МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждения образования «БЕЛОРУССКИЙ

ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Программной инженерии

Специальность 1-40 01 01 Программное обеспечение информационных технологий

Специализация 1-40 01 01 10 Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к дипломному проекту:**

Веб-приложение для организации взаимоотношений с клиентами

Дипломник ­\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Левша Марк Сергеевич

Руководитель проекта ­ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Гончар Е. А., асс.

Заведующий кафедрой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Смелов В. В., к.т.н., доцент

Консультант \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Евлаш А. И., ст. преп.

Нормоконтролеры: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Нистюк О. А., ст. преп.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Север А. С., преп.-стаж.

Дипломный проект защищен с оценкой

Председатель ГЭК \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Садов В. С., к.т.н., доцент

Минск 2024

ЛИСТ ЗАДАНИЯ

Реферат

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

Левша М.С.

Провер.

Гончар Е.А.

Н. контр.

Нистюк О.А.

Утв.

Смелов В.В.

Реферат

Лит.

Листов

1

74417043, 2024

Пояснительная записка курсового проекта содержит 66 страниц пояснительной записки, 31 иллюстрация, 15 листингов, 17 источников литературы, 10 приложений.

NODE.JS, NEST, JAVASCRIPT, TYPESCRIPT, REACT, GRAPHQL, APOLLO CLIENT, REDUX, MONGODB, CLASS-VALIDATOR, JWT

Целью дипломного проекта является разработка веб-приложения, которое бы помогло организовывать взаимоотношения с клиентами.

Пояснительная записка дипломного проекта состоит из реферата, содержания, введения, шести разделов, заключения и списка используемых источников.

В первом разделе проводится постановка задачи, обзор аналогов приложения, описание достоинств и недостатков каждого из них, проводится патентный поиск.

Во втором разделе изложено проектирование веб-приложения: обоснован выбор средств разработки, описана структура базы данных, архитектура серверной части веб-приложения, приведены диаграмма вариантов использования и диаграмма развертывания приложения, описана блок-схема алгоритма создания задачи.

В третьем разделе описывается структура проекта серверной и клиентской части веб-приложения, а также их разработка, приведены листинги с реализацией тех или иных частей приложения.

В четвертом разделе описывается проведенное unit тестирование для серверной части веб-приложения, а также проведенное ручное тестирование.

В пятом разделе изложено руководство пользователя по эксплуатации разработанного веб-приложения, описан принцип работы со страницами веб-приложения.

В шестом разделе описано выполненное технико-экономическое обоснование проекта: определена полная себестоимость проекта, вычислена чистая прибыль от продажи, определена рентабельность приложения.

В заключении приведены результаты проделанной работы, описан функционал получившегося веб-приложения.

В графической части представлено 4 чертежа А3 и 2 плаката А4.

Abstract

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

Левша М.С.

Провер.

Гончар Е.А.

Н. контр.

Нистюк О.А.

Утв.

Смелов В.В.

Abstract

Лит.

Листов

1

74417043, 2024

The explanatory note of the course project contains 66 pages of explanatory note, 31 illustrations, 15 listings, 17 literature sources, 10 appendices.

NODE.JS, NEST, JAVASCRIPT, TYPESCRIPT, REACT, GRAPHQL, APOLLO CLIENT, REDUX, MONGODB, CLASS-VALIDATOR, JWT

The goal of the thesis project is to develop a web application that would help organize relationships with clients.

The explanatory note of the thesis project consists of an abstract, content, introduction, six sections, conclusion and a list of sources used.

The first section contains a statement of the problem, a review of analogues of the application, a description of the advantages and disadvantages of each of them, and a patent search.

The second section describes the design of a web application: the choice of development tools is justified, the structure of the database is described, the architecture of the server part of the web application is presented, a diagram of use cases and a diagram of deployment of the application are presented, and a flowchart of the algorithm for creating a task is described.

The third section describes the project structure of the server and client parts of the web application, as well as their development, and provides listings with the implementation of certain parts of the application.

The fourth section describes the unit testing carried out for the server part of the web application, as well as the manual testing carried out.

The fifth section provides a user guide for operating the developed web application and describes the principle of working with web application pages.

The sixth section describes the completed feasibility study of the project: the total cost of the project was determined, the net profit from the sale was calculated, and the profitability of the application was determined.

In conclusion, the results of the work done are presented and the functionality of the resulting web application is described.

The graphic part contains 4 А3 drawings and 2 А4 posters.

**Содержание**

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

Левша М.С.

Провер.

Гончар Е.А.

Н. контр.

Нистюк О.А.

Утв.

Смелов В.В.

Содержание

Лит.

Листов

2

74417043, 2024

[Введение 7](#_Toc168446888)

[1 Обзор аналогичных решений и постановка задачи 8](#_Toc168446889)

[1.1 Анализ аналогов 8](#_Toc168446890)

[1.1.1 Веб-сервис Twenty 8](#_Toc168446891)

[1.1.2 Система Bitrix24 10](#_Toc168446892)

[1.1.3 Система ClickUp 12](#_Toc168446893)

[1.2 Патентный поиск 15](#_Toc168446894)

[1.3 Постановка задачи 15](#_Toc168446895)

[1.4 Выводы по разделу 16](#_Toc168446896)

[2 Проектирование веб-приложения 17](#_Toc168446897)

[2.1 Используемые средства разработки 17](#_Toc168446898)

[2.1.1 Visual Studio Code 17](#_Toc168446899)

[2.1.2 MongoDB Compass 17](#_Toc168446900)

[2.1.3 Postman 17](#_Toc168446901)

[2.1.4 Fork 17](#_Toc168446902)

[2.2 Используемые языки, библиотеки, сервисы 18](#_Toc168446903)

[2.2.1 JavaScript и Node.js 18](#_Toc168446904)

[2.2.2 Typescript 18](#_Toc168446905)

[2.2.3 Фреймворк Nest.js 18](#_Toc168446906)

[2.2.4 База данных MongoDB 19](#_Toc168446907)

[2.2.5 Библиотека Mongoose 19](#_Toc168446908)

[2.2.6 Библиотека Passport.js 20](#_Toc168446909)

[2.2.7 Библиотека React.js 20](#_Toc168446910)

[2.3 Проектирование архитектуры базы данных 21](#_Toc168446911)

[2.4 Проектирование структуры сервера 23](#_Toc168446912)

[2.5 Диаграмма развертывания приложения 24](#_Toc168446913)

[2.6 Диаграмма вариантов использования 25](#_Toc168446914)

[2.7 Алгоритм создания задачи 26](#_Toc168446915)

[2.8 Выводы по разделу 27](#_Toc168446916)

[3 Разработка веб-приложения 28](#_Toc168446917)

[3.1 Реализация серверной части 28](#_Toc168446918)

[3.1.1 Структура проекта 28](#_Toc168446919)

[3.1.2 Реализация аутентификации 30](#_Toc168446920)

[3.1.3 Реализация ресурса user 31](#_Toc168446921)

[3.1.4 Реализация ресурса task 34](#_Toc168446922)

[3.1.5 Реализация взаимодействия с базой данных 37](#_Toc168446923)

[3.2 Реализация клиентской части 38](#_Toc168446924)

[3.2.1 Структура проекта 38](#_Toc168446925)

[3.2.2 Реализация компонента App 40](#_Toc168446926)

[3.2.3 Реализация навигационной панели 41](#_Toc168446927)

[3.2.4 Реализация страницы задач 42](#_Toc168446928)

[3.2.5 Реализация компонента GraphQLProvider 42](#_Toc168446929)

[3.3 Выводы по разделу 43](#_Toc168446930)

[4 Тестирование веб-приложения 44](#_Toc168446931)

[4.1 Тестирование серверной части 44](#_Toc168446932)

[4.1.1 Unit-тестирование 44](#_Toc168446933)

[4.2 Ручное тестирование 47](#_Toc168446934)

[4.3 Выводы по разделу 48](#_Toc168446935)

[5 Руководство пользователя 49](#_Toc168446936)

[5.1 Роль «Администратор» 49](#_Toc168446937)

[5.2 Роль «Клиент» 53](#_Toc168446938)

[5.3 Роль «Разработчик» 54](#_Toc168446939)

[5.4 Роль «Менеджер» 55](#_Toc168446940)

[5.5 Выводы по разделу 56](#_Toc168446941)

[6 Технико-экономическое обоснование проекта 57](#_Toc168446942)

[6.1 Общая характеристика разрабатываемого программного средства 57](#_Toc168446943)

[6.2 Исходные данные для проведения расчетов и маркетинговый анализ 57](#_Toc168446944)

[6.3 Обоснование цены программного средства 58](#_Toc168446945)

[6.3.1 Расчет затрат рабочего времени на разработку программного средства 58](#_Toc168446946)

[6.3.2 Расчет основной заработной платы 59](#_Toc168446947)

[6.3.3 Расчет дополнительной заработной платы 60](#_Toc168446948)

[6.3.4 Расчет отчислений в Фонд социальной защиты населения и по обязательному страхованию 60](#_Toc168446949)

[6.3.5 Расчет суммы прочих прямых затрат 61](#_Toc168446950)

[6.3.6 Расчет суммы накладных расходов 61](#_Toc168446951)

[6.3.7 Сумма расходов на разработку программного средства 61](#_Toc168446952)

[6.3.8 Расходы на сопровождение и адаптацию 62](#_Toc168446953)

[6.3.9 Полная себестоимость 62](#_Toc168446954)

[6.3.10 Определение цены, оценка эффективности 62](#_Toc168446955)

[6.4 Выводы по разделу 63](#_Toc168446956)

[Заключение 65](#_Toc168446957)

[Список используемых источников 66](#_Toc168446958)

[ПРИЛОЖЕНИЕ А. Логическая схема базы данных 67](#_Toc168446959)

[ПРИЛОЖЕНИЕ Б. Диаграмма развертывания 68](#_Toc168446960)

[ПРИЛОЖЕНИЕ В. Диаграмма вариантов использования 69](#_Toc168446961)

[ПРИЛОЖЕНИЕ Г. Блок-Схема 70](#_Toc168446962)

[ПРИЛОЖЕНИЕ Д. Скриншот работы приложения 71](#_Toc168446963)

[ПРИЛОЖЕНИЕ Е. Таблица экономических расчетов 72](#_Toc168446964)

[ПРИЛОЖЕНИЕ Ж. Реализация класса TaskService 73](#_Toc168446965)

[ПРИЛОЖЕНИЕ И. Реализация класса MongoGenericRepository 76](#_Toc168446966)

[ПРИЛОЖЕНИЕ К. Реализация компонента TasksView 78](#_Toc168446967)

[ПРИЛОЖЕНИЕ Л. Реализация компонента GraphQLProvider 84](#_Toc168446968)

# Введение

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

Левша М.С.

Провер.

Гончар Е.А.

Н. контр.

Нистюк О.А.

Утв.

Смелов В.В.

Введение

Лит.

Листов

1

74417043, 2024

Современная бизнес-среда характеризуется высокой динамикой, растущими требованиями клиентов и жесткой конкуренцией. В таких условиях компании постоянно сталкиваются с необходимостью быстро реагировать на изменения, оперативно внедрять новые решения и эффективно управлять своими проектами. Однако зачастую этот процесс усложняется из-за разрозненности информации, трудностей в коммуникации между заинтересованными сторонами и отсутствия прозрачности в ходе выполнения поставленных задач.

Ключевой проблемой, с которой сталкиваются многие организации, является отсутствие единой, интегрированной платформы для управления проектами. Традиционные подходы, основанные на использовании разрозненных инструментов и электронных таблиц, не обеспечивают необходимого уровня согласованности и видимости всех аспектов проектной деятельности. Менеджеры тратят значительное время на сбор данных, коммуникацию с командами и попытки контроля над ходом работ, что снижает их эффективность и замедляет принятие важных решений.

Кроме того, недостаток прозрачности в управлении проектами приводит к проблемам с согласованием ожиданий клиентов и реальными результатами, а также усложняет оценку эффективности использования ресурсов. Это влечет за собой риски неудовлетворенности заказчиков, срывов сроков и перерасхода бюджетов.

Перед современными компаниями стоит острая необходимость в разработке комплексного решения, которое позволит эффективно управлять проектами, обеспечивая прозрачность, согласованность и высокую степень вовлеченности всех участников. Эффективное управление проектами является ключевым фактором успеха современных организаций, стремящихся к непрерывному развитию и росту.

Таким образом, целью дипломного проектирования является разработка веб-приложения, которое способно эффективно выстроить коммуникацию между всеми участниками жизненного цикла проекта. Для достижения поставленной цели необходимо выполнить следующие задачи:

* провести обзор аналогов приложения, сформировать требования к проекту;
* спроектировать архитектуру приложения и структуру базы данных;
* разработать приложение;
* протестировать приложение;
* написать руководство пользователя;
* провести технико-экономическое обоснование проекта.

# 1 Обзор аналогичных решений и постановка задачи

ФИО

Подпись

Дата

Лист

1

БГТУ 01.00.ПЗ

Разраб.

Левша М.С.

Провер.

Гончар Е.А.

\

Н. контр.

Нистюк О.А.

Утв.

Смелов В.В.

1 Обзор аналогичных решений и постановка задачи

Лит.

Листов

9

74417043, 2024

## **1.1 Анализ аналогов**

Для создания нового веб-приложения, используемого для организации взаимоотношений с клиентами необходимо найти и проанализировать уже существующие решения схожей направленности.

В качестве аналогов были выбраны сервисы Twenty, Bitrix24 и ClickUp, т.к. они являются наиболее популярными сервисами по теме дипломного проекта. Краткий обзор этих сервисов представлен ниже.

### **1.1.1 Веб-сервис Twenty**

Twenty является открытой CRM системой, которая позиционируется как современная альтернатива Salesforce, но с открытым исходным кодом и поддержкой сообщества в улучшении продукта [1]. Эта система была разработана с целью предоставить пользователям полный контроль над своими данными и бизнес-процессами, отказываясь от замкнутых экосистем.

Навигационное меню с функционалом сервиса представлено на рисунке 1.1.

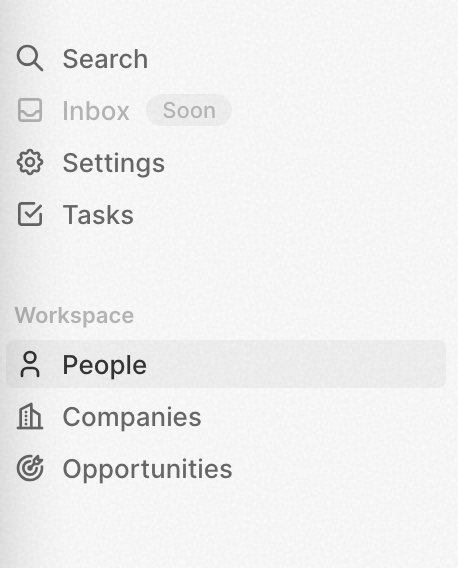


Рисунок 1.1 – Навигационное меню Twenty

Навигационная панель в CRM системе Twenty разработана с учетом удобства и эффективности работы пользователя. Она включает в себя различные функции, которые позволяют пользователям быстро и легко перемещаться по системе и выполнять необходимые задачи.

Страница с клиентами сервиса Twenty представлена на рисунке 1.2.

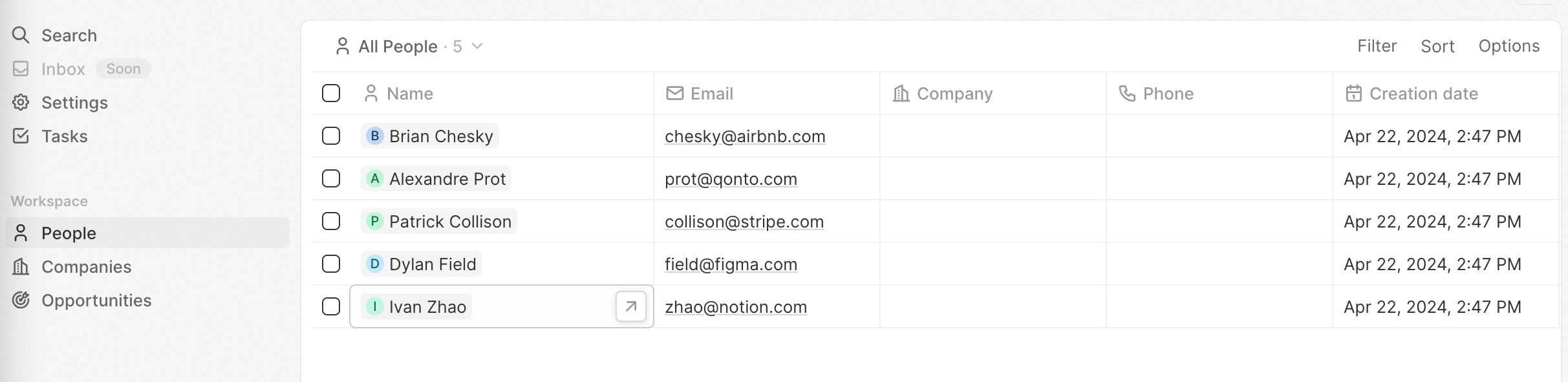


Рисунок 1.2 – Страница с клиентами сервиса Twenty

На данной странице отображаются клиенты и их контактные данные. Имеется возможность добавить нового клиента, изменить уже существующего и осуществлять удобный поиск клиентов с помощью фильтрации и сортировки.

На странице компаний представлен список партнеров, с которыми сотрудничает компания. Страница компаний представлена на рисунке 1.3.

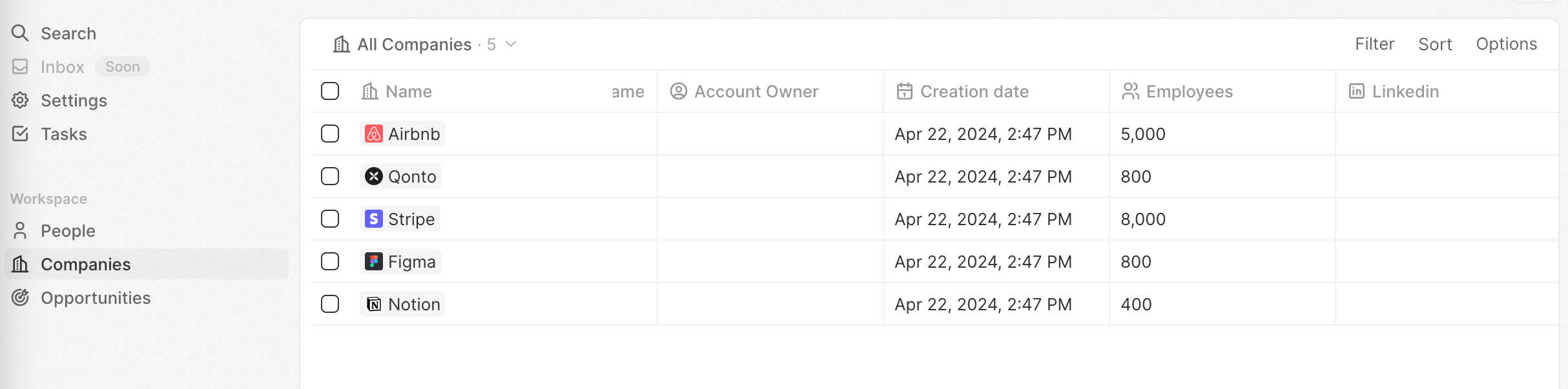


Рисунок 1.3 – Страница компаний сервиса Twenty

Страница компаний имеет такой же функционал, как и страница с клиентами.

На странице возможностей представлена канбан-доска, на которой можно создавать задачи. Страница возможностей представлена на рисунке 1.4.

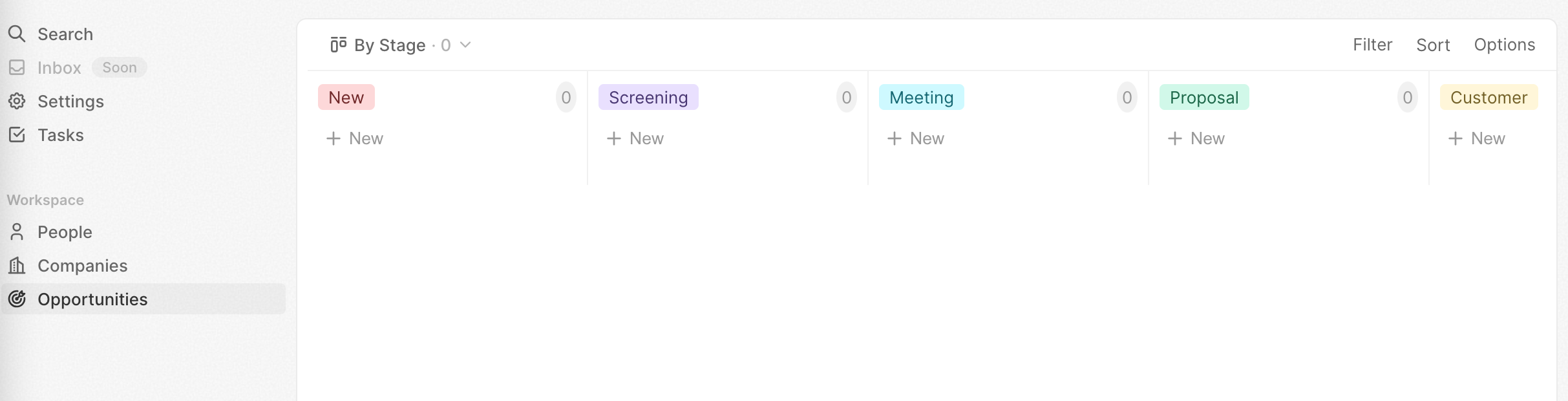


Рисунок 1.4 – Страница возможностей сервиса Twenty

Twenty – это современная и доступная платформа управления взаимоотношениями с клиентами с открытым исходным кодом, предлагающая ряд преимуществ: сильное сообщество разработчиков обеспечивает поддержку и экспертизу, а открытая дорожная карта позволяет пользователям вносить свои предложения и активно участвовать в развитии платформы. Однако недостатками могут быть ограниченный набор функций по сравнению с коммерческими аналогами и возможные сложности в настройке и поддержке для менее опытных пользователей.

### **1.1.2 Система Bitrix24**

Bitrix24 предлагает комплексное решение для управления бизнес-процессами, включающее CRM, управление задачами и проектами, создание сайтов и интернет-магазинов, а также автоматизацию различных аспектов бизнеса [2].

Сервис позиционируется как универсальное решение для бизнеса любого размера, предлагая различные тарифы и пакеты, включая коробочную версию для работы на собственном сервере.

Навигационное меню с функционалом сервиса представлено на рисунке 1.5.

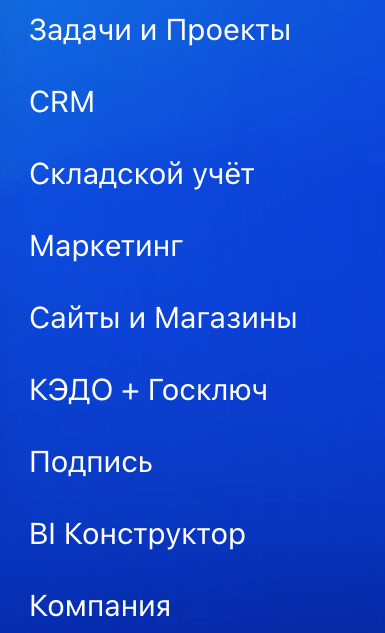


Рисунок 1.5 – Навигационное меню Bitrix24

Навигационная панель Bitrix24 включает в себя следующие разделы: «Задачи и Проекты» для управления задачами и проектами, где можно создавать, назначать и отслеживать выполнение задач; «CRM» для системы управления взаимоотношениями с клиентами, ведения клиентской базы и управления продажами.

