decodedToken.role!==" CUSTOMER" ? "profile/" + cookies.employeeId : "dashboard"} />} />

<Route path="profile/:id" element={<Profile navigate={navigate} /> } />

{

<Route path="dashboard" element={<Dashboard navigate={navigate} listenCookieChange={listenCookieChange} /> } />

}

<Route exact path="ticket/:id" element={<SingleTask navigate={navigate} /> }

<Route path="projects" element={<Projects navigate={navigate} /> } />

<Route path="users" element={<Users navigate={navigate} /> } />

<Route path="project/:id" element={<ProjectPage navigate={navigate} /> } />

<Route path="statistics" element={<Statistics navigate={navigate} /> } />

<Route path="orders" element={

<OnlyManagerAdminRoute>

<Orders navigate={navigate} />

</OnlyManagerAdminRoute>

}/>

<Route path="orders/:id/create" element={

<OnlyManagerAdminRoute>

<CreateProject navigate={navigate} />

</OnlyManagerAdminRoute>

}/>

<Route path="clients" element={

<OnlyManagerAdminRoute>

<Clients navigate={navigate} />

</OnlyManagerAdminRoute>

}/>

</Routes>

</div>

);}

ПРИЛОЖЕНИЕ В

Логическая схема базы данных

ПРИЛОЖЕНИЕ Г

Блок-схема алгоритма перемещения задачи на «дэшборде»

ПРИЛОЖЕНИЕ Д

Блок-схема алгоритма изменения прикрепленного к задаче файла

ПРИЛОЖЕНИЕ Е

Блок-схема алгоритма авторизации заказчика в приложении

ПРИЛОЖЕНИЕ Ж

Блок-схема алгоритма изменения пароля от аккаунта

ПРИЛОЖЕНИЕ И

Структурная схема архитектуры приложения

ПРИЛОЖЕНИЕ К

Листинг классов SecurityConfig и JwtFilter

package com.project.integration.web.security;

import com.project.integration.serv.services.UserService;

import com.project.integration.web.exception.CustomAccessDeniedHandler;

import com.project.integration.web.exception.CustomAuthenticationEntryPoint;

import com.project.integration.web.security.jwt.JwtFilter;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.http.HttpMethod;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.builders.WebSecurity;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

import org.springframework.security.config.http.SessionCreationPolicy;

import org.springframework.security.crypto.password.PasswordEncoder;

import org.springframework.security.web.access.AccessDeniedHandler;

import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

@Configuration

@EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter {

private final JwtFilter jwtFilter;

private final UserService userDetailsService;

private final PasswordEncoder passwordEncoder;

private final CustomAuthenticationEntryPoint unauthorizedHandler;

@Autowired

public SecurityConfig(

JwtFilter jwtFilter,

UserService userDetailsService,

PasswordEncoder passwordEncoder,

CustomAuthenticationEntryPoint unauthorizedHandler) {

this.jwtFilter = jwtFilter;

this.userDetailsService = userDetailsService;

this.passwordEncoder = passwordEncoder;

this.unauthorizedHandler = unauthorizedHandler;

}

@Bean

public AuthenticationManager authenticationManagerBean() throws Exception {

return super.authenticationManagerBean();

}

@Bean

public AccessDeniedHandler accessDeniedHandler() {

return new CustomAccessDeniedHandler();

}

@Override

public void configure(AuthenticationManagerBuilder auth) throws Exception {

auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder);

}

@Override

protected void configure(HttpSecurity http) throws Exception {

http.cors().and().httpBasic().disable().csrf().disable();

http.authorizeRequests()

.antMatchers("/", "/create\_request", "/auth", "/css/\*\*", "/js/\*\*")

.permitAll()

.antMatchers("/employee/new","/orders/{\\d+}/block", "/user/{\\d+}/block", "/user/{\\d+}/deactivate")

.hasRole("ADMIN")

.antMatchers( "/orders/all", "/orders/{\\d+}/description", "/project/create/{\\d+}")

.hasAnyRole("ADMIN", "MANAGER")

.antMatchers(HttpMethod.PUT, "/project/tickets/{\\d+}")

.hasAnyRole("ADMIN", "MANAGER", "EMPLOYEE")

.antMatchers(HttpMethod.POST, "/project/{\\d+}/tickets")

.hasAnyRole("ADMIN", "MANAGER", "EMPLOYEE")

.antMatchers("/project/reorder")

.hasAnyRole("ADMIN", "MANAGER", "EMPLOYEE")

.anyRequest()

.authenticated()

.and();

http.exceptionHandling()

.authenticationEntryPoint(unauthorizedHandler)

.accessDeniedHandler(accessDeniedHandler())

.and();

http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);

http.addFilterBefore(jwtFilter, UsernamePasswordAuthenticationFilter.class);

}

}

package com.project.integration.web.security.jwt;

@Component

@RequiredArgsConstructor

public class JwtFilter extends OncePerRequestFilter {

private final JwtProvider jwtProvider;

@Override

protected boolean shouldNotFilter(HttpServletRequest request) {

return request.getRequestURI().equals("/")

|| request.getRequestURI().equals("/create\_request")

|| request.getRequestURI().equals("/auth")

|| request.getRequestURI().equals("/css/landing.css")

|| request.getRequestURI().equals("/js/landing.js"); }

@Override

protected void doFilterInternal( HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)

throws ServletException, IOException {

String token = getTokenFromRequest(request);

String requestURL = request.getRequestURL().toString();

try {

if (StringUtils.hasText(token) && jwtProvider.validateToken(token)) {

if (requestURL.contains(URL.REFRESH))

throw new JwtException("Token is not expired yet");

UserDetails userDetails =

new UserDetailsImpl(

jwtProvider.getIdFromToken(token),

jwtProvider.getLoginFromToken(token),

Strings.EMPTY,

jwtProvider.getRolesFromToken(token),

jwtProvider.getStatusFromToken(token));

UsernamePasswordAuthenticationToken auth =

new UsernamePasswordAuthenticationToken(

userDetails, null, userDetails.getAuthorities());

SecurityContextHolder.getContext().setAuthentication(auth);

filterChain.doFilter(request, response);

return;

}

} catch (ExpiredJwtException e) {

if (requestURL.contains(URL.REFRESH)) {

if (jwtProvider.isRefreshAvailable(e)) allowForRefreshToken(e, request); else throw new JwtException("Token is not available for refresh yet"); } }

response.setStatus(HttpStatus.UNAUTHORIZED.value()); }

private String getTokenFromRequest(HttpServletRequest request) {

String bearerToken = request.getHeader(Attributes.AUTHORIZATION);

if (!StringUtils.hasText(bearerToken)) {

Cookie token = WebUtils.getCookie(request, Attributes.TOKEN);

if (Objects.nonNull(token)) bearerToken = "Bearer " + token.getValue(); }

if (StringUtils.hasText(bearerToken)

&& bearerToken.startsWith(Attributes.TOKEN\_BEGINNING\_IN\_HEADER)){

return bearerToken.replace(Attributes.TOKEN\_BEGINNING\_IN\_HEADER, Strings.EMPTY); } return Strings.EMPTY; } }