«Складской учет» для возможности управления складом, отслеживания запасов и учета товаров на складе; «Маркетинг» с инструментами для удобного проведения маркетинговых кампаний и аналитики; «Сайты и Магазины» для создания и управления сайтами и интернет-магазинами.

«КЭДО + Госключ» для работы с электронным документооборотом и государственными электронными услугами; «Подпись» для управления электронными подписями; «BI Конструктор» с инструментами для создания отчетов и анализа данных и «Компания» для управления внутренними процессами компании, включая управление сотрудниками и внутренние коммуникации.

Эти разделы обеспечивают комплексное решение для управления различными аспектами бизнеса на одной платформе. Далее рассмотрим эти разделы подробнее.

На рисунке 1.6 представлена вкладка «Задачи и проекты».

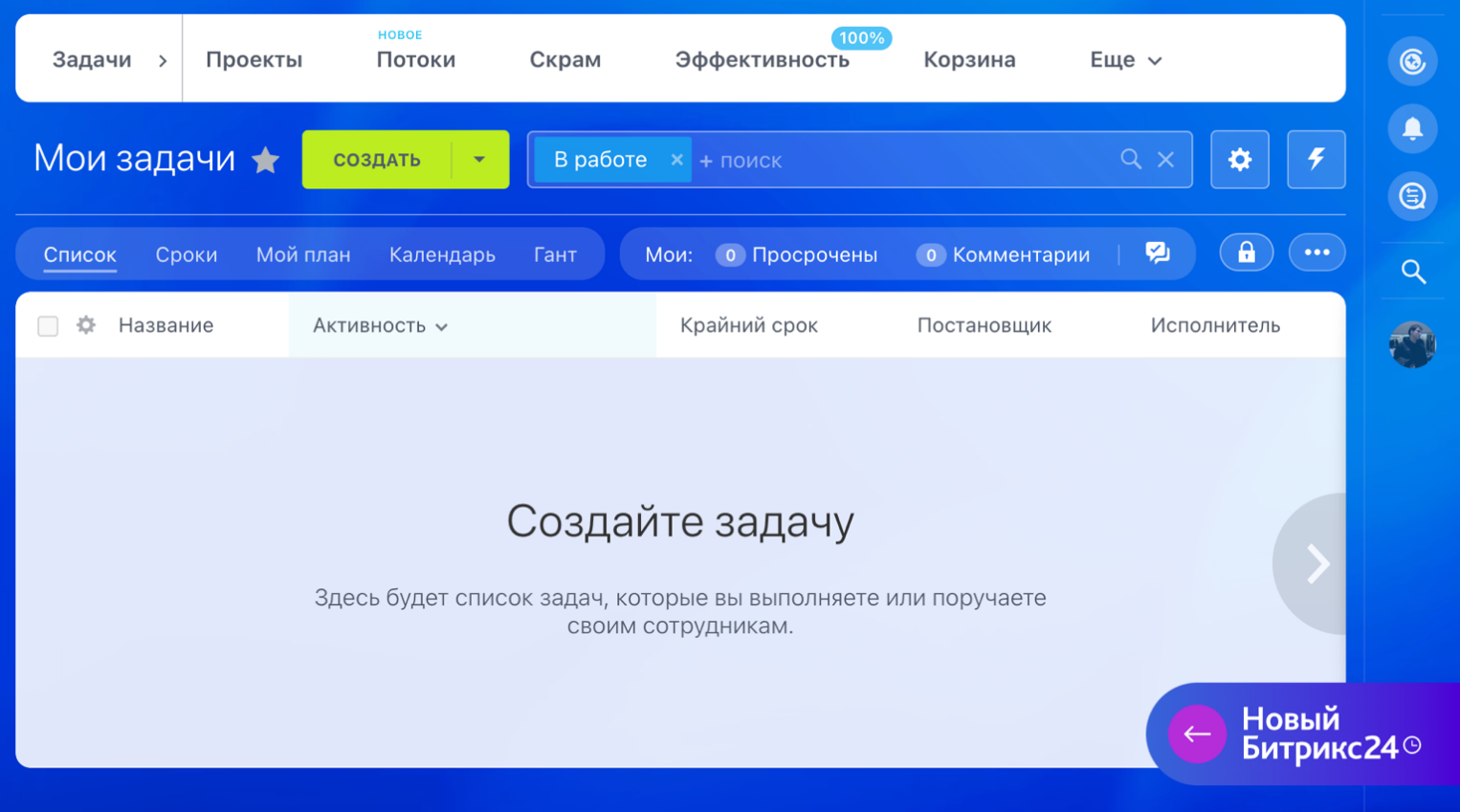


Рисунок 1.6 – Вкладка «Задачи и проекты»

На этой вкладке сервиса Bitrix24 отображается интерфейс пользователя для управления задачами. Основные разделы верхнего меню включают: «Задачи», «Проекты», «Потоки», «Скрам», «Эффективность» и «Корзина», позволяя быстро переключаться между основными функциями платформы.

Заголовок «Мои задачи» указывает на фильтрацию задач, назначенных именно этому пользователю. Это удобная опция, позволяющая быстро переключаться между общим списком задач и теми, за которые отвечает конкретный сотрудник. Возможность отметить некоторые задачи как «Избранные» с помощью звездочки позволяет выделить наиболее важные или приоритетные задачи, на которые нужно обращать особое внимание.

В этом разделе можно создать новую задачу, нажав на кнопку «Создать», и есть возможность фильтрации задач по статусу «В работе» и других параметрах с помощью строки поиска и настроек.

Под этим заголовком расположены различные виды отображения задач: «Список», «Сроки», «Мой план», «Календарь», «Гант». В таблице задач предусмотрены колонки «Название», «Активность», «Крайний срок», «Постановщик», «Исполнитель». Поскольку задач еще нет, в центральной части экрана показано сообщение «Создайте задачу» с пояснением, что здесь будет список задач, которые выполняются или поручены сотрудникам.

На рисунке 1.7 представлена вкладка «CRM».

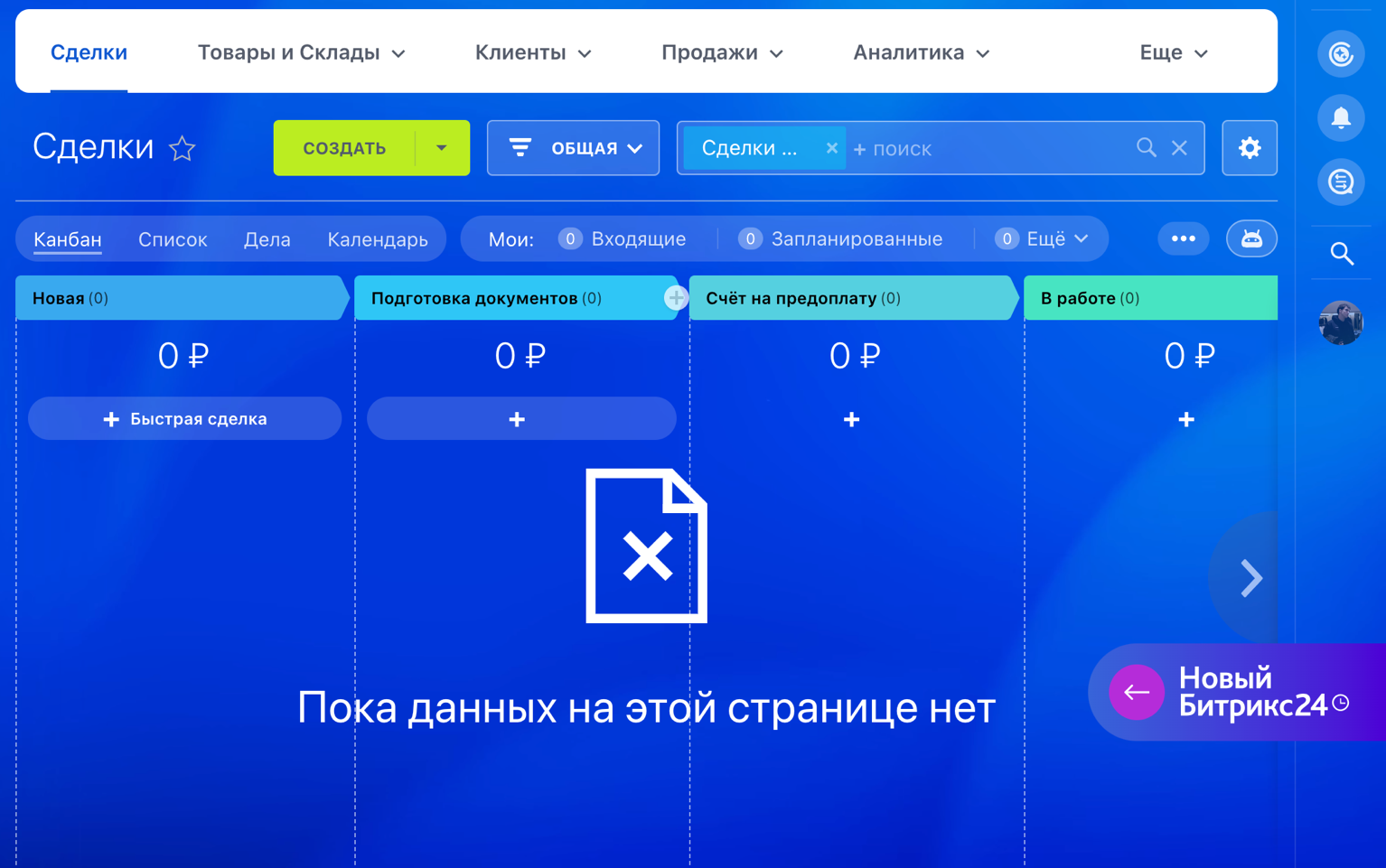


Рисунок 1.7 – Вкладка «CRM»

На этой вкладке Bitrix24 отображается интерфейс управления CRM (Customer Relationship Management). Основные разделы верхнего меню включают: «Сделки», «Клиенты», «Товары и склады», «Продажи» и «Аналитика».

Страница «Сделки» включает канбан-доску, на которой можно создавать сделки и менять им статусы, также представление может быть изменено на список, календарь или блоки, для наглядной визуализации информации.

Основными плюсами Bitrix24 являются комплексность решений, интеграция всех бизнес-процессов в одной системе, удобный интерфейс и богатый функционал, что помогает автоматизировать и оптимизировать работу компании. Однако, минусы включают возможные сложности в освоении из-за обширного функционала, зависимость от стабильного интернет-соединения, и некоторые пользователи могут столкнуться с высокими затратами на внедрение и обслуживание при использовании продвинутых функций. В целом, Bitrix24 является эффективным инструментом для компаний, стремящихся к цифровой трансформации и повышению производительности.

### **1.1.3 Система ClickUp**

ClickUp – это универсальная CRM-система, предназначенная для управления проектами и повышения производительности [3]. Она объединяет задачи, документы, командную коммуникацию и другие рабочие процессы в одном месте.

Основные функции включают доски, диаграммы Ганта, трекинг времени, чат, автоматизацию процессов и интеграцию с другими инструментами. ClickUp подходит для различных команд, от маркетинга до разработки, и помогает улучшить видимость проектов, сотрудничество и эффективность.

Система предлагает гибкость настройки и доступна бесплатно для начального использования. Вкладка «Projects» сервиса ClickUp представлена на рисунке 1.8.



Рисунок 1.8 – Вкладка «Projects»

Вкладка «Projects» – это раздел, предоставляющий полный список всех имеющихся проектов. Здесь имеется возможность найти подробную информацию о каждом проекте, чтобы легко управлять ими.

Каждый проект в списке сопровождается названием, которое было указано при создании проекта. Это помогает быстро идентифицировать каждый проект и ориентироваться среди них. Кроме того, для каждого проекта отображаются аватары сотрудников, которые назначены на проект. Таким образом, сразу видно, какие члены команды отвечают за каждый проект.

Уровень важности проекта также отражается в списке в виде флажков разных цветов. Имеется возможность определить уровень приоритета для каждого проекта, чтобы сосредоточиться на наиболее значимых задачах. Это помогает организовать свою работу и эффективно распределить время и ресурсы.

Дата создания проекта отображается рядом с ним. Это позволяет легко отслеживать, когда каждый проект был начат, и ориентироваться во времени. Знание даты создания помогает в планировании и приоритезации задач.

Для каждого проекта также можно менять статус выполнения. Имеется возможность отметить, что проект находится в процессе выполнения, завершен или находится в какой-либо другой стадии. Это полезно для отслеживания прогресса работы над проектами и обозначения их текущего состояния.

Вкладка «Projects» предоставляет возможность изменить отображение списка проектов. Можно выбрать между канбан-доской и календарем. Канбан-доска представляет проекты в виде колонок с задачами, которые можно перемещать и изменять их статус. Календарь позволяет визуально представить проекты в виде событий, запланированных на определенные даты.

Вкладка «Projects» обеспечивает удобный и наглядный способ управления проектами, помогая организовать работу, отслеживать прогресс и достигать поставленных целей в нужные сроки.

На рисунке 1.9 представлена вкладка «DashBoards».

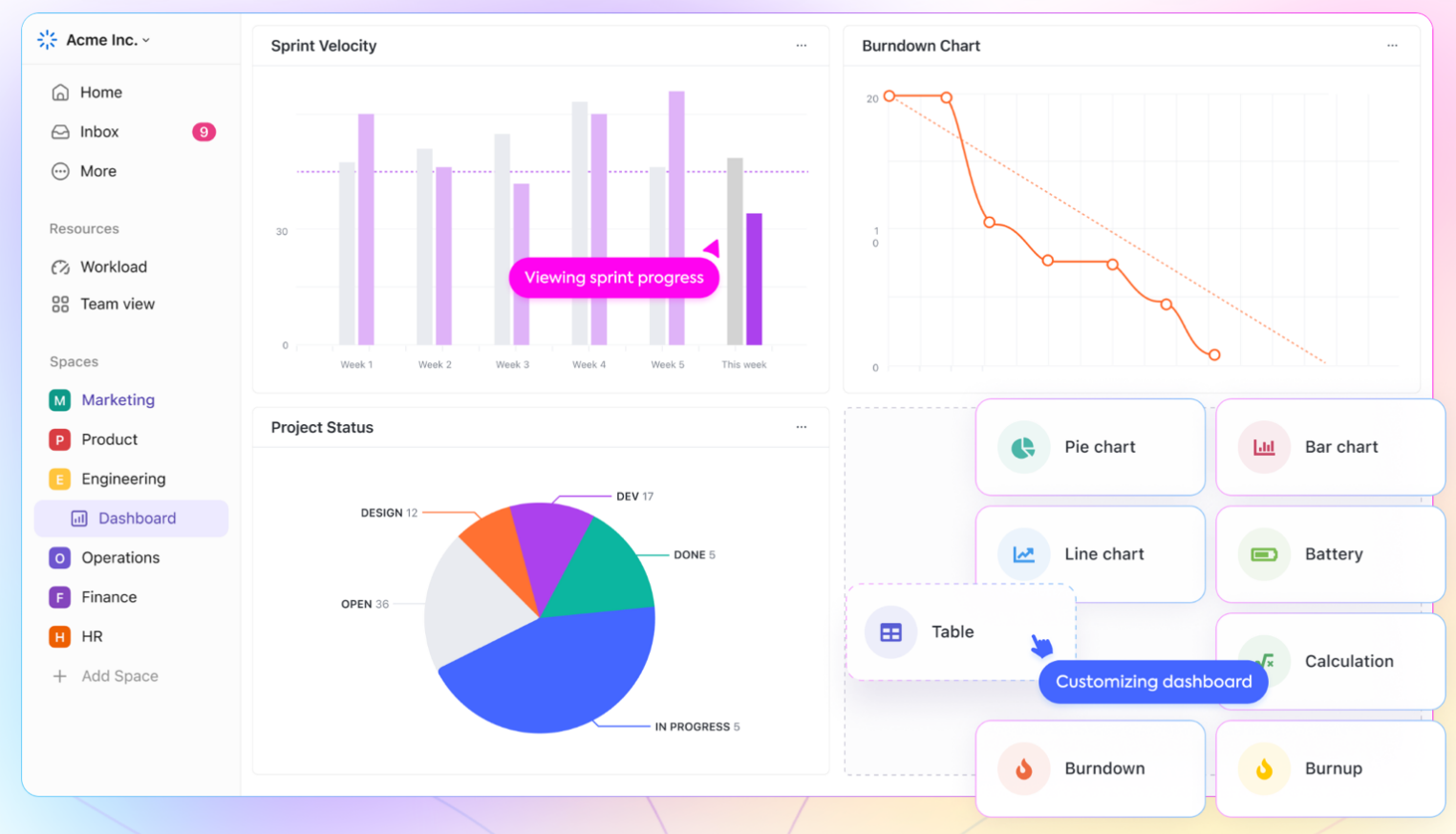


Рисунок 1.9 – Вкладка «DashBoards»

Вкладка «DashBoards» – это мощный инструмент, предоставляющий визуальное представление о динамике выполнения работ над проектами и других ключевых показателях. Здесь можно найти разнообразные графики и диаграммы, которые помогут легко отслеживать прогресс и оценивать эффективность работы вашей команды.

Один из основных графиков на вкладке «DashBoards» отображает динамику выполнения работ над проектами. Данный график предоставляет наглядное и интуитивно понятное представление о ходе выполнения различных проектов. Он позволяет с легкостью определить, насколько успешно проекты продвигаются, и выявить возможные задержки или проблемы на ранней стадии.

Интенсивность разработки также отражается на графиках вкладки «DashBoards». Имеется возможность увидеть, как активно ведется работа над проектами и какие периоды времени являются наиболее продуктивными. Это поможет оптимизировать распределение ресурсов и планировать более эффективное использование времени.

Количество задач и их статусы также представлены в виде диаграмм на вкладке «DashBoards». Имеется возможность визуально оценить общее количество задач, которые входят в каждый проект, и увидеть распределение задач по статусам. Это позволяет быстро оценить текущую нагрузку работы и сфокусироваться на задачах, требующих особого внимания.

Вкладка «DashBoards» предоставляет возможность настраивать графики и диаграммы в соответствии с вашими потребностями. Можно выбрать различные типы графиков (круговые, столбчатые, линейные и т. д.). Это дает гибкость в анализе данных и позволяет сосредоточиться на наиболее значимых аспектах вашей работы.

Среди преимуществ ClickUp – возможность централизованного управления задачами, документами и коммуникациями в одном месте, автоматизация бизнес-процессов, гибкая настройка рабочих процессов и широкий выбор интеграций с другими инструментами. Однако, некоторые пользователи могут столкнуться с изначальной сложностью освоения платформы и необходимостью осуществления дополнительных настроек для адаптации к конкретным потребностям команды.

## **1.2 Патентный поиск**

Патент – это исключительное право на изобретение. Патент наделяет своего владельца правом решать, каким образом изобретением может – и может ли – использоваться другими людьми. В обмен на это право патентообладатель раскрывает техническую информации об изобретении в публикуемом патентном документе.

В данном подразделе представлены результаты патентного исследования. В результате проведенного патентного поиска был выявлен схожий по тематике патент, представленный в таблице 1.1.

Таблица 1.1 – Описание патента

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование | Номер патента | Опубликовано | Авторы |
| Обработчик автоматического изменения поля источника данных GraphQL | US20230289339А1 | 14.09.2023 | Гаурав Сагар |

Реферат: «Сервер GraphQL хранит схемы для API GraphQL в хранилище схем, причем схемы определяют операции, типы и поля API. Резолверы полей связаны с полями API, чтобы указать, какой источник данных должен использоваться для выполнения запроса запроса API. При обнаружении события, которое вносит изменения в схему любого из источников данных, осуществляется доступ к хранилищу схем, и старое имя поля автоматически заменяется новым именем поля как в полученных схемах, так и в преобразователях полей, сопоставленных с полученными схемами. В ответ на получение запроса API определяется, есть ли в запросе имена полей, не соответствующие соответствующей схеме, и если да, то старое имя поля заменяется новым именем поля в запросе.» [4].

## **1.3 Постановка задачи**

После проведенного тщательного анализа существующих решений и потребностей пользователей, был сформирован функционал, которым должно обладать разрабатываемое приложение.

Приложение должно обладать следующим функционалом: просмотр канбан-доски, управление пользователями, управление задачами, общение посредством чата, а именно возможность создания чатов и отправки сообщений.

Приложение должно содержать четыре ключевых роли: администратор, менеджер, разработчик и клиент. Каждая из этих ролей будет наделена соответствующими правами и возможностями, обеспечивая эффективное взаимодействие всех участников процесса разработки проекта.

Администратор будет обладать широким спектром полномочий по управлению пользователями. Ему будет доступно добавление, удаление, редактирование и блокировка учетных записей. Кроме того, администратор сможет общаться с другими пользователями приложения посредством встроенного чата.

Менеджер, в свою очередь, будет наделен возможностями по созданию и управлению задачами. Он сможет назначать исполнителей (разработчиков) и заказчиков (клиентов) на конкретные задачи, создавать комментарии к ним, а также изменять статусы задач на канбан-доске. Менеджер также получит доступ к средствам общения через чат, чтобы общаться со всеми пользователями приложения.

Разработчики будут иметь возможность просматривать статусы задач на канбан-доске, оставлять комментарии к назначенным им задачам, а также использовать функционал чата для коммуникации с другими участниками.

Клиенты, в свою очередь, смогут наблюдать за статусами задач на канбан-доске и вести диалог с другими пользователями посредством встроенного чата.

Таким образом, четко определенные роли и соответствующие им права доступа позволят обеспечить структурированное и эффективное взаимодействие между всеми участниками процесса, вовлеченными в работу с приложением.

## **1.4 Выводы по разделу**

В данном разделе был проведен подробный анализ существующих аналогов разрабатываемого веб-приложения. В качестве аналогов были рассмотрены популярные сервисы, такие как Twenty, Bitrix24 и ClickUp.

Помимо обзора аналогов, в данном разделе был также проведен патентный поиск для выявления существующих технологий и решений, связанных с разрабатываемым веб-приложением. На основе анализа аналогов и патентного поиска был сформирован функционал, которым должно обладать новое веб-приложение, чтобы выделиться на фоне конкурентов и предложить уникальные возможности пользователям.

# 2 Проектирование веб-приложения

ФИО

Подпись

Дата

Лист

1

БГТУ 02.00.ПЗ

Разраб.

Левша М.С.

Провер.

Гончар Е.А.

Н. контр.

Нистюк О.А.

Утв.

Смелов В.В.

2 Проектирование

веб-приложения

Лит.

Листов

11

74417043, 2024

## **2.1 Используемые средства разработки**

### **2.1.1 Visual Studio Code**

Visual Studio Code – это мощный и гибкий редактор кода, разработанный Microsoft. Он поддерживает множество языков программирования и предлагает широкий спектр расширений, которые позволяют улучшить процесс разработки [5]. Visual Studio Code используется для написания кода, отладки и управления проектами.

### **2.1.2 MongoDB Compass**

MongoDB Compass – это графический интерфейс пользователя для MongoDB, который позволяет разработчикам и администраторам баз данных легко взаимодействовать с данными [6]. С помощью Compass можно создавать, редактировать и анализировать данные в MongoDB, а также управлять базами данных и коллекциями.

Кроме того, MongoDB Compass предоставляет возможность анализировать данные в MongoDB. С помощью интуитивных инструментов визуализации можно анализировать результаты в удобном графическом виде. Это облегчает и ускоряет процесс извлечения информации из базы данных.

### **2.1.3 Postman**

Postman – это мощный и широко используемый инструмент для тестирования и отладки API [7]. Он предоставляет разработчикам удобный способ отправки запросов к API и анализа полученных ответов.

Одним из основных преимуществ Postman является его поддержка различных типов запросов, включая GET, POST, PUT, DELETE, PATCH и многие другие. Пользователи могут легко создавать и настраивать запросы, указывая необходимые параметры, заголовки и тело запроса. Postman также предоставляет возможность работы с различными форматами данных, такими как JSON, XML, HTML и другими.

### **2.1.4 Fork**

Fork – это система контроля версий, которая позволяет разработчикам управлять изменениями в коде, сотрудничать над проектами и отслеживать историю изменений [8]. Fork поддерживает работу с Git и предлагает удобный графический интерфейс для управления репозиториями. Fork используется для координации работы над проектами, обеспечения совместной работы над кодом и управления версиями.

## **2.2 Используемые языки, библиотеки, сервисы**

В процессе разработки программного обеспечения, возникла необходимость использовать различные библиотеки и сервисы. Эти инструменты и технологии играют ключевую роль в создании эффективных и масштабируемых решений. В этом подразделе представлен обзор технологий, которые были наиболее активно использованы в процессе разработки, включая библиотеки и сервисы, которые помогли достичь поставленных целей и улучшить качество разработанного проекта.

### **2.2.1 JavaScript и Node.js**

Node.js – это платформа, которая позволяет разработчикам использовать JavaScript на стороне сервера [9]. Это открывает новые возможности для создания высокопроизводительных и масштабируемых веб-приложений и сервисов.

Node.js использует асинхронный, событийно-ориентированный подход, что делает его отличным выбором для разработки приложений, требующих высокой производительности и способных обрабатывать большое количество одновременных запросов. Node.js был выбран для создания серверной части приложения, обработки запросов от клиентов и взаимодействия с базой данных. Node.js также позволил мне использовать один и тот же язык программирования на всех этапах разработки, что упростило процесс разработки и улучшило качество кода.

### **2.2.2 Typescript**

TypeScript – это надмножество JavaScript, разработанное Microsoft, которое добавляет статическую типизацию в язык [10]. Это означает, что разработчики могут явно указывать типы данных для переменных, функций и объектов, что помогает улучшить читаемость кода и обнаруживать ошибки на ранних стадиях разработки. TypeScript также поддерживает современные функции JavaScript, такие как классы, интерфейсы, декораторы и модули, делая его мощным инструментом для разработки масштабируемых и надежных приложений.

В процессе разработки активно использовался TypeScript для создания сложных сервисов. Использование TypeScript позволило обеспечить более строгую проверку типов, что уменьшило количество ошибок во время выполнения и улучшило качество кода. TypeScript также упростил процесс разработки, предоставляя возможность использовать современные функции JavaScript и улучшенные возможности для работы с асинхронным кодом.

### **2.2.3 Фреймворк Nest.js**

Nest.js – это фреймворк для разработки серверных приложений, построенный на Node.js и TypeScript [11]. Он был создан с целью предоставить разработчикам мощный, масштабируемый и эффективный инструмент для создания современных веб-приложений и микросервисов. Nest.js использует архитектурный подход, основанный на принципах SOLID и DDD (Domain-Driven Design), что обеспечивает высокую степень модульности и тестируемости кода.

Одним из ключевых преимуществ Nest.js является его поддержка TypeScript, что позволяет разработчикам использовать статическую типизацию и другие преимущества этого языка, улучшая качество кода. Nest.js также предлагает модульную систему, которая позволяет легко организовывать код и повторно использовать компоненты в разных частях приложения.

В процессе разработки использовался Nest.js для создания серверной части приложения. Использование Nest.js позволило быстро создавать надежные и масштабируемые решения, благодаря его встроенным функциям, таким как обработка запросов, валидация данных, логирование и тестирование. Nest.js также поддерживает интеграцию с различными базами данных и внешними сервисами, что делает его универсальным инструментом для разработки современных веб-приложений.

В целом, Nest.js стал выбором для разработки серверной части приложений благодаря его гибкости, мощным функциям и поддержке TypeScript. Этот фреймворк позволяет создавать высокопроизводительные и масштабируемые решения с меньшим количеством ошибок и большей производительностью.

### **2.2.4 База данных MongoDB**

MongoDB – это документо-ориентированная база данных, которая использует формат хранения данных в виде документов, похожих на JSON [12]. Это отличает MongoDB от традиционных реляционных баз данных, где данные хранятся в таблицах. MongoDB предлагает гибкую схему данных, что позволяет легко адаптировать структуру данных под изменяющиеся требования приложения.

Одним из ключевых преимуществ MongoDB является ее масштабируемость, которая позволяет эффективно обрабатывать большие объемы данных. MongoDB использует горизонтальное масштабирование, что означает возможность добавления новых узлов для распределения нагрузки. Это позволяет системе легко масштабироваться по мере роста объема данных и увеличения требований к производительности.

При горизонтальном масштабировании MongoDB автоматически распределяет данные между узлами кластера, обеспечивая балансировку нагрузки и предотвращая ее перегрузку на отдельных узлах. Это позволяет достичь высокой производительности и эффективности при работе с большими объемами данных.

### **2.2.5 Библиотека Mongoose**

Mongoose – это библиотека для Node.js, которая предоставляет простую и мощную абстракцию для работы с MongoDB [13]. Mongoose также поддерживает множество функций, таких как запросы, валидация, плагины и миксины, что делает его мощным инструментом для разработки приложений, работающих с MongoDB.

Одним из ключевых преимуществ Mongoose является его поддержка схем данных. С помощью библиотеки Mongoose разработчики могут определить структуру данных, включая типы полей, валидацию и даже методы и статические методы, которые могут быть применены к документам. Это обеспечивает высокий уровень контроля над данными и помогает предотвратить ошибки, связанные с неправильной структурой данных.

Одним из преимуществ использования Mongoose является его интеграция с TypeScript. Благодаря поддержке TypeScript, Mongoose позволяет разработчикам использовать статическую типизацию и другие преимущества этого языка, улучшая качество кода и обеспечивая лучшую поддержку разработчика.

### **2.2.6 Библиотека Passport.js**

Passport.js – это аутентификационная библиотека для Node.js, которая предоставляет универсальный и модульный подход к аутентификации [14]. Она поддерживает множество стратегий аутентификации, включая OAuth, OpenID, и локальную аутентификацию с использованием имени пользователя и пароля. Passport.js позволяет разработчикам легко интегрировать различные методы аутентификации в свои приложения, обеспечивая безопасность и удобство для пользователей.

В процессе разработки веб-приложения для реализации системы аутентификации пользователей был использован популярная библиотека Passport.js. Одним из ключевых преимуществ использования Passport.js стала возможность легко реализовать аутентификацию на основе JWT токенов.

Одним из преимуществ использования Passport.js является его гибкость и модульность. Разработчики могут легко добавлять новые стратегии аутентификации или настраивать существующие, что делает Passport.js идеальным решением для разработки веб-приложений с различными требованиями к аутентификации.

### **2.2.7 Библиотека React.js**

React.js – это библиотека JavaScript для создания пользовательских интерфейсов, разработанная Facebook [15]. Она позволяет разработчикам создавать динамические и интерактивные веб-приложения с использованием компонентного подхода. React.js использует виртуальный DOM для оптимизации производительности, что позволяет приложениям быстро реагировать на изменения состояния и обновлять пользовательский интерфейс без перезагрузки страницы.

В процессе разработки использовался React.js для создания интерактивных веб-приложений и компонентов пользовательского интерфейса. Благодаря поддержке JSX, React.js позволяет разработчикам писать HTML-подобный код внутри JavaScript, что упрощает создание и поддержку пользовательских интерфейсов. React.js также предлагает мощные функции для управления состоянием приложения и обработки событий, что делает его идеальным инструментом для разработки сложных и динамических веб-приложений. Кроме того, React.js обеспечивает высокую производительность за счет использования виртуального DOM и технологии обновления только необходимых частей пользовательского интерфейса.

Существует множество библиотек и инструментов, разработанных специально для работы с React.js, таких как Redux для управления состоянием, React Router для маршрутизации и Material-UI для создания пользовательских интерфейсов. Эти инструменты и библиотеки расширяют возможности React.js и позволяют разработчикам создавать более мощные и надежные приложения.

В целом, React.js стал выбором для разработки пользовательских интерфейсов благодаря его гибкости, мощным функциям и поддержке компонентного подхода. Это позволит создать интерактивное и динамическое веб-приложение с высокой производительностью и удобством для пользователей.

## **2.3 Проектирование архитектуры базы данных**

В качестве СУБД была выбрана MongoDB, за ее гибкую схему данных, что позволяет хранить данные без явного определение структуры заранее. Это позволяет работать с данными переменной структуры.

В базе данных находится 5 коллекций, логическая схема базы данных представлена в приложении А и на рисунке 2.1.

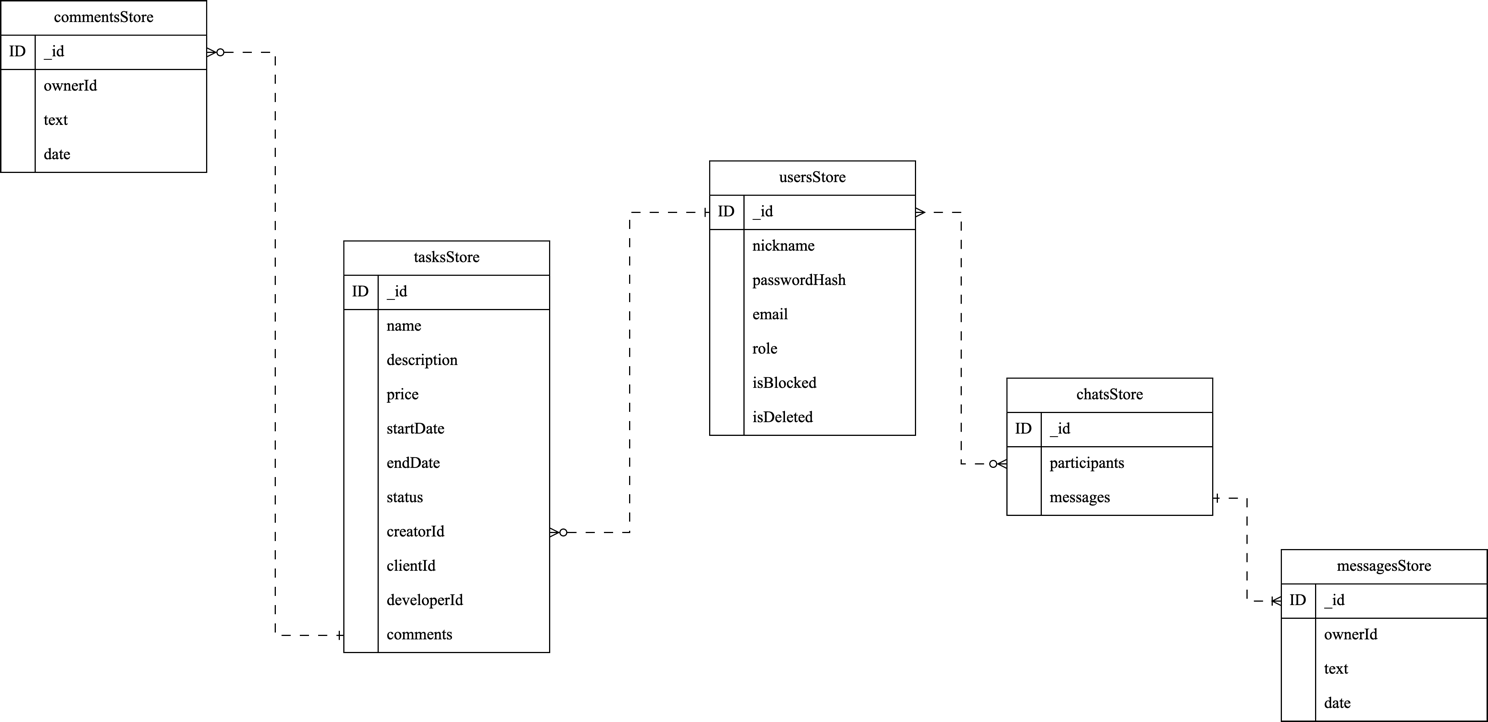


Рисунок 2.1 – Логическая схема базы данных

В коллекции usersStore хранится информация о пользователях приложения, tasksStore хранит все задачи, которые поступили от клиентов, в chatsStore находятся чаты, созданные пользователями, messagesStore хранит сообщения пользователей, а commentsStore комментарии к задачам.

Коллекция usersStore содержит следующие поля:

* \_id – тип данных ObjectId;
* nickname – тип данных String;
* passwordHash – тип данных String;
* role – тип данных String;
* email – тип данных String;
* isBlocked – тип данных Boolean;
* isDeleted – тип данных Boolean.

Столбец \_id содержит идентификатор пользователя, которому соответствует информация в документе, nickname – содержит никнейм пользователя, с помощью которого он аутентифицируется, passwordHash – содержит хешированный пароль пользователя, role – содержит роль пользователя в приложении, email – содержит почту пользователя, isBlocked – служит для блокировки пользователя, isDeleted – служит для мягкого удаления пользователя.

Коллекция tasksStore содержит следующие поля:

* \_id – тип данных ObjectId;
* name – тип данных String;
* description – тип данных String;
* price – тип данных Float;
* startDate – тип данных Date;
* endDate – тип данных Date;
* developerId – тип данных ObjectId;
* creatorId – тип данных ObjectId;
* status – тип данных String;
* comments – тип данных Array;
* clientId – тип данных ObjectId.

Столбец \_id содержит идентификатор задачи, которой соответствует информация в документе, name – содержит название задачи, description –описание задачи, price – содержит назначенную цену за задачу, startDate – содержит дату начала выполнения задачи, endDate – содержит дату, к которой задача должна быть выполнена, developerId – содержит идентификатор разработчика, ответственного за задачу, creatorId – содержит идентификатор пользователя, создавшего задачу, status – содержит статус выполенения задачи, comments – содержит массив объектов комментариев к задаче, clientId – содержит идентификатор клиента, заказавшего задачу.

Коллекция chatsStore содержит следующие поля:

* \_id – тип данных ObjectId;
* participants – тип данных Array;
* messages – тип данных Array.

Столбец \_id содержит идентификатор чата, которому соответствует информация в документе, participants – содержит массив пользователей, состоящих в чате, messages – содержит сообщения пользователей.

Коллекция commentsStore содержит следующие поля:

* \_id – тип данных ObjectId;
* ownerId – тип данных ObjectId;
* text – тип данных String;
* date – тип данных Date.

Столбец \_id содержит идентификатор комментария, которому соответствует информация в документе, ownerId – содержит идентификатор автора комментария, text – содержит комментарий пользователя, date – содержит дату написания комментария.

Коллекция messagesStore содержит следующие поля:

* \_id – тип данных ObjectId;
* ownerId – тип данных ObjectId;
* text – тип данных String;
* date – тип данных Date.

Столбец \_id содержит идентификатор сообщения, которому соответствует информация в документе, ownerId – содержит идентификатор автора сообщения, text – содержит сообщение пользователя, date – содержит дату написания сообщения.

После тщательного анализа и изучения каждой из коллекций базы данных, мы получили полное представление о структуре данных, используемых в нашем приложении. Этот процесс позволил нам определить ключевые поля документов, их типы данных и связи между различными коллекциями. Благодаря детальному описанию полей документов, мы смогли установить четкие правила для взаимодействия с базой данных, что обеспечило надежность и эффективность работы приложения.

Основываясь на полученных знаниях, были оптимизированы процессы взаимодействия с базой данных и упрощен процесс разработки.

Важность правильного проектирования базы данных не может быть недооценена. Она является основой для создания надежных и эффективных приложений, способных удовлетворять потребности пользователей и обеспечивать высокое качество работы. Благодаря тщательному анализу и описанию каждой из коллекций базы данных было улучшено качество веб-приложения.

В заключение, работа с базой данных является ключевым элементом в процессе разработки приложений. Она позволяет нам создавать мощные и гибкие решения, способные выдерживать изменения и расти вместе с нашими потребностями.

## **2.4 Проектирование структуры сервера**

При разработке сервера была использована модульная архитектура. Модульная архитектура использует концепции, такие как контроллеры, провайдеры, модули, посредники, фильтры исключений, конвейеры, защитники и перехватчики. Эти компоненты позволяют организовать код в модули, обеспечивая легкую масштабируемость и поддержку различных аспектов приложения, таких как маршрутизация, обработка запросов, валидация данных, аутентификация и авторизация.

Основные принципы модульной архитектуры включают:

* Разделение проблем: разделение приложения на отдельные разделы, каждый из которых отвечает за конкретную проблему или ответственность. Это гарантирует, что любое изменение в одном модуле будет иметь минимальное влияние на другие, ограничивая эффект ряби и повышая удобство обслуживания.
* Высокая сплоченность: сплоченность относится к степени, в которой элементы внутри модуля связаны друг с другом. Высокая согласованность означает, что модуль фокусируется на одной четко определенной цели, что упрощает разработку, отладку и тестирование, делая модуль более удобным для повторного использования, понятным и чистым.
* Низкая связь: связь – это степень взаимозависимости между модулями. Низкая связанность указывает на то, что модули независимы и минимально зависят друг от друга. Это способствует лучшей ремонтопригодности и гибкости, поскольку изменения в одном модуле меньше влияют на другие модули в системе.
* Сокрытие информации: сокрытие информации включает в себя сокрытие внутренней работы модуля, открывая только то, что необходимо, через четко определенный интерфейс. Эта абстракция предотвращает доступ внешних зависимостей к внутренним деталям модуля, сводя к минимуму влияние изменений и повышая удобство сопровождения приложения.
* Взаимодействие модулей на основе интерфейса: модули должны взаимодействовать друг с другом через стабильные, четко определенные интерфейсы вместо прямого доступа к внутренней реализации друг друга. Это разделяет модули, делая их более гибкими, отказоустойчивыми и удобными в сопровождении.

## **2.5 Диаграмма развертывания приложения**

Диаграмма развертывания приложения – это диаграмма, которая отображает организацию и взаимосвязи различных компонентов или модулей внутри приложения. Эта схема предоставляет высокоуровневый обзор структуры приложения, обозначая ключевые элементы и связи между ними. Цель структурной схемы заключается в том, чтобы облегчить понимание организации кода и архитектурных решений. Правильная структура приложения позволяет легко находить нужные файлы и компоненты, понимать их взаимосвязи и вносить изменения без ущерба для всего приложения. Она также способствует повторному использованию кода, улучшает его читаемость и делает его более отказоустойчивым. Структурная схема приложения представлена в приложении Б и на рисунке 2.2.

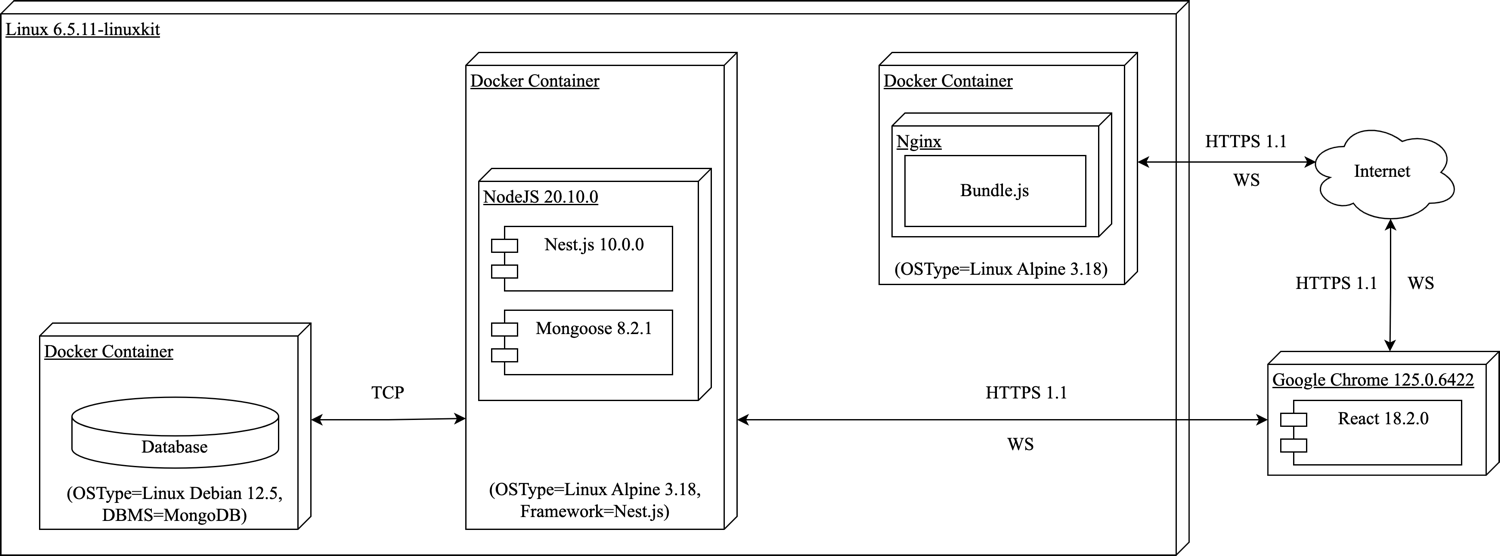


Рисунок 2.2 – Диаграмма развертывания приложения

Веб-приложение состоит из нескольких компонентов, которые взаимодействуют между собой для обеспечения работы приложения. Клиент, представленный браузером, запускает приложение, написанное на React, и отправляет запросы на сервер, где работает Nginx. Сервер принимает запросы от клиента через HTTPS или WebSocket, в зависимости от типа запроса, обрабатывает их и перенаправляет на NodeJS-приложение, используя протокол HTTPS.

NodeJS-приложение, в свою очередь, получает запросы от Nginx, обрабатывает их и взаимодействует с базой данных MongoDB. Для выполнения операций чтения и записи данных приложение использует библиотеку Mongoose, которая взаимодействует с MongoDB через протокол TCP. После выполнения необходимых операций с данными приложение возвращает результаты обратно через Nginx клиенту. Клиент, получив ответ от сервера, обновляет пользовательский интерфейс.

Весь процесс взаимодействия обеспечивает безопасное и эффективное соединение между компонентами приложения, поддерживая работу приложения в реальном времени. Таким образом была рассмотрена диаграмма развертывания приложения.

## **2.6 Диаграмма вариантов использования**

Диаграмма вариантов использования – это тип диаграммы UML, который показывает взаимодействие между системой и ее окружением, описывая различные сценарии использования системы со стороны ее пользователей или других систем [16].

Веб-приложение, в зависимости от роли, предоставляет пользователю функциональные возможности, которые показаны в диаграмме вариантов использования, которая представлена в приложении В и на рисунке 2.3.



Рисунок 2.3 – Диаграмма вариантов использования

Для того, чтобы администратор мог использовать возможности приложения, он должен аутентифицироваться. После этого администратор может осуществлять управление пользователями: добавлять, удалять, изменять, блокировать и разблокировать пользователей, также администратор может общаться с помощью чата с другими пользователями приложения.

Для того, чтобы менеджер мог использовать возможности приложения, он должен аутентифицироваться. После этого менеджер может создавать задачи, назначать им разработчиков и клиентов, писать комментарии к задачам, менять статусы задач на канбан-доске и общаться с помощью чата.

Для того, чтобы разработчик мог использовать возможности приложения, он должен аутентифицироваться. После этого разработчик может просматривать статусы задач на канбан-доске, писать комментарии к назначенным задачам, создавать чаты и отправлять сообщения другим пользователям.

Для того, чтобы клиент мог использовать возможности приложения, он должен аутентифицироваться. После этого клиент может просматривать статусы задач на канбан-доске и общаться с помощью чата.

Таким образом, была рассмотрена диаграмма вариантов использования.

## **2.7 Алгоритм создания задачи**

В процессе проектирования также была разработана блок-схема алгоритма создания задачи, что позволит лучше понять структуру и логику работы алгоритма, блок-схема представлена в приложении Г и на рисунке 2.4.

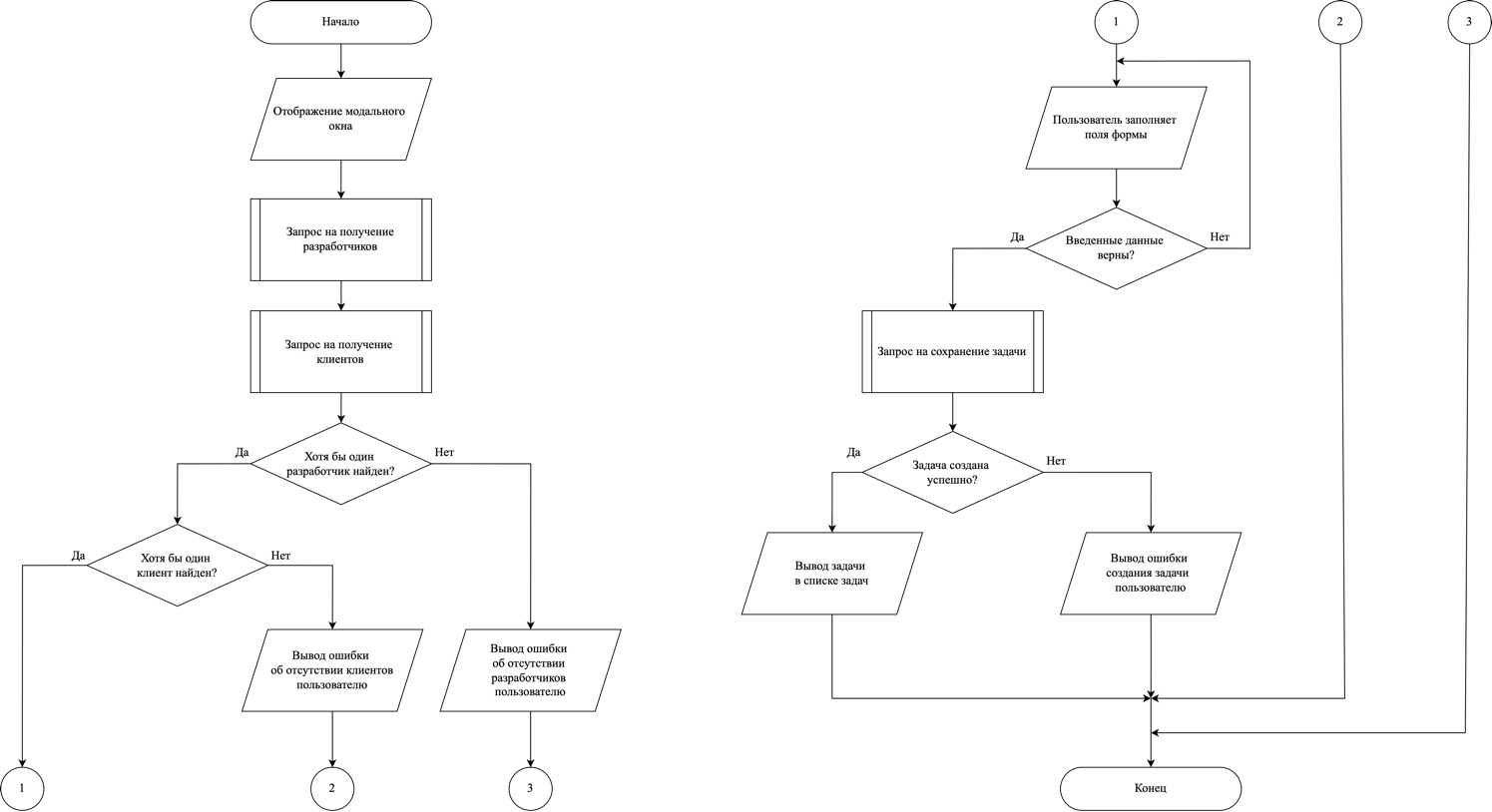


Рисунок 2.4 – Блок-схема алгоритма создания задачи

Блок-схема состоит из следующих блоков, каждый из которых выполняет определенное действие или проверку в процессе создания задачи. Давайте подробнее рассмотрим каждый блок:

* начало: обозначает начало процесса;
* отображение модального окна: модальное окно отображается пользователю для заполнения данных о задаче;
* запрос на получение разработчиков: выполняется запрос к серверу для получения списка всех разработчиков;
* запрос на получение клиентов: выполняется запрос к серверу для получения списка всех клиентов приложения;
* хотя бы один разработчик найден?: если «Нет», то переход к блоку «Вывод ошибки пользователю» и затем к «Конец», если «Да», то переход к следующему блоку;
* xотя бы один клиент найден?: если «Нет», то переход к блоку «Вывод ошибки пользователю» и затем к «Конец», если «Да», то переход к правому потоку;
* пользователь заполняет поля формы: пользователь вводит необходимые данные в форму, отображаемую в модальном окне;
* введенные данные верны?: если «Нет», то переход к блоку «Вывод ошибки пользователю» и затем к «Конец», если «Да», то переход к следующему блоку.
* запрос на сохранение задачи: выполняется запрос на сервер для сохранения новой задачи с введенными данными;
* задача создана успешно?: если «Нет», то переход к блоку «Вывод ошибки пользователю» и затем к «Конец», если «Да», то переход к «Конец»;
* вывод ошибки пользователю: в случае ошибки, сообщение об соответствующей ошибке отображается пользователю;
* конец: обозначает завершение процесса.

Таким образом был рассмотрен алгоритм создания задачи.

## **2.8 Выводы по разделу**

В данном разделе было выполнено проектирование веб-приложения. Был обоснован выбор средств разработки: Visual Studio Code, MongoDB Compass, Postman, Fork. Обоснован выбор технологий для разработки серверной (Node.js, TypeScript, Nest.js, Mongoose, Passport.js), клиентской части (React) веб-приложения, а также обоснован выбор технологий хранения данных (MongoDB).

На основе требований была разработана диаграмма вариантов использования.

Для хранения основной информации решено было использовать MongoDВ. Было разработано 5 коллекций, описано назначение каждой из них и содержимое документов. Также была спроектирована логическая схема базы данных.

В качестве архитектуры сервера была выбрана модульная архитектура, приведено ее описание и преимущества ее использования.

Кроме того, была спроектирована диаграмма развертывания веб-приложения и изложено ее детальное описание.

# 3 Разработка веб-приложения

ФИО

Подпись

Дата

Лист

1

БГТУ 03.00.ПЗ

Разраб.

Левша М.С.

Провер.

Гончар Е.А.

Н. контр.

Нистюк О.А.

Утв.

Смелов В.В.

3 Разработка веб-приложения

Лит.

Листов

16

74417043, 2024

## **3.1 Реализация серверной части**

### **3.1.1 Структура проекта**

Проект был разработан с использованием фреймворка Nest, который поддерживает модульную архитектуру. Для достижения модульности и удобства организации кода, файлы программы были распределены по соответствующим директориям.

Разделение серверной части проекта на логические модули имеет следующую структуру, представленную на рисунке 3.1.

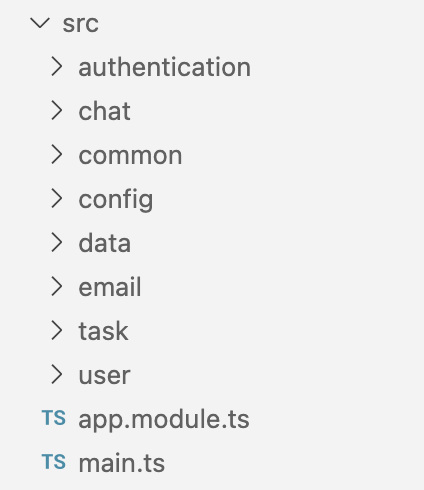


Рисунок 3.1 – Структура серверной части веб-приложения

Директория authentication в проекте содержит необходимые компоненты для реализации функциональности аутентификации. В ней находятся резолвер, который обрабатывает запросы на аутентификацию, сервисы для генерации и проверки JWT-токенов, а также сервис для хэширования паролей. Поддиректория strategies содержит стратегию аутентификации, определяющая способы проверки подлинности. Кроме того, в директории присутствует модуль аутентификации, который объединяет все эти компоненты вместе, устанавливает их зависимости и обеспечивает их взаимодействие. Такая организация файлов и директорий упрощает поддержку и расширение функциональности аутентификации в проекте, содержимое директории представлено на рисунке 3.2.



Рисунок 3.2 – Структура директории authentication

В директории common находятся вспомогательные сущности, такие как: константы, пользовательские декораторы, фильтры, гварды, интерфейсы, в поддиректории helpers находятся вспомогательные функции, поддиректория gpaphql содержит определение типов и структур для корректной работы graphQL, содержимое директории представлено на рисунке 3.3.

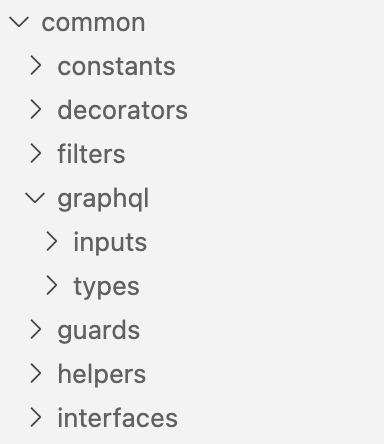


Рисунок 3.3 – Структура директории common

В директории config проекта содержатся файлы, которые содержат конфигурационные параметры для различных компонентов системы, таких как GraphQL, JWT и MongoDB. Конфигурация GraphQL определяет схему данных, типы запросов и мутаций, а конфигурация JWT включает параметры для генерации и проверки JWT-токенов. Файл конфигурации MongoDB содержит параметры подключения к базе данных MongoDB. Разделение конфигурации на отдельные файлы облегчает настройку проекта. Содержимое директории представлено на рисунке 3.4.

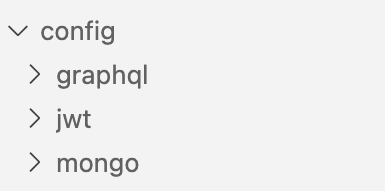


Рисунок 3.4 – Структура директории config

В директории data находятся модели, которые представляют сущности базы данных, в поддиректории repository содержатся репозитории для каждой модели, содержимое директории представлено на рисунке 3.5.

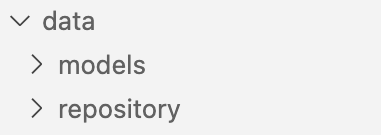


Рисунок 3.5 – Структура директории data

В директории email находится сервис для работы с электронной почтой, содержимое директории представлено на рисунке 3.6.

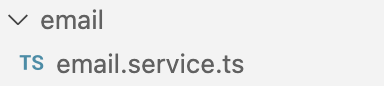


Рисунок 3.6 – Структура директории email

В директориях user, task и chat находятся ресурсы для обработки входящих запросов и работы с сущностями базы данных.

### **3.1.2 Реализация аутентификации**

Для реализации аутентификации используется jwt токен. Использование такой стратегии позволяет обеспечить достаточно высокий уровень безопасности.

При процессе аутентификации пользователь предоставляет свои учетные данные: логин и пароль. Если учетные данные верны, сервер создает jwt токен, который содержит информацию о пользователе или его правах. Затем сервер отправляет этот токен обратно клиенту, который сохраняет его в локальном хранилище.

При последующих запросах к серверу клиент включает токен в заголовке запроса. Сервер проверяет подпись токена и декодирует его содержимое, чтобы получить доступ к информации о пользователе. Если подпись действительна и токен не истек, сервер разрешает доступ к запрашиваемым ресурсам.

Подробнее рассмотрим реализацию метода login, который отвечает за генерацию jwt-токена. Код метода представлен в листинге 3.1.

async login(user: User): Promise<LoginResponse> {

const accessToken = await this.jwtService.signAsync({

nickname: user.nickname,

role: user.role,

sub: user.\_id,

});

const fullUserInfo = await this.userRepository.getOne({ \_id: user.\_id });

return {

accessToken: accessToken,

user: fullUserInfo,

};

}

Листинг 3.1 – Реализация метода login

Данный код представляет асинхронную функцию login, которая выполняет аутентификацию пользователя и генерирует jwt-токен. Внутри функции происходит подписывание данных пользователя с использованием jwtService и создание jwt-токена, включающего информацию о никнейме, роли и идентификаторе пользователя. Затем происходит получение полной информации о пользователе из userRepository. В результате функция возвращает объект LoginResponse, содержащий сгенерированный jwt-токен и полную информацию о пользователе.

Таким образом, была рассмотрена реализация аутентификации.

### **3.1.3 Реализация ресурса user**

В качестве примера реализации будут приведены несколько классов: UserService – содержит бизнес-логику управления пользователем и UserResolver– это класс, который принимает запросы для работы с сущностью пользователя.

Методы класса UserResolver:

* getUsers: выполняет запрос на получение списка пользователей с помощью метода getUsers из userService. Для доступа к этому методу требуются роли ADMIN или MANAGER. Возвращает объект GetUsersResponse;
* getUser: выполняет запрос на получение информации о конкретном пользователе с помощью метода getUser из userService. Для доступа к этому методу также требуются роли ADMIN или MANAGER. Возвращает объект User;
* createUser: выполняет создание нового пользователя с помощью метода createUser из класса userService. Для доступа к этому методу требуется роль ADMIN или MANAGER. Если роль, указанная при создании пользователя, недопустима для текущего пользователя, выбрасывается исключение BadRequestException. Возвращает созданного пользователя и отправляет уведомление на электронную почту с помощью emailNotificationService;
* deleteUser: выполняет удаление пользователя с помощью метода deleteUser из userService. Доступ к этому методу разрешен только для роли ADMIN. Если текущий пользователь пытается удалить самого себя, выбрасывается исключение BadRequestException. Возвращает удаленного пользователя и отправляет уведомление на электронную почту с помощью emailNotificationService;
* blockUser: выполняет блокировку или разблокировку пользователя с помощью метода blockUser из userService. Доступ к этому методу разрешен только для роли ADMIN. Если текущий пользователь пытается заблокировать или разблокировать самого себя, выбрасывается исключение BadRequestException. Возвращает заблокированного или разблокированного пользователя и отправляет уведомление на электронную почту с помощью emailNotificationService;
* updateUser: выполняет обновление информации о пользователе с помощью метода updateUser из userService. Доступ к этому методу разрешен только для роли ADMIN. Возвращает обновленного пользователя.

Подробнее рассмотрим реализацию метода blockUser, который отвечает за блокировку или разблокировку пользователя и отправку на почту пользователя информационного сообщения. Код метода представлен в листинге 3.2.

@Roles(UserRole.ADMIN)

@UseGuards(JwtAuthGuard, RoleGuard)

@Mutation(() => User, { name: 'blockUser', nullable: true })

async blockUser(

@Args('blockUserInput') blockUserInput: BlockUserInput,

@Context() context

): Promise<User> {

const user: JwtValidatedOutput = context.req.user;

if (user.userId === blockUserInput.\_id)

throw new BadRequestException('Вы не можете заблокировать самого себя');

const blockedUser = await this.userService.blockUser(blockUserInput);

const emailParams: EmailNotificationParams = {

userEmail: blockedUser.email,

subject: 'Your access was updated',

message: `Your account was ${blockedUser.isBlocked ? 'blocked' : 'unblock'}`,

};

this.emailNotificationService.sendEmailNotification(emailParams);

return blockedUser;

}

Листинг 3.2 – Реализация метода blockUser в классе UserResolver

Данный код представляет метод blockUser в классе UserResolver, который выполняет блокировку или разблокировку пользователя на основе входных данных. Внутри метода происходит получение информации о текущем пользователе, проверка на попытку заблокировать самого себя, вызов соответствующего метода из userService для блокировки или разблокировки пользователя, отправка уведомления на электронную почту заблокированного пользователя, и возврат заблокированного или разблокированного пользователя. Метод также содержит декораторы для определения требуемых ролей, гвард для аутентификации и авторизации, а также указания типа возвращаемого значения и имени операции.

Для каждого метода класса UserResolver реализован соответствующий метод класса UserService, далее будет приведено описание методов класса UserService.

Методы класса UserService:

* getUsers: получает список пользователей с помощью метода getMany из userRepository на основе переданных параметров getUsersInput и clientId. Возвращает объект GetUsersResponse, содержащий список пользователей и общее количество пользователей, удовлетворяющих условиям фильтрации;
* getUser: получает информацию о конкретном пользователе на основе переданного идентификатора userId. Использует метод getOne из класса userRepository для получения пользователя. Возвращает объект User;
* createUser: создает нового пользователя на основе переданных данных createUserInput. Проверяет, существует ли уже пользователь с таким же никнеймом. Шифрует пароль с помощью cryptographyService и сохраняет пользователя в базе данных с помощью метода create из userRepository. Возвращает созданного пользователя или выбрасывает исключение BadRequestException в случае ошибки;
* deleteUser: удаляет пользователя на основе переданного идентификатора userId. Обновляет поле isDeleted для пользователя с помощью метода updateOne из userRepository. Возвращает удаленного пользователя или выбрасывает исключение BadRequestException в случае ошибки;
* blockUser: блокирует или разблокирует пользователя на основе переданных данных blockUserInput. Обновляет поле isBlocked для пользователя с помощью метода updateOne из userRepository. Возвращает заблокированного или разблокированного пользователя или выбрасывает исключение BadRequestException в случае ошибки;
* updateUser: обновляет информацию о пользователе на основе переданных данных updateUserInput. Обновляет соответствующие поля пользователя с помощью метода updateOne из userRepository. Возвращает обновленного пользователя или выбрасывает исключение BadRequestException в случае ошибки.

Для полноты картины в классе userService также рассмотрим метод blockUser. Код метода представлен в листинге 3.3.

async blockUser(blockUserInput: BlockUserInput): Promise<User> {

try {

const updateFields = {

isBlocked: blockUserInput.isBlocked,

};

const filter = { \_id: convertStringToObjectId(blockUserInput.\_id) };

await this.userRepository.updateOne(updateFields, filter);

return await this.userRepository.getOne(filter);

} catch (e) {

throw new BadRequestException('Ошибка блокировки пользователя');

}

}

Листинг 3.3 – Реализация метода blockUser в классе UserService

Метод blockUser в классе UserService выполняет блокировку или разблокировку пользователя на основе переданных данных. Внутри метода происходит обновление поля isBlocked пользователя в базе данных с помощью метода updateOne из userRepository. Затем получается обновленная информация о пользователе с помощью метода getOne из userRepository и возвращается. Если происходит ошибка в процессе блокировки или разблокировки, выбрасывается исключение BadRequestException. Метод возвращает Promise<User>, так как выполняет асинхронные операции.

Таким образом была рассмотрена реализация ресурса user.

### **3.1.4 Реализация ресурса task**

В качестве примера реализации будут приведены классы TaskService, содержащий бизнес-логику для работы с сущностью задачи, и TaskResolver, принимающий запросы относящиеся к сущности задачи.

Рассмотрим класс TaskResolver, методы этого класса:

* getTasks: получает список задач на основе переданных параметров getTasksInput. Метод вызывает getTasks из taskService, передавая параметры getTasksInput и идентификатор пользователя из контекста. Возвращает объект GetTasksResponse, содержащий список задач и дополнительную информацию;
* createTask: создает новую задачу на основе переданных данных createTaskInput. Метод вызывает createTask из taskService, передавая параметры createTaskInput и идентификатор пользователя из контекста. Возвращает созданную задачу;
* createComment: cоздает новый комментарий к задаче на основе переданных данных createCommentInput. Метод вызывает createComment из taskService, передавая параметры createCommentInput и идентификатор пользователя из контекста. Возвращает обновленную задачу с добавленным комментарием;
* getTask: Получает информацию о конкретной задаче на основе переданного идентификатора taskId. Метод вызывает getTask из taskService для получения задачи, передавая taskId. Возвращает объект Task;
* deleteTask: Удаляет задачу на основе переданного идентификатора taskId. Метод вызывает deleteTask из taskService, передавая taskId. Возвращает удаленную задачу.

Более подробнее рассмотрим метод createComment, который создает новый комментарий. Код метода представлен в листинге 3.4.

@Roles(UserRole.ADMIN, UserRole.MANAGER, UserRole.DEVELOPER, UserRole.CLIENT)

@UseGuards(JwtAuthGuard, RoleGuard)

@Mutation(() => Task, { name: 'createComment' })

async createComment(

@Args('createCommentInput') createCommentInput: CreateCommentInput, @Context() context): Promise<Task> {

const client: JwtValidatedOutput = context.req.user as JwtValidatedOutput;

return this.taskService.createComment(createCommentInput, client.userId);

}

Листинг 3.4 – Реализация метода createComment в классе TaskResolver

Метод createComment в классе TaskResolver выполняет создание нового комментария к задаче. Внутри метода проверяются роли пользователей, используется аутентификация с помощью JWT-токена, и запускается операция GraphQL для создания комментария. Данные комментария передаются в createCommentInput. Из контекста извлекается информация о текущем пользователе, и вызывается метод createComment из taskService, который осуществляет создание комментария в соответствии с бизнес-логикой. Возвращается обновленная задача с добавленным комментарием. Метод возвращает Promise<Task>, так как выполняет асинхронные операции.

Каждый из методов класса TaskResolver соответствует одноименному методу класса TaskService, который взаимодействует с базой данных для выполнения соответствующих операций с задачами. Реализация класса TaskService представлена в приложении Ж, далее будет приведено описание методов класса.

Методы класса TaskService:

* getTasks: получает список задач на основе переданных параметров getTasksInput и идентификатора клиента clientId. Внутри метода создается регулярное выражение для фильтрации задач по имени. Затем выполняется подсчет общего количества задач, соответствующих фильтру. Далее задачи получаются из базы данных с помощью метода getMany из taskRepository, применяется фильтрация по создателю, разработчикам и клиенту, и возвращается список задач вместе с их общим количеством;
* createTask: создает новую задачу на основе переданных данных createTaskInput и идентификатора клиента clientId. Внутри метода создается объект taskFields, содержащий данные для создания задачи, включая идентификатор создателя и пустой массив комментариев. Затем вызывается метод create из taskRepository для создания задачи в базе данных. Если происходит ошибка, выбрасывается исключение BadRequestException. Возвращается созданная задача;
* createComment: создает новый комментарий к задаче на основе переданных данных createCommentInput и идентификатора клиента clientId. Внутри метода извлекаются данные из createCommentInput, создается объект commentFields с информацией о комментарии и идентификатором пользователя. Затем вызывается метод update из класса taskRepository, чтобы добавить комментарий к задаче с помощью оператора $push. Затем обновленная задача получается из базы данных с помощью метода getOne из класса taskRepository. Для каждого комментария в задаче извлекается информация о владельце комментария с помощью метода getOne из класса userRepository. Затем комментарии обновляются с информацией о владельце, и возвращается обновленная задача;
* getTask: получает информацию о задаче на основе переданного идентификатора taskId. Внутри метода вызывается метод getOne из taskRepository для получения задачи из базы данных. Затем для каждого комментария в задаче извлекается информация о владельце комментария с помощью метода getOne из userRepository. Комментарии обновляются с информацией о владельце, и возвращается задача с обновленными комментариями;
* deleteTask: удаляет задачу на основе переданного идентификатора taskId. Внутри метода вызывается метод delete из taskRepository для удаления задачи из базы данных. Возвращается удаленная задача.

Рассмотрим метод createComment класса TaskService. Код данного метода представлен в листинге 3.5.

async createComment(createCommentInput: CreateCommentInput, clientId: string): Promise<Task> {

const { taskId, ...commentInfo } = createCommentInput;

const commentFields = {

...commentInfo,

user: convertStringToObjectId(clientId),

};

try {

await this.taskRepository.update(taskId, {

$push: { comments: commentFields },

});

const task = await this.taskRepository.getOne({ \_id: convertStringToObjectId(taskId) });

const newComments = [];

for (const item of task.comments) {

const owner = await this.userRepository.getOne({

\_id: convertStringToObjectId(item.user.\_id),

});

newComments.push({

text: item.text,

date: item.date,

user: owner,

});

}

task.comments = newComments;

return task;

} catch (e) {

throw new BadRequestException('Ошибка создания комментария');

}

}

Листинг 3.5 – Реализация метода createComment в классе TaskService

Метод createComment в классе TaskService выполняет создание нового комментария к задаче. Внутри метода комментарий извлекается из входных данных, создается объект с данными комментария и идентификатором пользователя, который добавляется в базу данных с помощью метода update из taskRepository. Затем обновленная задача получается с помощью метода getOne из taskRepository, и для каждого комментария извлекается информация о владельце комментария. Обновленная задача с комментариями и владельцами возвращается. Если происходит ошибка, выбрасывается исключение BadRequestException.

Таким образом, в ходе разработки была успешно реализована функциональность, связанная с ресурсом task. Это включает в себя создание, отображение и удаление задач. Кроме того, важными аспектами реализации функциональности ресурса task являются обработка ошибок и обеспечение безопасности данных. Все запросы и операции, связанные с задачами, должны быть надежными и защищенными от несанкционированного доступа.

### **3.1.5 Реализация взаимодействия с базой данных**

Для взаимодействия с базой данных был выбран пакет mongoose, чтобы его использовать было необходимо реализовать модели для сопоставления с документами, пример модели представлен в листинге 3.6.

@Schema({ collection: 'usersStore', versionKey: false })

@ObjectType()

export class User {

@Field(() => String)

\_id: string;

@Prop({ required: true })

@Field(() => String)

nickname: string;

@Prop({ required: true})

@Field(() => String)

email: string;

@Prop({ required: true })

@Field(() => String)

passwordHash: string;

@Prop()

@Field(() => String, { nullable: true })

avatar: string;

@Prop({ required: true, type: String, enum: UserRole })

@Field(() => UserRole)

role: UserRole;

@Prop()

@Field(() => Boolean)

isBlocked: false;

@Prop()

@Field(() => Boolean)

isDeleted: false;

}

export type UserDocument = HydratedDocument<User>;

export const UserSchema = SchemaFactory.createForClass(User);

Листинг 3.6 – Реализация модели User

В репозиториях, которые служат связующим звеном между моделями и бизнес-логикой, реализован MongoGenericRepository для работы с моделями в приложении. Этот репозиторий предоставляет 11 методов: getMany для получения нескольких сущностей, getOne для получения одной сущности, create для создания сущности, createMany для создания нескольких сущностей, update для изменения сущности, updateMany для изменения нескольких сущностей, delete для удаления сущности, deleteMany для удаления нескольких сущностей, bulkWrite для пакетной записи нескольких документов и count для подсчета количества элементов в коллекции. Некоторые методы поддерживают параметр populate, который позволяет выполнять маппинг связанных полей из других коллекций. Этот параметр позволяет извлекать данные из разных коллекций и объединять их в один объект при получении данных из базы. Таким образом, MongoGenericRepository предоставляет удобный интерфейс для работы с базой данных, абстрагируя детали взаимодействия с MongoDB и облегчая выполнение операций CRUD и других операций с моделями приложения. Более детальная реализация MongoGenericRepository представлена в приложении И. Реализация метода getMany репозитория представлена в листинге 3.7.

async getMany(filter: any, sort?: any, skip?: any, limit?: any, session?: any): Promise<T[]> {

return this.\_repository

.find(filter, null, { session: session })

.skip(skip)

.limit(limit)

.sort(sort)

.populate(this.\_populatedFields)

.exec();

}

Листинг 3.7 – Реализация метода getMany репозитория MongoGenericRepository

Все репозитории в приложении наследуются от MongoGenericRepository, что обеспечивает единообразный подход к работе с базой данных MongoDB. Наследование также позволяет добавлять дополнительные методы или переопределять существующие методы в подклассах репозиториев, чтобы адаптировать их под конкретные требования приложения. Это приводит к сокращению дублирования кода, упрощению разработки и поддержки репозиториев, помогает сосредоточиться на специфической бизнес-логике в репозиториях, не беспокоясь о реализации основных операций базы данных, так как они уже реализованы в MongoGenericRepository.

Таким образом была рассмотрена реализация взаимодействия с базой данных.

## **3.2 Реализация клиентской части**

Клиентская часть проекта была реализована с использованием библиотеки React. React был выбран в качестве основной технологии благодаря своей популярности и широкому экосистеме библиотек и инструментов, доступных для разработчиков. React предоставляет мощный набор инструментов для создания динамических и масштабируемых пользовательских интерфейсов. Он основан на компонентном подходе, где пользовательский интерфейс разбит на небольшие, переиспользуемые компоненты. Это позволяет разработчикам строить приложения из независимых компонентов, которые могут быть использованы повторно в разных частях приложения.

### **3.2.1 Структура проекта**

При реализации клиентской части проекта с использованием JavaScript библиотеки React был применен модульный подход. Это означает, что пользовательский интерфейс был разбит на отдельные компоненты, которые можно переиспользовать в различных частях приложения. Модульный подход позволяет улучшить структуру кода, облегчить его понимание и обеспечить легкость поддержки и расширения приложения. Структура клиентскойчасти представлена на рисунке 3.7.

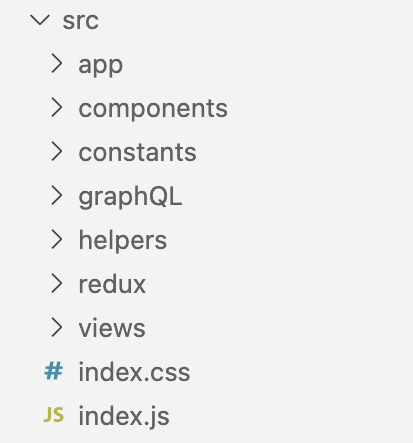


Рисунок 3.7 – Структура клиентской части веб-приложения

В директории app находится главный компонент приложения, который служит отправной точкой для всей архитектуры приложения и управляет его основной логикой, маршрутизацией и состоянием.

Папка components содержит многоразовые компоненты, которые используются на различных страницах приложения. Эти компоненты включают навигационную панель, шаблон страниц и другие элементы интерфейса, обеспечивающие взаимодействие пользователя с приложением.

Директория constants содержит константы, которые используются в различных частях приложения для обеспечения консистентности и упрощения управления кодом.

Папка graphQL содержит запросы и мутации, используемые для взаимодействия с серверной частью приложения. Здесь находятся все запросы на получение данных, их изменение или удаление, что позволяет эффективно управлять данными на клиенте.

В директории helpers находятся вспомогательные функции, которые помогают упростить выполнение общих задач в приложении. Эти функции включают обработку данных, форматирование дат, выполнение валидации и другие утилиты, которые часто используются в различных частях приложения.

Директория redux содержит хранилище Redux и действия для изменения состояния хранилища. Здесь находятся редьюсеры, отвечающие за обновление состояния на основе получаемых действий, и сами действия, определяющие типы изменений состояния. Эта структура позволяет централизованно управлять состоянием приложения.

Папка views содержит страницы приложения, которые являются основными представлениями, отображаемыми пользователю. Каждая страница состоит из нескольких компонентов и представляет собой определенный экран в приложении, например, чат или панель администратора.

Кроме того, в корне папки src находятся файлы index.css и index.js. Файл index.css содержит глобальные стили для всего приложения, определяющие базовое оформление и внешний вид. Файл index.js является точкой входа в приложение, где происходит инициализация и рендеринг основного компонента приложения в DOM-дерево.

### **3.2.2 Реализация компонента App**

Компонент App является корневым компонентом приложения, который определяет маршрутизацию при помощи библиотеки react-router-dom. Код данного компонента представлен в листинге 3.8.

function App() {

return (

<>

<Router>

<Routes>

<Route element={<LoginView />} path="/login" />

<Route element={<Layout />} path="/">

<Route element={<TasksView />} path="tasks" />

<Route element={<AdminView />} path="admin" />

<Route element={<TaskView />} path="task" />

<Route element={<KanbanView />} path="kanban" />

<Route element={<ChatView />} path="chat" />

</Route>

</Routes>

</Router>

</>);}  
export default *App*;

Листинг 3.8 – Реализация компонента App

Внутри компонента App используется компонент Router, который обертывает весь контент приложения и обеспечивает маршрутизацию на основе URL-адреса. Внутри Router используется компонент Routes, который определяет набор маршрутов, которые должны быть обработаны приложением.

Второй маршрут использует компонент Layout в качестве общего макета для нескольких вложенных маршрутов. Когда URL-адрес соответствует корневому пути, будет отображаться компонент Layout.

Компонент Layout представляет макет страницы, которая будет использоваться внутри корневого компонента App. Он определяет элементы макета и используется для отображения вложенных маршрутов. Код компонента приведен в листинге 3.9.

const Layout = () => {

return (

<div className={styles.contentContainer}>

<NavBar />

<Outlet />

</div>

);

};  
export default Layout;

Листинг 3.9 – Реализация компонента Layout

Компонент Layout импортирует компонент Outlet из библиотеки react-router-dom. Outlet является специальным компонентом, который отображает вложенные маршруты, определенные в родительском маршруте.

Таким образом были рассмотрены компоненты App и Layout.

### **3.2.3 Реализация навигационной панели**

Компонент NavBar представляет навигационную панель, которая отображается в левой части страницы и предоставляет пользователю возможность перемещаться между страницами приложения. Она состоит из списка элементов, каждый из которых представляет определенную страницу приложения. Код компонента NavBar приведен в листинге 3.10.

const NavBar = () => {

const navigate = useNavigate();

const user = useSelector((state) => state.app.user);

return (<>

<ListGroup defaultActiveKey="#link4">

<ListGroup.Item action onClick={() => navigate('/chat')} href="#link4"> Chat

</ListGroup.Item>

{user.role !== 'ADMIN' && (

<ListGroup.Item action onClick={() => navigate('/kanban')} href="#link1"> Kanban

</ListGroup.Item>

)}

{(user.role === 'MANAGER' || user.role === 'DEVELOPER') && (

<ListGroup.Item action onClick={() => navigate('/tasks')} href="#link2"> Tasks

</ListGroup.Item>

)}

{user.role === 'ADMIN' && (

<ListGroup.Item action onClick={() => navigate('/admin')} href="#link3"> Admin panel

</ListGroup.Item>

)}

<ListGroup.Item action onClick={() => logout()} href="#link5"> Logout

</ListGroup.Item>

</ListGroup>

</>);};

export default NavBar;

Листинг 3.10 – Реализация компонента NavBar

Хук useSelector из react-redux позволяет компоненту получать информацию из глобального состояния приложения, хранящегося в хранилище Redux. В данном случае, компонент NavBar использует useSelector для получения информации о текущем пользователе из состояния приложения, чтобы разрешать пользователю доступ к нужным страницам. В итоге, компонент NavBar предоставляет удобную навигационную панель для перемещения между страницами приложения.

### **3.2.4 Реализация страницы задач**

Данная страница содержит функционал для отображения списка задач, их создания и удаления. Внутри компонента определены состояния для хранения данных о задачах, разработчиках, клиентах и других свойствах задачи, таких как имя, описание, цена, разработчик, клиент и дата. Также определены состояния для отображения модального окна и для хранения информации о добавленной и удаленной задачах. С реализацией компонента TasksView можно ознакомиться в приложении К.

В начале компонента определены несколько состояний с помощью хука useState. Например, tasks хранит массив задач, show определяет, должно ли отображаться модальное окно, developers и clients содержат списки разработчиков и клиентов соответственно. Остальные состояния, такие как name, description, price, developer, client и date, используются для управления значениями полей формы в модальном окне.

Далее определены функции-обработчики событий, такие как handleNameChange, handleDescriptionChange и т.д., которые обновляют соответствующие состояния при изменении значений полей формы.

Компонент также использует хук useSelector из библиотеки react-redux для получения данных из глобального состояния приложения, в данном случае - пользователя.

Затем следует использование хука useLazyQuery для выполнения GraphQL-запросов на получение задач и пользователей. Запросы определены в файлах taskQueries и userQueries, соответственно. При получении данных, результаты запросов сохраняются в соответствующих состояниях (tasks, developers, clients).

Также определены функции handleClose и handleShow, которые управляют отображением модального окна. При вызове handleShow, происходит выполнение запросов на получение списка разработчиков и клиентов.

Таким образом была рассмотрена страница задач.

### **3.2.5 Реализация компонента GraphQLProvider**

Компонент GraphQlProvider является оберткой для библиотеки Apollo Client, который обеспечивает взаимодействие с сервером GraphQL. Реализация этого компонента приведена в приложении Л.

Компонент принимает дочерние элементы и оборачивает их в ApolloProvider, который предоставляет Apollo Client внутри приложения.

Сперва создается экземпляр httpLink с использованием функции createHttpLink. Этот экземпляр ссылается на URL-адрес сервера GraphQL, указанный в константе HTTP\_API\_URL.

Далее создается экземпляр wsLink с использованием GraphQLWsLink, который устанавливает соединение с WebSocket сервером GraphQL. Этот экземпляр ссылается на URL-адрес WebSocket сервера GraphQL, указанный в константе WS\_API\_URL. В параметрах соединения также передается токен доступа, полученный из глобального состояния Redux.

Затем создается authLink с использованием setContext. Этот authLink добавляет заголовок Authorization с токеном доступа в запросы Apollo Client.

Далее создается httpLinkWithAuth, который объединяет authLink и uploadLink для добавления заголовка Authorization к HTTP запросам.

Затем используется функция split для разделения ссылки на WebSocket и HTTP ссылки в зависимости от типа операции GraphQL (запрос или подписка). Если операция является подпиской, то используется wsLink, иначе используется httpLinkWithAuth.

Затем создается errorLink с использованием onError. Этот errorLink служит для перехвата ошибок, возникающих в запросах Apollo Client. В случае ошибки, она обрабатывается функцией errorHandler, возвращая текст ошибки. Затем ошибка передается в Redux для дальнейшей обработки.

И, наконец, создается экземпляр client с использованием ApolloClient. В нем используются созданные ранее errorLink и splitLink. Также определяется кэш InMemoryCache и заголовок Authorization со значением токена доступа.

Таким образом был рассмотрен компонент GraphQLProvider.

## **3.3 Выводы по разделу**

В результате разработки было создано полноценное веб-приложение, которое полностью удовлетворяет описанным требованиям. Для серверной части проекта была определена и описана структура проекта, включающая различные модули.

В рамках серверной части были разработаны основные составляющие, необходимые для полноценного функционирования веб-приложения. Были созданы модели данных, а также определены соответствующие схемы GraphQL для взаимодействия с данными. Кроме того, были разработаны резолверы, которые обрабатывают запросы клиента и возвращают соответствующие данные.

Для клиентской части проекта была определена структура проекта, включающая различные компоненты, файлы стилей и другие ресурсы. Были разработаны компоненты, которые взаимодействуют друг с другом и осуществляют запросы на сервер для получения данных.

# 4 Тестирование веб-приложения

ФИО

Подпись

Дата

Лист

1

БГТУ 04.00.ПЗ

Разраб.

Левша М.С.

Провер.

Гончар Е.А.

Н. контр.

Нистюк О.А.

Утв.

Смелов В.В.

4 Тестирование

веб-приложения

Лит.

Листов

5

74417043, 2024

## **4.1 Тестирование серверной части**

Для проверки корректности работы серверной части приложения были разработаны и выполнены юнит-тесты с использованием фреймворка Jest. Jest является фреймворком для тестирования JavaScript, который обеспечивает уверенность в правильной работе любого кода на JavaScript.

### **4.1.1 Unit-тестирование**

Unit-тестирование проверяет на корректность работу отдельных модулей исходного кода. Написание этих тестов гарантирует, что методы работают корректно, обрабатывают разные сценарии использования и защищены от ошибок.

Была реализована проверка на то, что userResolver при вызове метода blockUser, бросает BadRequestException при попытки заблокировать самого себя. Реализация данного теста приведена в листинге 4.1.

|  |
| --- |
| it('should throw BadRequestException if trying to block own account', async () => {  const invalidBlockUserInput: BlockUserInput = {  \_id: 'user\_id',  };  const invalidUser: JwtValidatedOutput = {  userId: 'user\_id',  };  await expect(  resolver.blockUser(invalidBlockUserInput, { req: { user: invalidUser } })  ).rejects.toThrow(BadRequestException);  }); |

Листинг 4.1 – Реализация негативного теста для проверки метода blockUser

Тест считается успешным, если метод выбросил исключение типа BadRequestException. Это был негативный пример работы метода blockUser, конечно же написан тест для проверки положительного результата. Положительный тест помогает убедиться, что метод blockUser функционирует правильно и обеспечивает корректную блокировку пользователя, а также отправку уведомления по электронной почте при успешном выполнении теста. Реализация положительного теста приведена в листинге 4.2.

|  |
| --- |
| it('should block user and send email notification', async () => {  jest.spyOn(userService, 'blockUser').mockResolvedValue(blockedUser);  jest.spyOn(emailNotificationService, 'sendEmailNotification').mockResolvedValue();  const result = await resolver.blockUser(blockUserInput, { req: { user } });  expect(userService.blockUser).toHaveBeenCalledWith(blockUserInput);  expect(emailNotificationService.sendEmailNotification).toHaveBeenCalledWith({  userEmail: blockedUser.email,  subject: 'Your access was updated',  message: 'Your account was blocked',  });  expect(result).toEqual(blockedUser);  }); |

Листинг 4.2 – Реализация положительного теста для проверки метода blockUser

Далее рассмотрим тестирование метода createUser класса UserService. Первый тест проверяет ситуацию, при которой администратор пытается создать уже существующего пользователя. Это важно, чтобы предотвратить создание дублирующихся записей в базе данных. Реализация данного теста приведена в листинге 4.3.

|  |
| --- |
| it('should throw a BadRequestException if the user already exists', async () => {  const createUserInput: CreateUserInput = {  nickname: 'testuser',  email: 'test@example.com',  password: 'password123',  role: 'user', };  jest.spyOn(userModel, 'findOne').mockResolvedValue({  \_id: '1', nickname: 'testuser', email: 'test@example.com', passwordHash: 'hashedPassword', isDeleted: false,  isBlocked: false,  role: 'user',  avatar: '', });  await expect(userService.createUser(createUserInput)).rejects.toThrow(  BadRequestException );  expect(userModel.findOne).toHaveBeenCalledWith({  nickname: 'testuser',  isDeleted: false, }); }); |

Листинг 4.3 – Реализация теста для проверки метода createUser

Тест будет успешным, если при вызове метода createUser класса UserService будет выброшено исключение типа BadRequestException. Это означает, что тест проверяет, что метод правильно обрабатывает ситуацию, когда пользователь с заданным никнеймом уже существует в базе данных.

Следующий тест проверяет ситуацию, при которой произошла ошибка в базе данных при сохранении пользователя. Данный тест важен, чтобы убедиться, что метод createUser правильно обрабатывает ошибки, возникающие при взаимодействии с базой данных. Если при сохранении пользователя возникнет ошибка, метод должен корректно обрабатывать эту ситуацию и возвращать соответствующее исключение, чтобы оно могло быть обработано на более высоком уровне приложения. Реализация теста представлена в листинге 4.4.

|  |
| --- |
| it('should throw a BadRequestException if there is an error creating the user', async () => {  const createUserInput: CreateUserInput = {  nickname: 'testuser',  email: 'test@example.com',  password: 'password123',  role: 'user',  };  jest.spyOn(userModel, 'findOne').mockResolvedValue(null);  jest.spyOn(cryptographyService, 'encryptPassword').mockResolvedValue('hashedPassword');  jest.spyOn(userModel, 'create').mockRejectedValue(new Error('Database error'));  await expect(userService.createUser(createUserInput)).rejects.toThrow(  BadRequestException  );  expect(userModel.findOne).toHaveBeenCalledWith({  nickname: 'testuser',  isDeleted: false,  });  expect(cryptographyService.encryptPassword).toHaveBeenCalledWith('password123');  expect(userModel.create).toHaveBeenCalledWith({  nickname: 'testuser',  email: 'test@example.com',  passwordHash: 'hashedPassword',  isDeleted: false,  isBlocked: false,  role: 'user',  avatar: '',  });  }); |

Листинг 4.4 – Реализация теста для проверки метода createUser

Тест считается успешным, если метод createUser выбросил исключение типа BadRequestException. Это означает, что метод правильно обрабатывает ошибочные ситуации, когда при создании пользователя возникает ошибка в базе данных.

Последний тест метода createUser проверяет успешное создание нового пользователя. В этом тесте моделируется ситуация, когда пользователя с указанным никнеймом еще нет в базе. Реализация данного теста приведена в листинге 4.5.

|  |
| --- |
| it('should create a new user successfully', async () => {  const createUserInput: CreateUserInput = {  nickname: 'testuser', email: 'test@example.com',  password: 'password123', role: 'user',};  jest.spyOn(userModel, 'findOne').mockResolvedValue(null);  jest.spyOn(cryptographyService, 'encryptPassword').mockResolvedValue('hashedPassword');  jest.spyOn(userModel, 'create').mockResolvedValue({  \_id: '1', nickname: 'testuser', email: 'test@example.com', passwordHash: 'hashedPassword', isDeleted: false,  isBlocked: false,  role: 'user',  avatar: '',});  const user = await userService.createUser(createUserInput);  expect(userModel.findOne).toHaveBeenCalledWith({  nickname: 'testuser', isDeleted: false,});  expect(cryptographyService.encryptPassword).toHaveBeenCalledWith('password123');  expect(userModel.create).toHaveBeenCalledWith({  nickname: 'testuser', email: 'test@example.com',  passwordHash: 'hashedPassword', isDeleted: false,  isBlocked: false, role: 'user', avatar: '', });  expect(user).toEqual({\_id: '1', nickname: 'testuser',  email: 'test@example.com', passwordHash: 'hashedPassword', isDeleted: false, isBlocked: false, role: 'user',  avatar: '', });  }); |

Листинг 4.5 – Реализация теста для проверки метода createUser

Тест гарантирует, что метод createUser правильно создает и сохраняет нового пользователя в базе данных при отсутствии пользователя с заданным никнеймом.

## **4.2 Ручное тестирование**

Для обеспечения надежности и качества взаимодействия между клиентской и серверной частями приложения было проведено тщательное ручное тестирование. В ходе тестирования были проверены основные функциональные возможности. Результаты проведенного ручного тестирования представлены в таблице 4.1.

Таблица 4.1 – Таблица тест-кейсов

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Шаги | Ожидаемый  результат | Результат теста |
| 1 | 2 | 3 | 4 |
| Создание пользователя с уже существующим никнеймом | 1. На навигационной панели, вкладке «Admin» нажать на кнопку создания пользователя.  2. Заполнить поля, в поле никнейма ввести уже существующий.  3. Нажать кнопку «Save». | Отображение всплывающего окна с ошибкой создания пользователя | Пройден |

Продолжение таблицы 4.1

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| Создание пользователя с некорректной почтой | 1. На навигационной панели, вкладке «Admin» нажать на кнопку создания пользователя.  2. Заполнить поля, в поле почты  ввести некорректную почту.  3. Нажать кнопку «Save». | Отображение всплывающего окна с ошибкой создания пользователя | Пройден |
| Создание нового чата | 1. На навигационной панели, вкладке «Chat» нажать на кнопку создания нового чата.  2. Выбрать нужного получателя, написать сообщение.  3. Нажать кнопку «Send». | Чат создан, сообщение отправлено | Пройден |
| Создание задачи с некорректным заполнением поля цены | 1. На навигационной панели, вкладке «Tasks» нажать на кнопку создания задачи.  2. Ввести в поле цены нечисловое значение. | Поле остается пустым | Пройден |
| Создание задачи без наличия разработчиков в системе | 1. На навигационной панели, вкладке «Tasks» нажать на кнопку создания задачи. | Отображение всплывающего окна с ошибкой создания задачи | Пройден |
| Создание задачи без наличия клиентов в системе | 1. На навигационной панели, вкладке «Tasks» нажать на кнопку создания задачи. | Отображение всплывающего окна с ошибкой создания задачи | Пройден |
| Изменение у пользователя почты на некорректную | 1. На навигационной панели, вкладке «Admin» нажать на кнопку изменения пользователя.  2. Заполнить поля, в поле почты  ввести некорректную почту.  3. Нажать кнопку «Save». | Отображение всплывающего окна с ошибкой изменения пользователя | Пройден |

Таким образом, было проведено ручное тестирование приложения, было сделано как негативное тестирование функционала приложения, так и позитивное. Приложение во всех ситуациях отработало корректно.

## **4.3 Выводы по разделу**

В данном разделе были рассмотрены реализации написанных unit тестов серверной части. Кроме того, было проведено ручное тестирование, в ходе которого были проверены основные функции веб-приложения.

Важно отметить, что тестирование приложения выполнялось не только после завершения разработки, но и на протяжении всего процесса создания. На этапе разработки производились проверки созданных компонентов, их работоспособности и взаимодействия друг с другом. Полученные в ходе тестирования результаты позволяют сделать вывод, что приложение работает корректно.

# 5 Руководство пользователя

ФИО

Подпись

Дата

Лист

1

БГТУ 05.00.ПЗ

Разраб.

Левша М.С.

Провер.

Гончар Е.А.

Н. контр.

Нистюк О.А.

Утв.

Смелов В.В.

5 Руководство пользователя

Лит.

Листов

8

74417043, 2024

В данном разделе представлено подробное руководство по использованию веб-приложения для четырех основных ролей: «Администратор», «Менеджер», «Разработчик» и «Клиент». Руководство поможет пользователям ознакомиться с функциональностью приложения. Скриншот работы приложения представлен в приложении Д.

## **5.1 Роль «Администратор»**

Администратор в приложении выполняет широкий спектр задач по администрированию пользователей. Первое, что видит администратор при запуске приложения, – это страница для входа в приложение, как показано на рисунке 5.1.

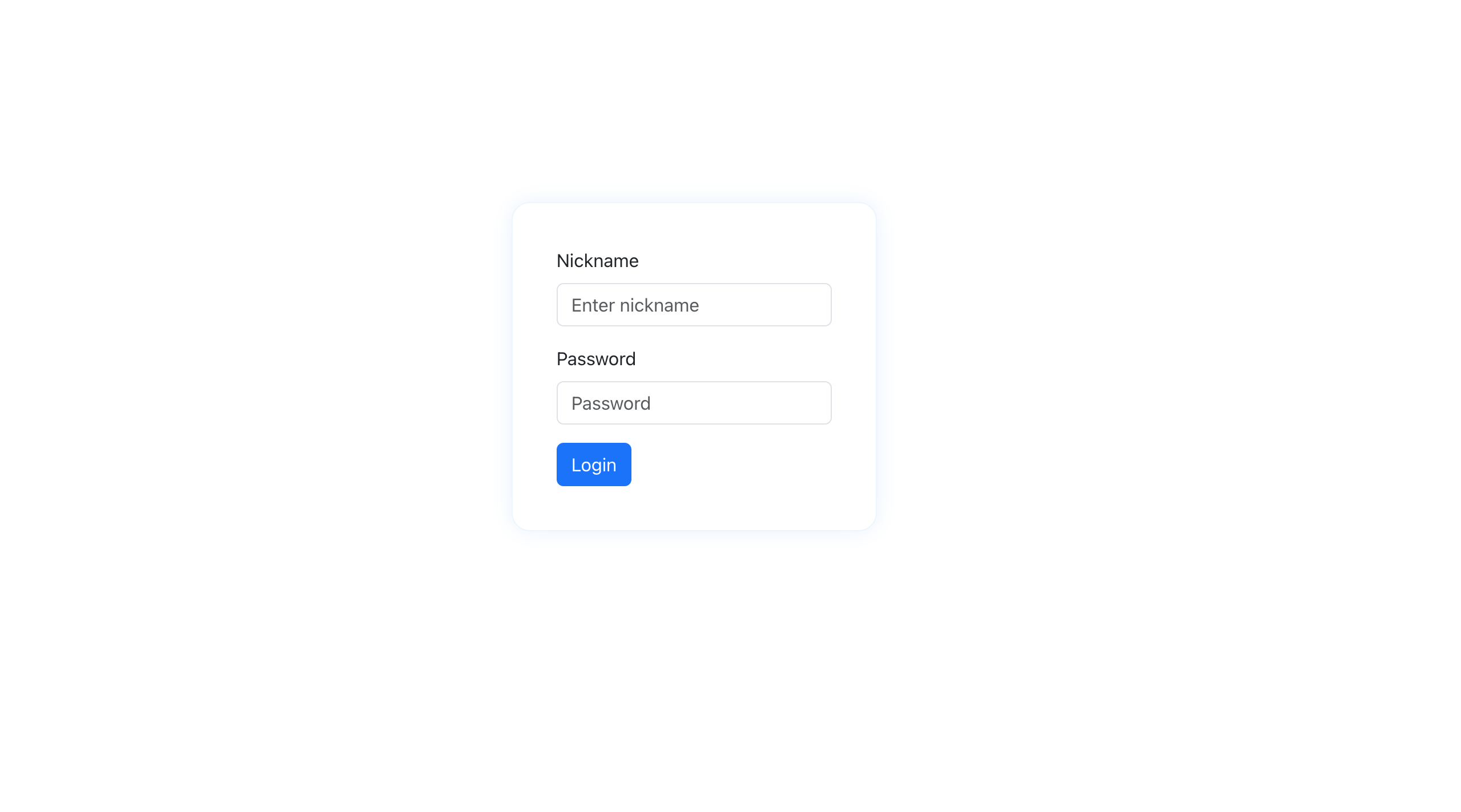


Рисунок 5.1 – Страница входа в приложение

Чтобы администратор мог попасть в систему, ему необходимо ввести свои учетные данные - никнейм и пароль. После успешного входа в приложение, администратор попадает на вкладку чатов. Администратор может использовать эту вкладку для создания новых чатов с другими пользователями, а также для написания сообщений в уже существующих чатах.

Таким образом, вкладка чатов является центральным местом, откуда администратор может управлять всеми коммуникациями в приложении. Внешний вид этой вкладки представлен на рисунке 5.2.

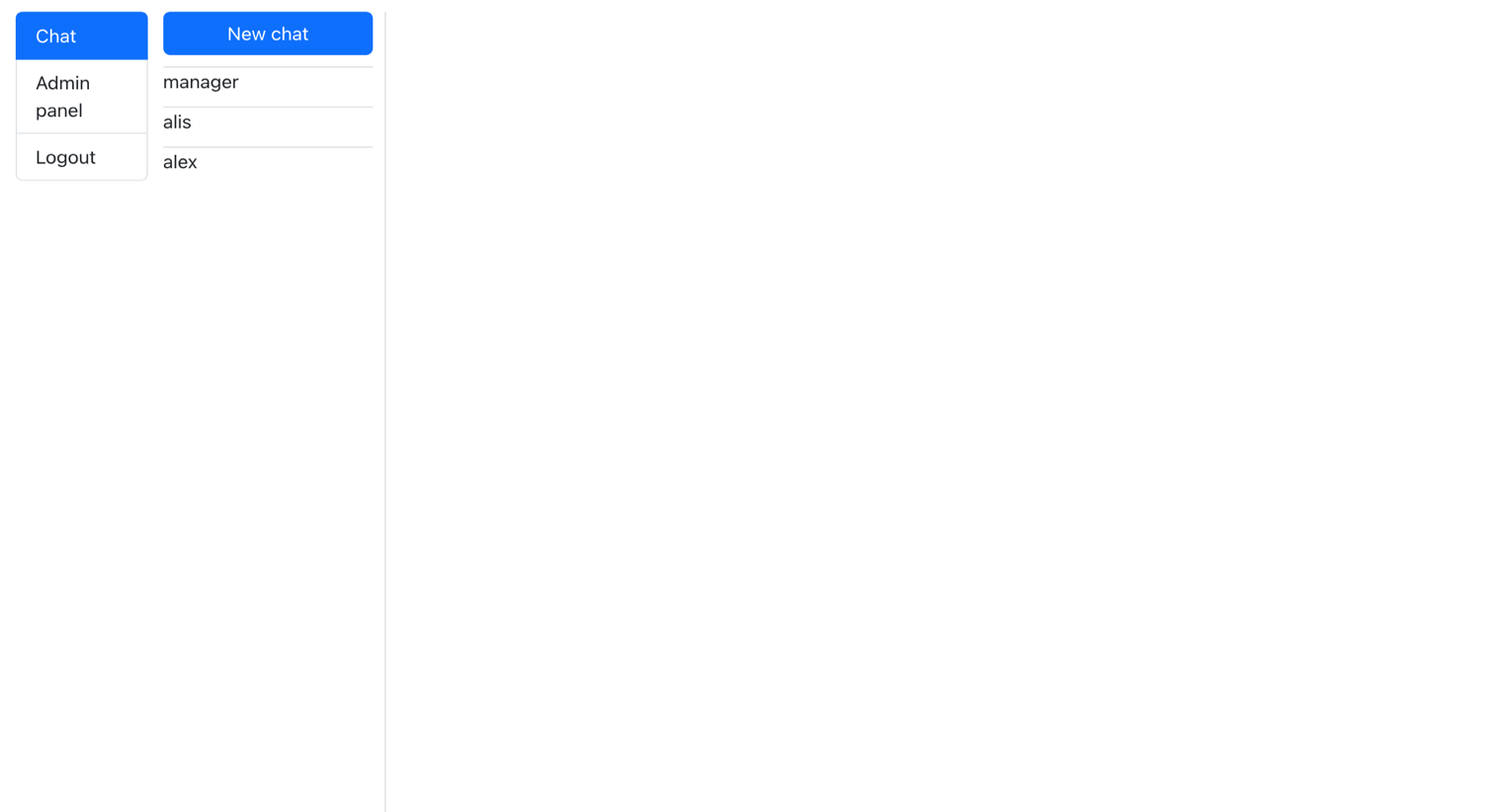


Рисунок 5.2 – Вкладка чатов

Для создания нового чата администратор должен нажать на кнопку «New chat» на странице чатов. После нажатия этой кнопки появляется модальное окно, в котором администратор должен указать получателя нового чата и написать приветственное сообщение. Внешний вид этого модального окна показан на рисунке 5.3.

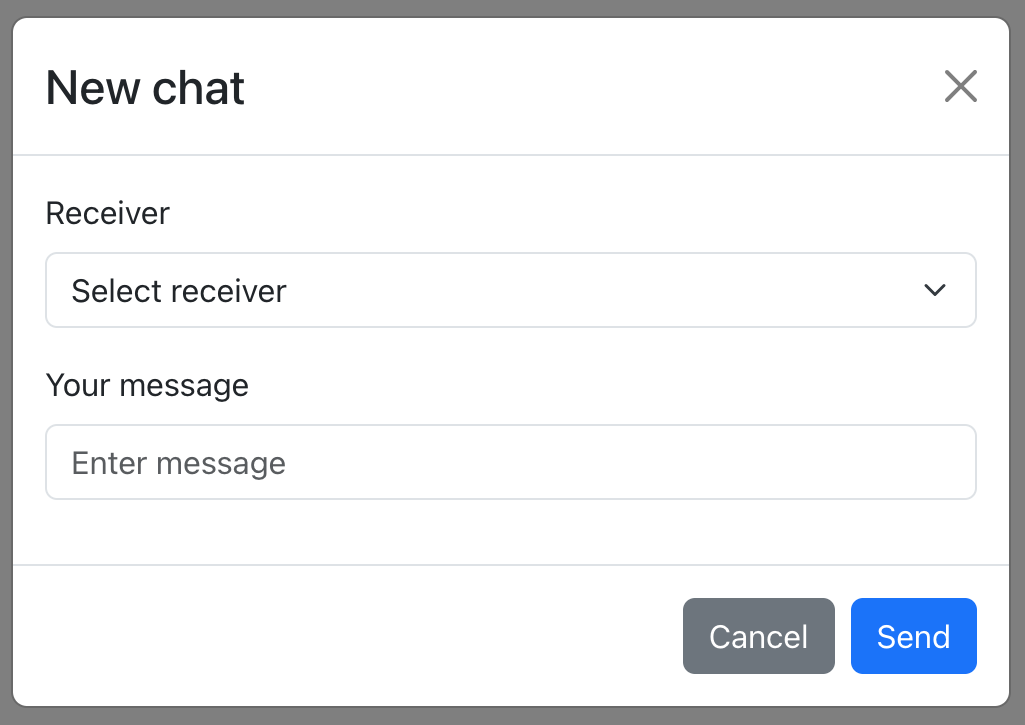


Рисунок 5.3 – Модальное окно создания чата

После того, как администратор заполняет необходимые поля в модальном окне и нажимает кнопку «Send», новый чат будет незамедлительно создан и отображен в общем списке чатов на соответствующей странице. Это позволяет пользователям быстро и эффективно организовывать новые каналы коммуникации по мере необходимости. На рисунке 5.4 изображен сам чат с другим пользователем.

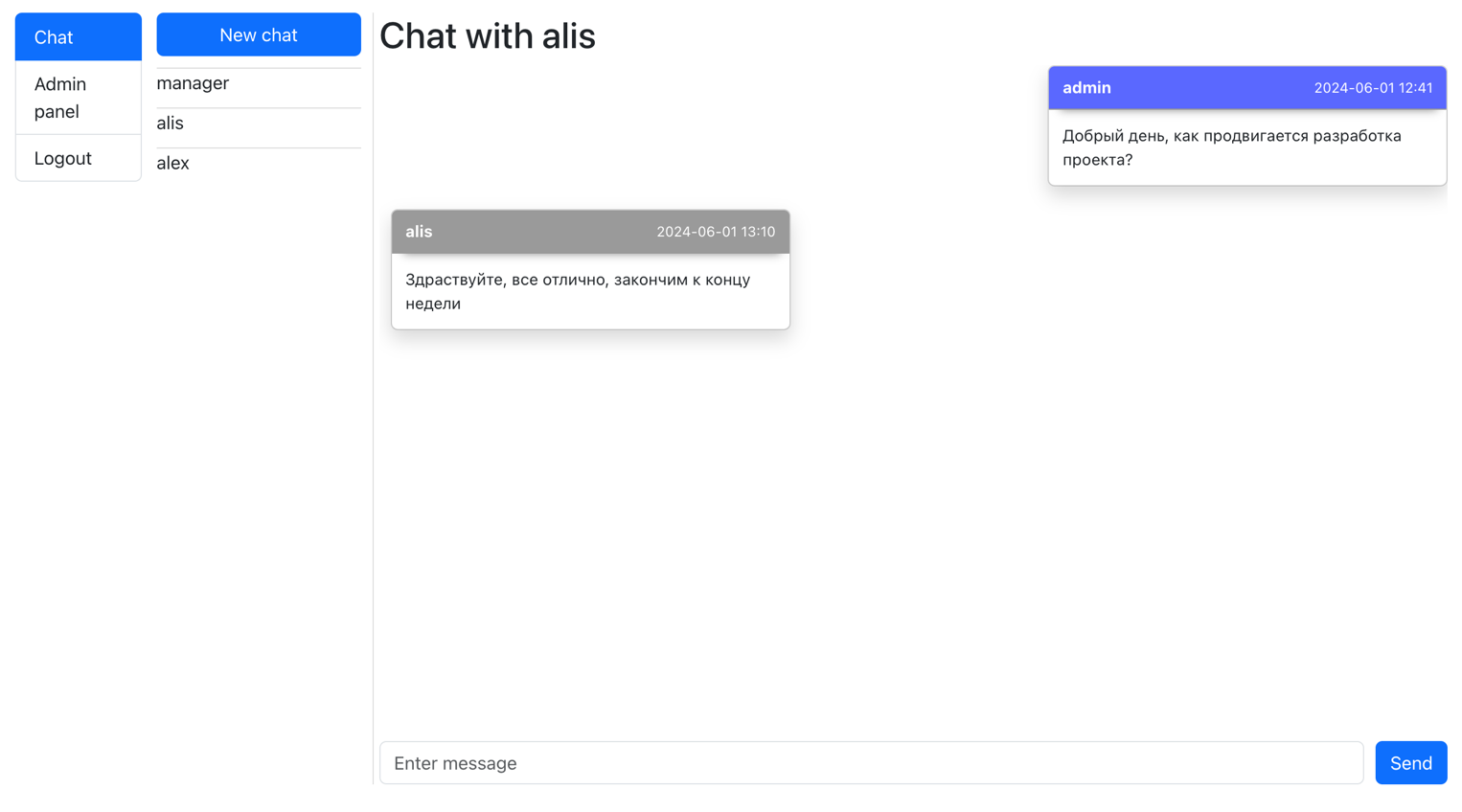


Рисунок 5.4 – Вкладка чата

На странице чата отображается никнейм пользователя, с которым происходит общение. В верхней части каждого сообщения собеседника указан его никнейм, а также время отправки сообщения. Сами сообщения собеседника выделены серым цветом, для лучшей визуальной дифференциации. Что касается собственных сообщений пользователя, они окрашены в фиолетовый цвет.

В нижней части страницы чата расположено поле для ввода текста нового сообщения. Здесь пользователь может напечатать свой ответ или новое сообщение. Рядом с полем ввода находится кнопка отправки сообщения, которая позволяет мгновенно передать сообщение в чат.

Такое интуитивно понятное и упорядоченное представление информации в чате способствует эффективному и удобному общению между пользователями. Четкое визуальное разграничение реплик, а также наличие необходимых элементов управления делают процесс коммуникации в приложении максимально комфортным.

Далее рассмотрим панель администратора. Она является важным инструментом, который позволяет администратору эффективно управлять пользователями приложения. Через эту панель администратор может создавать новых пользователей, удалять существующих, изменять их данные, а также блокировать и разблокировать доступ к системе в случае необходимости.

Данный интуитивно понятный интерфейс, представленный на рисунке 5.5, позволяет администратору эффективно контролировать учетные записи и обеспечивать надлежащую работу всего приложения.

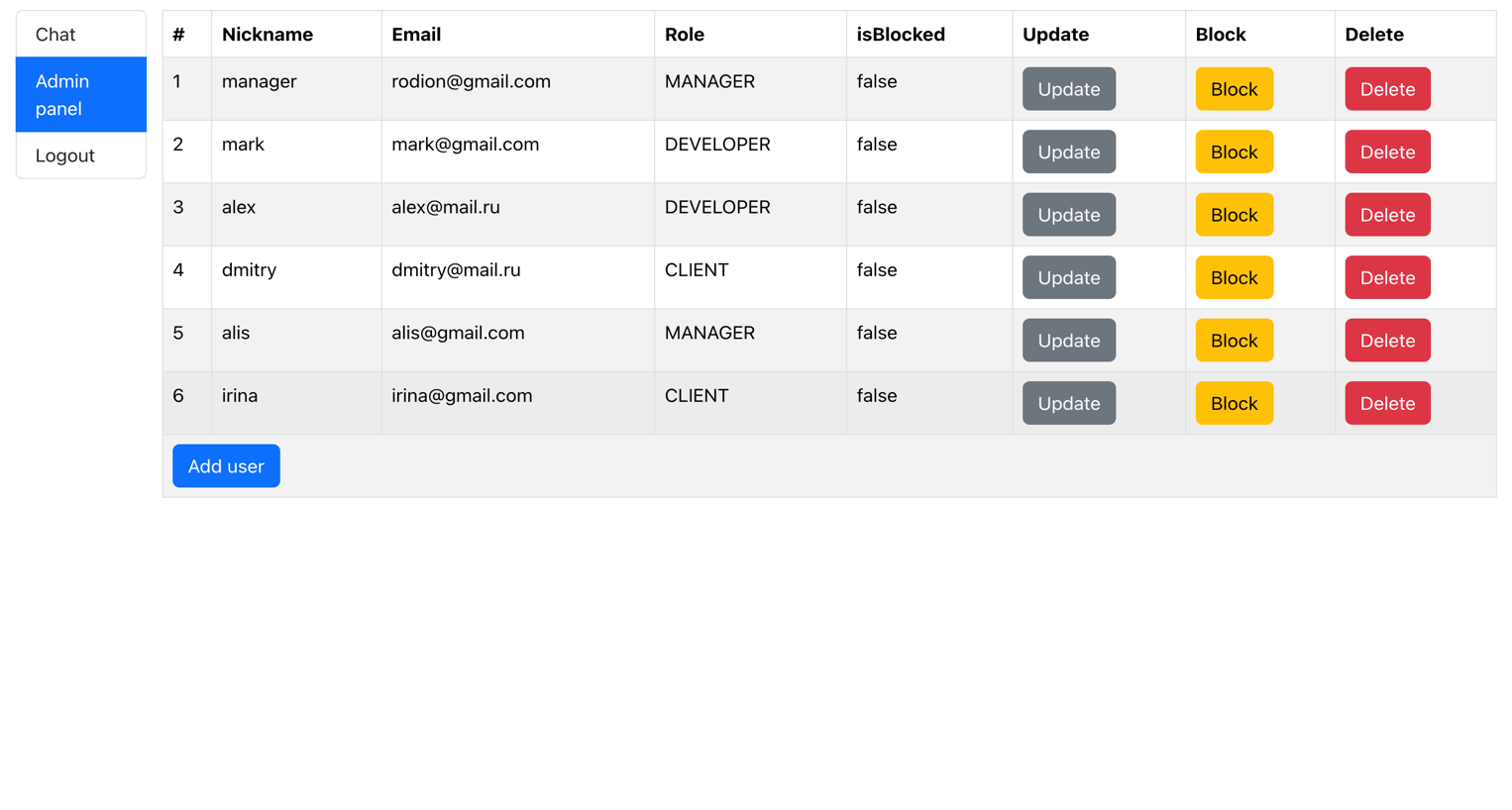


Рисунок 5.5 – Вкладка панели администратора

Прежде всего, через панель администратора можно создавать новых пользователей, вводя необходимую регистрационную информацию. Это дает возможность быстро расширять круг пользователей приложения, подключая к системе новых сотрудников или клиентов по мере необходимости. Модальное окно для создания нового пользователя представлено на рисунке 5.6.

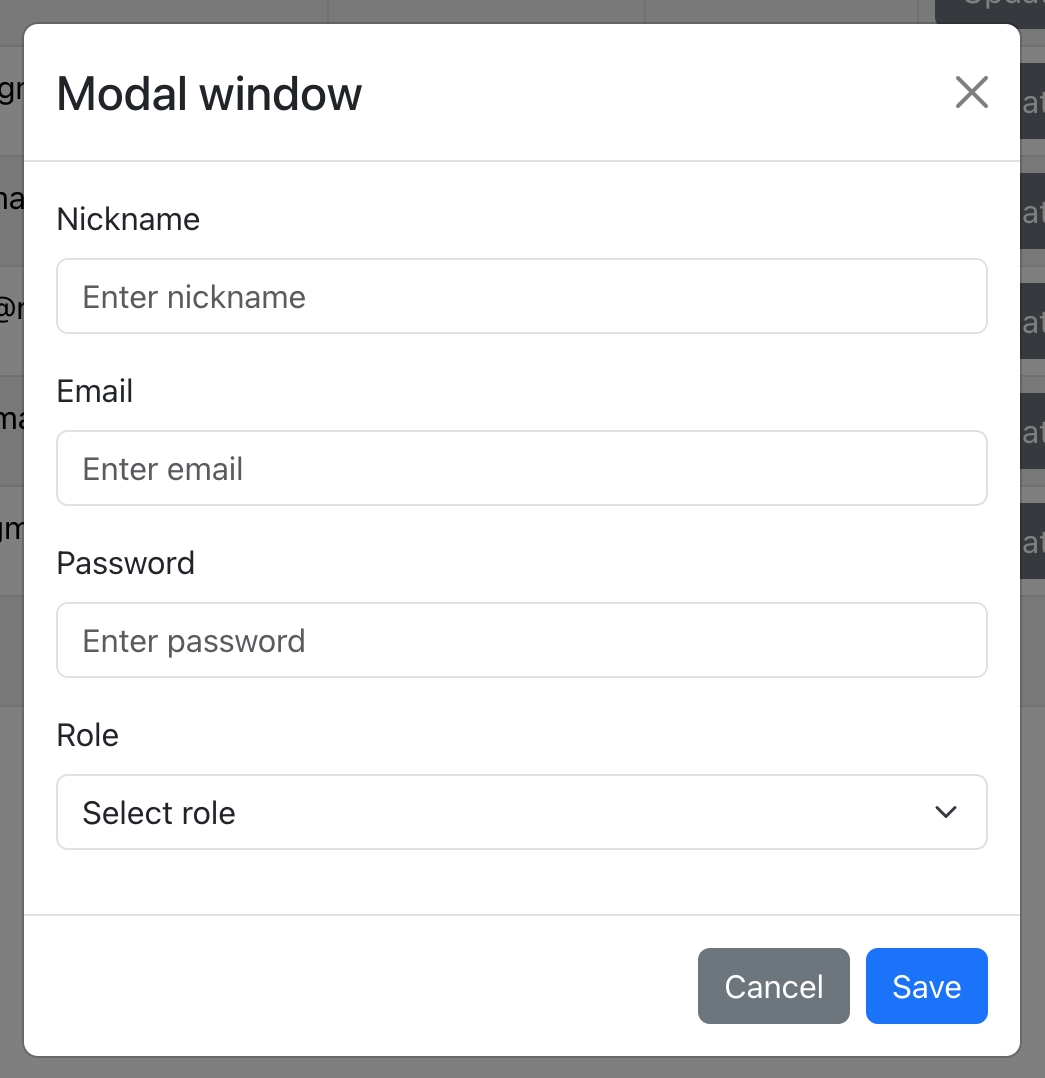


Рисунок 5.6 – Модальное окно создания пользователя

После выполнения любых действий с учетной записью пользователя, таких как создание, удаление, блокировка или разблокировка, система автоматически отправляет соответствующее уведомление на электронную почту этого пользователя.

Когда администратор создает нового пользователя, на указанный в регистрационной форме адрес электронной почты отправляется письмо, содержащее всю необходимую информацию об учетной записи – имя пользователя и пароль, для доступа к веб-приложению. Такое уведомление позволяет новому пользователю оперативно получить данные для входа в систему и начать работу. Пример информационного сообщения при создании пользователя приведен на рисунке 5.7.

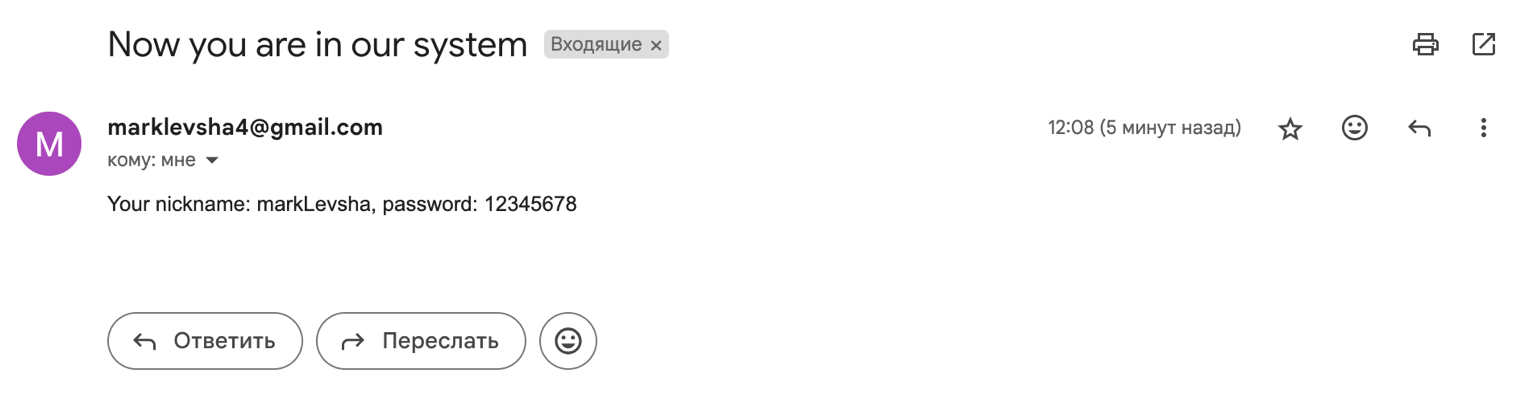


Рисунок 5.7 – Пример информационного сообщения при создании пользователя

Таким образом была рассмотрена роль «Администратор».

## **5.2 Роль «Клиент»**

Клиент может войти в систему с помощью данных, которые пришли ему на почту, после того как администратор создаст его пользователя. Для входа используется та же страница, что и у администратора.

Клиент может просматривать канбан-доску, на которой отображены его заказы и их статусы. Внешний вид канбан-доски представлен на рисунке 5.8.

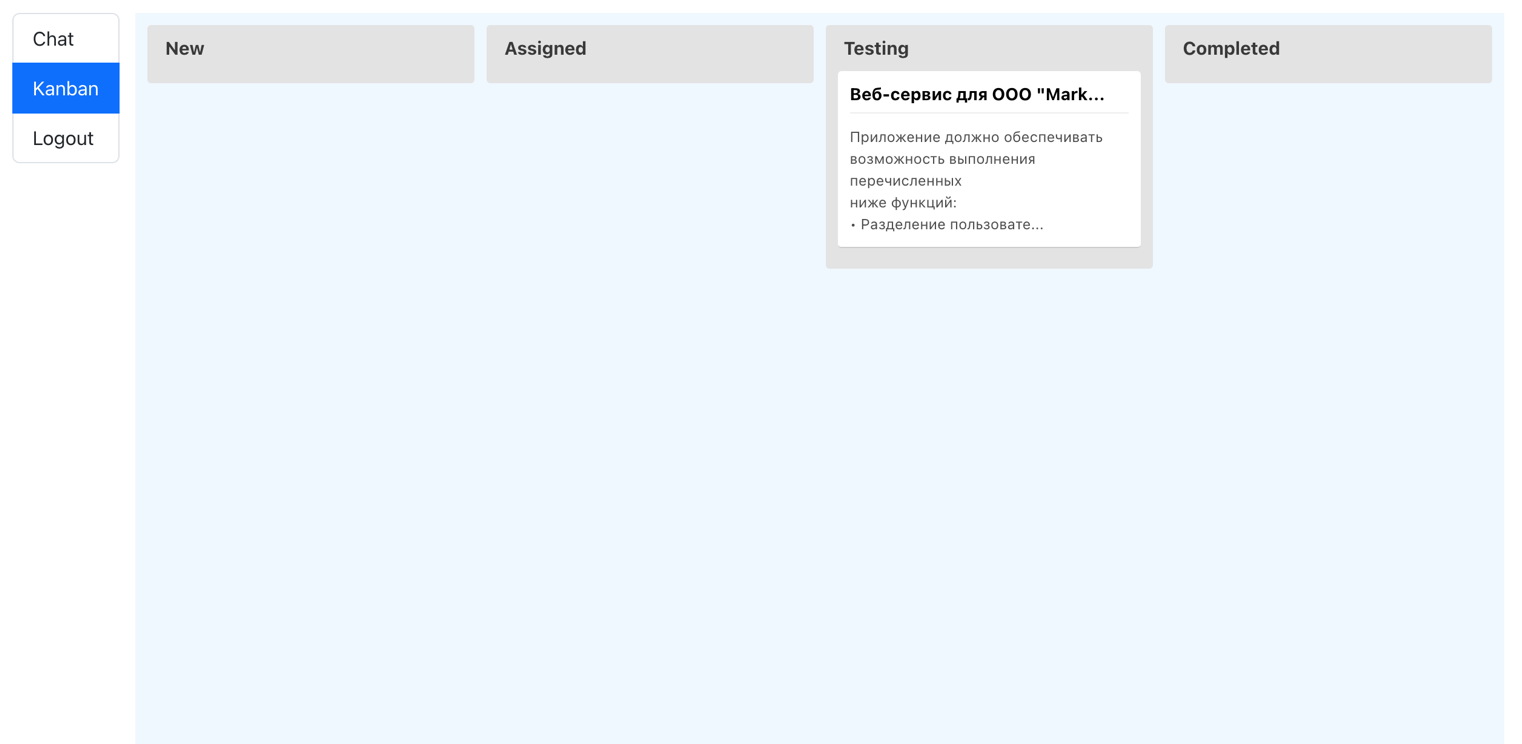


Рисунок 5.8 – Внешний вид канбан-доски

Интерфейс чата для клиента идентичен тому, что доступен администратору. Клиент может создавать новые чаты, отправлять сообщения, а также получать новые сообщения в режиме реального времени.

## **5.3 Роль «Разработчик»**

Также как и у клиента, у разработчика есть доступ к канбан-доске, на которой отображены закрепленные за ним задачи.

У разработчика есть возможность детально изучить описание закрепленных за ним задач и писать комментарии к ним для уточнения деталей и обновления статуса готовности той, или иной, части приложения. Пример взаимодействия пользователей с помощью комментариев показан на рисунке 5.9.

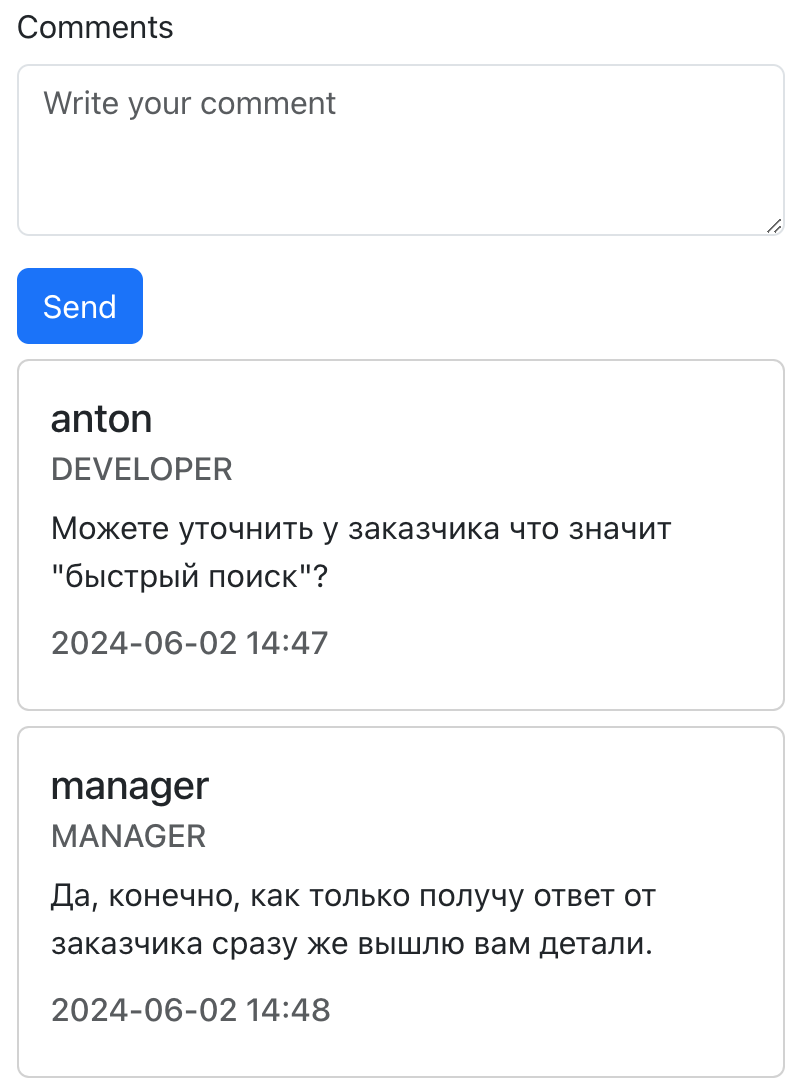


Рисунок 5.9 – Пример взаимодействия пользователей с помощью комментариев

Аналогично клиентам и администраторам, разработчики имеют доступ к функционалу интерактивного чата, встроенного в систему. Доступ разработчиков к функционалу интерактивного чата позволяет им создавать и поддерживать более эффективное и персонализированное взаимодействие с менеджерами и клиентами.

## **5.4 Роль «Менеджер»**

Менеджер обладает возможностями как клиента, так и разработчика, а также расширенным функционалом для управления задачами. Выступая связующим звеном между клиентами и разработчиками и обеспечивая эффективное управление и достижение целей всех участников взаимодействия.

Главной функцией менеджера является создание новых задач в системе и распределение их между разработчиками. При создании задачи менеджер заполняет ее основные характеристики - название, описание, сроки выполнения, цену и т.д. Затем он назначает исполнителей из числа доступных разработчиков, определяя, кто будет нести ответственность за реализацию данной задачи. Вкладка задач приведена на рисунке 5.10.



Рисунок 5.10 – Вкладка задач

На вкладке «Задачи» в приложении менеджер имеет доступ к полному списку всех существующих задач. Здесь он может не только просматривать текущее состояние каждой задачи, но и выполнять ряд операций по их управлению.

Во-первых, менеджер может создавать новые задачи. Для этого он нажимает на соответствующую кнопку и заполняет форму с необходимыми данными - название, описание, сроки, цена, назначение исполнителей и т.д. Созданная таким образом задача сразу же появляется в общем списке.

Во-вторых, менеджер может при необходимости удалять существующие задачи. Это может потребоваться, например, при отмене или закрытии проектов, изменении требований заказчика и других ситуациях, когда задача теряет свою актуальность. Для удаления менеджер нажимает на кнопку «Delete» у нужной задачи.

Когда менеджер переходит на вкладку с деталями определенной задачи, перед ним открывается подробное представление этой задачи. Здесь собрана вся ключевая информация, необходимая для понимания и управления данной задачей.

Таким образом, переход на страницу конкретной задачи дает менеджеру полный доступ ко всей необходимой информации. Он может не только ознакомиться с деталями задачи, но и активно взаимодействовать с разработчиком, оставляя комментарии к задаче и отвечая на вопросы. Пример задачи представлен на рисунке 5.11.

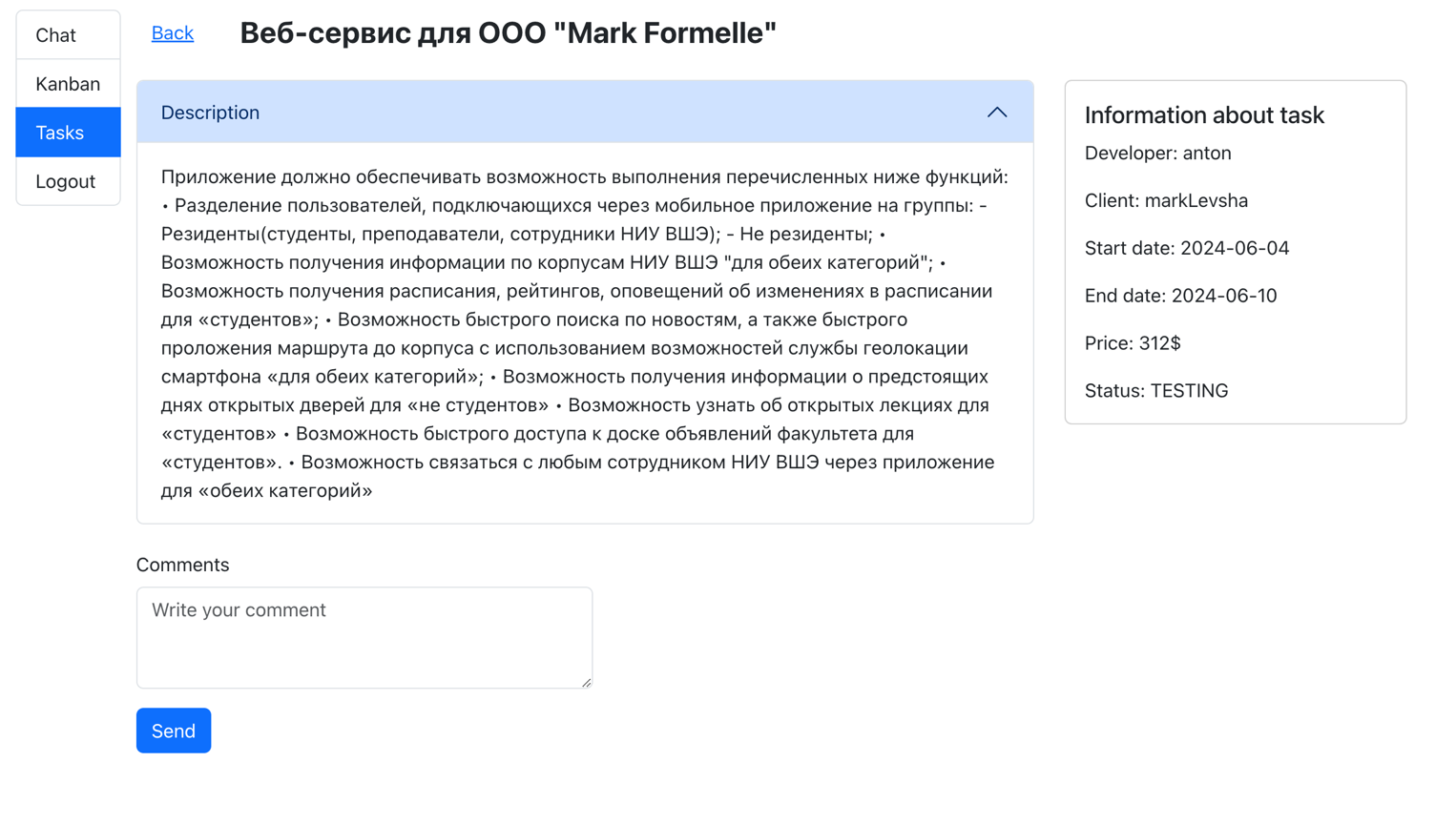


Рисунок 5.11 – Пример задачи

Важно отметить, что менеджер может перетаскивать задачи на канбан-доске для изменения их статусов. Такой drag-and-drop механизм управления задачами на канбан-доске обеспечивает быструю и наглядную визуализацию текущего статуса задач. Кроме того, эта функциональность упрощает коммуникацию с командой, позволяя всем участникам в режиме реального времени отслеживать изменения.

## **5.5 Выводы по разделу**

В разделе представлены основные моменты работы с веб-приложением для администратора, менеджера, разработчика и клиента. Представленные скриншоты и подробные описания демонстрируют полноту и удобство взаимодействия пользователей различных ролей с веб-приложением, способствуя повышению эффективности управления процессами разработки.

Интерфейс приложения интуитивно понятен, поэтому у пользователя не должно возникнуть трудностей с его эксплуатацией.

# 6 Технико-экономическое обоснование проекта

ФИО

Подпись

Дата

Лист

1

БГТУ 06.00.ПЗ

Разраб.

*Левша М.С.*

Провер.

*Гончар Е.А.*

Консульт.

*Евлаш А.И.*

Н. контр.

Нистюк О.А.

Утв.

Смелов В.В.

6 Технико-экономическое

обоснование проекта

Лит.

Листов

8

74417043, 2024

## **6.1 Общая характеристика разрабатываемого программного средства**

Цель данной дипломной работы заключается в создании программного средства для организации взаимоотношений с клиентами, которое поможет клиентам отслеживать этапы разработки своих проектов, а исполнителям даст возможность удобно управлять процессом разработки. Программное средство представляет собой клиент-серверное приложение.

Разработанный программный продукт позволяет менеджерам создавать проекты, назначать на них разработчиков и клиентов, а клиентам отслеживать статусы выполнения проектов и поддерживать связь с менеджером посредством чата.

Во время разработки дипломного проекта использовался фреймворк Nest.js для написания серверной части приложения и библиотека React для написания клиентской части приложения. Данное веб-приложение разработано для последующего использования в коммерческих целях.

Стратегия монетизации предполагает разовую покупку программного средства.

## **6.2 Исходные данные для проведения расчетов и маркетинговый анализ**

Источниками исходных данных для расчетов выступают действующие законы и нормативно-правовые акты [17]. Исходные данные для расчета приведены в таблице 6.1.

Таблица 6.1 – Исходные данные для расчета

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование показателя | Единица  измерения | Условные  обозначения | Норматив |
| Численность разработчиков | чел. | Чр | 6,0 |
| Норматив дополнительной заработной платы | % | Ндз | 15,0 |
| Ставка отчислений в Фонд социальной защиты населения | % | Нфсзн | 34,0 |
| Ставка отчислений в БРУСП «Белгосстрах» | % | Нбгс | 0,6 |
| Норматив прочих затрат | % | Нпз | 35 |
| Норматив накладных расходов | % | Нн.р. | 40,0 |
| Норматив расходов на сопровождение и адаптацию | % | Нрса | 17,0 |

Данные из этой таблицы будут использоваться далее в вычислениях. Для того, чтобы проанализировать стоимость разработки программного обеспечения в первую очередь необходимо рассмотреть стоимость разработки приложений-аналогов. Для этого было выбрано 3 приложения-аналога.

Каждое из этих приложений обладает своим уникальным функционалом и предлагает различные услуги, что делает их идеальными кандидатами для сравнительного анализа. Специфика каждого из выбранных приложений-аналогов позволяет оценить, как различные аспекты разработки влияют на итоговую стоимость проекта. Результаты проведения анализа стоимости представлены в таблице 6.2.

Таблица 6.2 – Анализ стоимости разработки

|  |  |  |  |
| --- | --- | --- | --- |
| Продукт-аналог | Источник | Стоимость, руб | Примечание |
| Веб-приложение для организации взаимоотношений с клиентами | https://twenty.com/ | 27 000 | Мало функционала, отсутствует панель администратора, отсутствует чат, отсутствует отображения статистики |
| Веб-приложение для организации взаимоотношений с клиентами | https://www.bitrix24.by/ | 40 000 | Наличие панели администратора, наличие статистики,  наличие чата, наличие диска для хранения файлов, наличие сквозной аналитики |
| Веб-приложение для организации взаимоотношений с клиентами | https://clickup.com/ | 35 000 | Комлексное решение, наличие календаря, наличие чата, наличие статистики |

В процессе проведения маркетингового анализа была определена стоимость разработки аналогичного программного продукта для организации взаимоотношений с клиентами. Средняя стоимость разработки такого продукта оценивается в диапазоне от 27 000 до 40 000 рублей. Следовательно, общая стоимость разработки данного программного средства, выбранного в качестве базы для сравнения, составляет 25 000 рублей. Разрабатываемый продукт не только дешевле и выгоднее самого доступного аналога, но и превосходит его по функциональности. Веб-приложение аналога имеет меньшие возможности: пользователи могут только просматривать информацию о партнерах и клиентах и создавать заметки. В то же время, данное программное средство имеет чат и панель администратора.

## **6.3 Обоснование цены программного средства**

### **6.3.1 Расчет затрат рабочего времени на разработку программного средства**

В таблице 6.3 указаны в укрупненном виде все работы, реально выполненные для создания, указанного в дипломной работе программного средства, и количество рабочих дней, реально потраченных для выполнения этих работ. Были учтены затраты на разработку программного средства, поиск нужных компонентов платформы, с сервисами которой будет интегрировано программное средство, а также тестирование и написание руководства пользователя.

Таблица 6.3 – Затраты рабочего времени на разработку ПС

|  |  |  |
| --- | --- | --- |
| Содержание работ | Исполнитель | Затраты рабочего времени, часов |
| Исследование и анализ требований | Бизнес-аналитик | 6 |
| Анализ аналогичных приложений | Бизнес-аналитик | 10 |
| Подготовка технического задания | Бизнес-аналитик | 8 |
| Разработка UI/UX дизайна | Дизайнер | 40 |
| Создание архитектуры приложения | Системный архитектор | 52 |
| Проектирование базы данных | Системный архитектор | 32 |
| Разработка серверной части приложения | Программист | 150 |
| Разработка клиентской части приложения | Программист | 250 |
| Написание тестов для серверной части приложения | Тестировщик | 50 |
| Тестирование бизнеслогики приложения | Тестировщик | 20 |
| Написание руководства пользователя | Бизнес-аналитик | 5 |
| Всего |  | 623 |

Таким образом, затраты рабочего времени на разработку составляют 623 часа.

### **6.3.2 Расчет основной заработной платы**

После определения часовых ставок и трудозатрат исполнителей определяются основные заработные платы всех исполнителей. Основная заработная плата отдельного специалиста будет рассчитываться по формуле 6.1. Результаты подсчетов представлены в таблице 6.4.

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

где Соз – основная заработная плата, руб.;

Траз – трудоемкость (чел./час.);

Сзп – средняя часовая ставка руб./час.

Таблица 6.4 – Расчет основной заработной платы специалистов

|  |  |  |  |
| --- | --- | --- | --- |
| Исполнитель | Затраты рабочего  времени, часов | Средняя часовая ставка, руб./час | Основная заработная плата, руб. |
| Бизнес-аналитик | 29 | 14,88 | 431,52 |
| Системный архитектор | 84 | 20,83 | 1 749,72 |
| Программист | 400 | 9,52 | 3 808 |
| Тестировщик | 70 | 7,74 | 541,8 |
| Дизайнер | 40 | 7,14 | 285,6 |
| Всего | 623 |  | 6 816,64 |

Таким образом, при разработке программного средства основная заработная плата бизнес-аналитика составит 431,52 руб., системного архитектора – 1 749,72 руб., программиста – 3 808 руб., тестировщика – 541,8 руб., а дизайнера – 285,6 руб. Суммарная основная заработная плата всех специалистов проекта по разработке веб-приложения для организации взаимоотношений с клиентами составит 6 816,64.

### **6.3.3 Расчет дополнительной заработной платы**

Дополнительная заработная плата на конкретное программное средство включает выплаты, предусмотренные законодательством о труде, и определяется по нормативу в процентах к основной заработной плате по формуле (6.2).

|  |  |
| --- | --- |
|  | (6.2) |

где Соз– основная заработная плата, руб.;

Ндз – норматив дополнительной заработной платы, %.

руб.

По итогам подсчетов я получил сумму дополнительной заработной платы за все время разработки проекта.

### **6.3.4 Расчет отчислений в Фонд социальной защиты населения и по обязательному страхованию**

Отчисления в Фонд социальной защиты населения (ФСЗН) и по обязательному страхованию от несчастных случаем на производстве и профессиональных заболеваний определяются в соответствии с действующими законодательными актами Республики Беларусь по нормативу в процентном отношении к фонду основной и дополнительной зарплаты исполнителей.

Отчисления в Фонд социальной защиты 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 47,03 руб., а в фонд социальной защиты населения –2 665,30 руб.

### **6.3.5 Расчет суммы прочих прямых затрат**

Сумма прочих затрат Спз определяется как произведение основной заработной платы исполнителей на конкретное программное средство Соз на норматив прочих затрат в целом по организации Нпз, и находится по формуле 6.5.

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

Все данные необходимые для вычисления есть, поэтому можно определить сумму прочих затрат, за время разработки проекта.

Спз = ⋅ 25 / 100 = 1 704,16 руб.

В итоге получена сумма прочих прямых затрат равная 1 704,16 руб.

### **6.3.6 Расчет суммы накладных расходов**

Сумма накладных расходов *. –* произведение основной заработной платы исполнителей на конкретное программное средствона норматив накладных расходов в целом по организации *.*,по формуле 6.6.

|  |  |  |
| --- | --- | --- |
|  |  | (6.6) |

Сумма накладных расходов составит:

В итоге была получена сумму накладных расходов на сумму 3 067,49 руб.

### **6.3.7 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 6.7.

|  |  |
| --- | --- |
| Ср = Соз + Сдз + Сфсзн + Сбгс + Спз + Сн.р.. | (6.7) |

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

Ср = 6 816,64+ ++ + 1 704,16 + = 15 323,11 руб.

Сумма расходов на разработку программного средства была вычислена на основе данных, рассчитанных ранее в данном разделе, и составила 15 323,11 рублей.

### **6.3.8 Расходы на сопровождение и адаптацию**

Сумма расходов на сопровождение и адаптацию программного средства Срса определяется как произведение суммы расходов на разработки на норматив расходов на сопровождение и адаптацию Нрса, и находится по формуле 6.8.

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

= 15 323,11 ⋅ 17 / 100 = 2 604,93 руб.

Сумма расходов на сопровождение и адаптацию была вычислена на основе данных, рассчитанных ранее в данном разделе.

Все проведенные выше расчеты необходимы для вычисления полной себестоимости программного средства.

### **6.3.9 Полная себестоимость**

Полная себестоимость Сп определяется как сумма двух элементов: суммы расходов на разработку Ср и суммы расходов на сопровождение и адаптацию программного средства Срса.

Полная себестоимость Сп вычисляется по формуле 6.9.

|  |  |  |
| --- | --- | --- |
|  |  | (6.9) |

Сп = 15 323,11 + 2 604,93 = 17 928,04 руб.

Полная себестоимость программного средства была вычислена на основе данных, рассчитанных ранее в данном разделе.

### **6.3.10 Определение цены, оценка эффективности**

В ходе маркетингового анализа, описанного в пункте 6.2, было установлено, что полная стоимость разработки приложения составляет 25 000 рублей. На основе этой цены и рассчитанной себестоимости разработки устанавливается планируемая отпускная цена разрабатываемого приложения.

ЦсНДС = 25 000 руб.

Сумма налога на добавленную стоимость НДС определяется как произведение цены разработчика на ставку НДС Нндс и находится по формуле ниже.

. (6.10)

Цена разработки программного средства Цр без налогов определяется как разность двух элементов: отпускной цены ЦсНДС минус сумма налога на добавленную стоимость НДС и вычисляется по формуле ниже.

 (6.11)

Прибыль от реализации программного средства Ппс определяется как разность цены разработки программного средства без налога Цр минус полную себестоимость продукта Сп и находится по формуле ниже.

 (6.12)

Организация не является резидентом ПВТ, поэтому рассчитываем чистую прибыль по формуле 6.13.

|  |  |
| --- | --- |
|  | (6.13) |

где НП – ставка налога на прибыль, %.

Рентабельность считается как деление прибыли от реализации продукта Ппс делить на полную себестоимость продукта Сп и вычисляется по формуле ниже.

 (6.14)

НДС = 25 000 ⋅ 20 / 120 = 4 166,66 руб.;

Цр = 25 000 – 4 166,66 = 20 833,34 руб.;

Ппс = 20 833,34 – 17 928,04 = 2 905,30 руб.;

Пч = 2 905,30 ⋅ (1 – 20 / 100) = 2 324,24 руб.;

Урент = (2 905,30 / 17 928,04) ⋅ 100% = 16,2%.

В результате расчетов было определено, что цена без НДС составляет 20 833,34 руб., рентабельность – 16,2%, а чистая прибыль – 2 324,24 руб.

6.4 Выводы по разделу

В ходе тщательного анализа были определены ключевые параметры, влияющие на стоимость разработки веб-приложения для организации взаимоотношений с клиентами. Отпускная цена, которая отражает общую стоимость проекта, включая затраты на разработку, тестирование и последующую поддержку, была установлена на основе детального изучения всех факторов. Этот анализ позволил учесть все аспекты, начиная от первоначальных затрат на разработку и заканчивая долгосрочными расходами на обслуживание и развитие проекта.

В таблице 6.5 представлены результаты расчетов основных экономических показателей, включая стоимость разработки и ожидаемую прибыль. Эти данные дают четкое представление о финансовой целесообразности проекта, позволяя оценить, насколько быстро проект окупит свои начальные затраты и начнет приносить прибыль. Такой подход к анализу позволяет не только оценить текущую стоимость проекта, но и прогнозировать его будущую доходность, что является важным фактором при принятии решений о разработке новых продуктов или услуг. Данная таблица представлена в приложении Е.

Таблица 6.5 – Результаты расчетов основных экономических показателей разработки веб-приложения для организации взаимоотношений с клиентами

|  |  |
| --- | --- |
| Наименование показателя | Значение |
| Время разработки, часов | 623 |
| Количество разработчиков, чел. | 5 |
| Основная заработная плата, руб. | 6 816,64 |
| Дополнительная заработная плата, руб. | 1 022,49 |
| Отчисления в Фонд социальной защиты населения и БРУСП «Белгосстрах», руб. | 2 712,33 |
| Прочие прямые затраты, руб. | 1 704,16 |
| Накладные расходы, руб. | 3 067,49 |
| Себестоимость разработки программного средства, руб. | 15 323,11 |
| Расходы на сопровождение и адаптацию, руб. | 2 604,93 |
| Полная себестоимость, руб. | 17 928,04 |
| Цена реализации, сформированная на основе проведенного маркетингового анализа рынка, руб. | 25 000 |
| Прибыль от реализации, руб. | 2 905,30 |
| Рентабельность разработки, % | 16,2 |
| Чистая прибыль, руб. | 2 324,24 |

Учитывая среднюю стоимость разработки подобного веб-приложения в веб-студии, составляющую 30 000 рублей, цена в 25 000 рублей считается вполне приемлемой. Исходя из расчетов экономической эффективности, можно утверждать, что разработка веб-приложения для взаимодействия с клиентами оправдана и обеспечивает рентабельность в 16,2%. Таким образом, проект признается экономически обоснованным.

Стоимость данного приложения заметно ниже, чем у аналогичных продуктов на рынке, что позволяет существенно снизить бюджет, сохраняя при этом высокое качество и функциональность. В отличие от других решений, веб-приложение предлагает богатый набор возможностей и высокую производительность по более доступной цене. Экономия достигается за счет оптимизации процесса разработки и использования передовых технологий, что позволяет предложить клиентам высококачественный продукт по более низким ценам. Это дает клиентам конкурентное преимущество, выбирая более экономичное, но не менее эффективное решение.

Низкая стоимость создания программного продукта обусловлена выбором технологии Nest.js, которая обладает рядом преимуществ, включая простоту расширения функционала и модульность компонентов.

В рамках данного дипломного проекта в качестве основы разработки используется бесплатная технология, что привело к снижению общей себестоимости конечного продукта. Экономия, достигнутая благодаря использованию бесплатной технологии, позволяет оптимизировать бюджет проекта и сосредоточиться на других важных аспектах разработки.

# Заключение

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

Левша М.С.

Провер.

Гончар Е.А.

Н. контр.

Нистюк О.А.

Утв.

Смелов В.В.

Заключение

Лит.

Листов

1

74417043, 2024

В процессе выполнения дипломного проекта были выявлены и изучены основные аналоги с их достоинствами и недостатками, а также исследованы возможности рассматриваемых программных средств. На основе этого анализа было выполнено проектирование веб-приложения. На этапе проектирования была разработана диаграмма вариантов использования, спроектирована база данных приложения, выполнено проектирование архитектуры серверной части, составлена диаграмма развертывания приложения.

Веб-приложение реализовано на платформе Node.js, в качестве фреймворка использовался Nest.js. В качестве основного хранилища данных использовалась MongoDB. Для реализации аутентификации использовалась технология JWT. Для разработки клиентской части использовалась библиотека React совместно с Redux и ApolloClient. Разработанный программный продукт предоставляет API (GraphQL).

Разработано веб-приложение для организации взаимоотношений с клиентами предоставляющее следующий функционал: просмотр статусов задач на канбан-доске, создание и удаление задач, создание чата с другим пользователем, отправка сообщений с помощью чата, изменение статусов задач с помощью канбан-доски, создание, удаление, изменение, блокировка и разблокировка пользователей, уведомление по электронной почте об изменении доступа к приложению.

Для проверки правильности работы приложения были написаны unit и интеграционные тесты для серверной части веб-приложения, а также проведено ручное тестирование, показавшее, что приложение работает правильно и соблюдает все ранее установленные требования. При составлении руководства пользователя была подробно описана работа с каждой страницей веб-приложения.

Выполнено технико-экономическое обоснование проекта: определена полная себестоимость проекта, вычислена чистая прибыль от продажи веб-приложения, определена рентабельность приложения. На основании полученных данных можно сделать вывод, что разработанный проект является экономически выгодным.

Таким образом, была достигнута поставленная цель, а задачи дипломного проекта выполнены в полном объеме.

# Список используемых источников

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

Левша М.С.

Провер.

Гончар Е.А.

Н. контр.

Нистюк О.А.

Утв.

Смелов В.В.

Список используемых источников

Лит.

Листов

1

74417043, 2024

1. Веб-сервис Twenty [Электронный ресурс] / twenty.com. – Режим доступа: https://twenty.com/. – Дата доступа: 10.04.2024.
2. Система Bitrix24 [Электронный ресурс] / bitrix24.by. – Режим доступа: https://www.bitrix24.by/. – Дата доступа: 11.04.2024.
3. Система ClickUp [Электронный ресурс] / clickup.com. – Режим доступа: https://clickup.com/. – Дата доступа: 12.04.2024.
4. Google Patents [Электронный ресурс] / patents.google.com. – Режим доступа: https://patents.google.com/patent/US20230289339A1/en?oq=us20230289339/. – Дата доступа: 14.04.2024.
5. Visual Studio Code [Электронный ресурс] / code.visualstudio.com. – Режим доступа: https://code.visualstudio.com/. – Дата доступа: 15.04.2024.
6. MongoDB Compass [Электронный ресурс] / mongodb.com. – Режим доступа: https://www.mongodb.com/products/tools/compass/. – Дата доступа: 16.04.2024.
7. Postman [Электронный ресурс] / postman.com. – Режим доступа: https://www.postman.com/. – Дата доступа: 17.04.2024.
8. Fork [Электронный ресурс] / git-fork.com. – Режим доступа: https://git-fork.com/. – Дата доступа: 18.04.2024.
9. Node.js [Электронный ресурс] / nodejs.org. – Режим доступа: https://nodejs.org/. – Дата доступа: 20.04.2024.
10. Язык программирования TypeScript [Электронный ресурс] / typescriptlang.org. – Режим доступа: https://www.typescriptlang.org/. – Дата доступа: 22.04.2024.
11. Nest documentation [Электронный ресурс] / nestjs.com. – Режим доступа: https://nestjs.com. – Дата доступа: 23.04.2024.
12. База данных MongoDB [Электронный ресурс] / mongodb.com. – Режим доступа: https://www.mongodb.com/. – Дата доступа: 25.04.2024.
13. Библиотека Mongoose [Электронный ресурс] / mongoosejs.com. – Режим доступа: https://mongoosejs.com/. – Дата доступа: 27.04.2024.
14. Библиотека Passport.js [Электронный ресурс] / passportjs.org. – Режим доступа: https://www.passportjs.org/. – Дата доступа: 29.04.2024.
15. Бэнкс, А. React и Redux. Функциональная веб-разработка. / А. Бэнкс, Е. Порселло. – СПб: Питер, 2020. – 336с.
16. Фаулер, М. UML. Основы. Краткое руководство по стандартному языку объектного моделирования. / М. Фаулер. – 3-е изд. – СПб: Символ-Плюс, 2004. – 192с.
17. Каштелян, Т.В. Экономическое обоснование дипломных проектов / Каштелян Т.В. – Минск: УО «Белорусский государственный технологический университет», 2013. – 88 с.

# ПРИЛОЖЕНИЕ А. Логическая схема базы данных

# ПРИЛОЖЕНИЕ Б. Диаграмма развертывания

# ПРИЛОЖЕНИЕ В. Диаграмма вариантов использования

# ПРИЛОЖЕНИЕ Г. Блок-Схема

# ПРИЛОЖЕНИЕ Д. Скриншот работы приложения

# ПРИЛОЖЕНИЕ Е. Таблица экономических расчетов

# ПРИЛОЖЕНИЕ Ж. Реализация класса TaskService

@Injectable()

export class TaskService {

constructor(

private taskRepository: TaskRepository,

private userRepository: UserRepository

) {}

async getTasks(getTasksInput: GetTasksInput, clientId: string): Promise<GetTasksResponse> {

const regex = new RegExp(getTasksInput.filterString, 'i');

const filter = {

name: { $regex: regex },

};

const count: number = await this.taskRepository.count('name', filter);

const skip = Math.max(getTasksInput.skip, 0);

const limit = getTasksInput.limit <= 0 ? null : getTasksInput.limit;

const tasks = (await this.taskRepository.getMany(filter, null, skip, limit)).filter(

(item) =>

item.creator.\_id.toString() === clientId ||

item.developers[0].\_id.toString() === clientId ||

item.client.\_id.toString() === clientId

);

let newComments = [];

for (const item of tasks) {

for (const comment of item.comments) {

const owner = await this.userRepository.getOne({

\_id: convertStringToObjectId(comment.user.\_id),

});

newComments.push({

text: comment.text,

date: comment.date,

user: owner,

});

}

item.comments = newComments;

newComments = [];

}

return {

tasks,

count,

};

}

async createTask(createTaskInput: CreateTaskInput, clientId: string): Promise<Task> {

const taskFields = {

...createTaskInput,

creator: convertStringToObjectId(clientId),

comments: [],

};

try {

const task = await this.taskRepository.create(taskFields);

return task;

} catch (e) {

throw new BadRequestException('Ошибка создания задачи');

}

}

async createComment(createCommentInput: CreateCommentInput, clientId: string): Promise<Task> {

const { taskId, ...commentInfo } = createCommentInput;

const commentFields = {

...commentInfo,

user: convertStringToObjectId(clientId),

};

try {

await this.taskRepository.update(taskId, {

$push: { comments: commentFields },

});

const task = await this.taskRepository.getOne({ \_id: convertStringToObjectId(taskId) });

const newComments = [];

for (const item of task.comments) {

const owner = await this.userRepository.getOne({

\_id: convertStringToObjectId(item.user.\_id),

});

newComments.push({

text: item.text,

date: item.date,

user: owner,

});

}

task.comments = newComments;

return task;

} catch (e) {

throw new BadRequestException('Ошибка создания комментария');

}

}

async getTask(taskId: string): Promise<Task> {

const task = await this.taskRepository.getOne({ \_id: convertStringToObjectId(taskId) });

const newComments = [];

for (const item of task.comments) {

const owner = await this.userRepository.getOne({

\_id: convertStringToObjectId(item.user.\_id),

});

newComments.push({

text: item.text,

date: item.date,

user: owner,

});

}

task.comments = newComments;

return task;

}

async deleteTask(taskId: string): Promise<Task> {

const task = await this.taskRepository.delete(taskId);

return task;

}

}

# ПРИЛОЖЕНИЕ И. Реализация класса MongoGenericRepository

export class MongoGenericRepository<T> {

private \_repository: Model<T>;

protected \_populatedFields: string[];

constructor(repository: Model<T>, populatedFields?: string[]) {

this.\_repository = repository;

this.\_populatedFields = populatedFields;

}

updateMany(update: any, filter?: any, options?: any): any {

return this.\_repository.updateMany(filter, update, options);

}

updateOne(update: any, filter?: any, options?: any): any {

return this.\_repository.updateOne(filter, update, options);

}

async create(item: any): Promise<T> {

return this.\_repository.create(item);

}

async createMany(item: any[], options?: any): Promise<any> {

return this.\_repository.create(item, options);

}

async update(id: string, item: any, session?: any): Promise<T> {

const options = !session ? { new: true } : { session, new: true };

return this.\_repository.findByIdAndUpdate(id, item, options).exec();

}

async getMany(filter: any, sort?: any, skip?: any, limit?: any, session?: any): Promise<T[]> {

return this.\_repository

.find(filter, null, { session: session })

.skip(skip)

.limit(limit)

.sort(sort)

.populate(this.\_populatedFields)

.exec();

}

async getOne(filter: any, sort?: any, session?: any): Promise<any> {

return this.\_repository

.findOne(filter, null, { session: session })

.sort(sort)

.populate(this.\_populatedFields)

.exec();

}

async delete(id: string, options?: any): Promise<any> {

return this.\_repository.findByIdAndDelete(id, options);

}

deleteMany(filter: any, options?: any): any {

return this.\_repository.deleteMany(filter, options);

}

async bulkWrite(bulkWriteOperations: any, options?: any): Promise<any> {

return this.\_repository.bulkWrite(bulkWriteOperations, options);

}

async count(field: string, filter: any = {}): Promise<number> {

const countObject = await this.\_repository.aggregate([

{ $match: filter },

{ $count: field },

]);

const count: number = countObject.length > 0 ? countObject[0][field] : 0;

return count;

}

}

# ПРИЛОЖЕНИЕ К. Реализация компонента TasksView

const TasksView = () => {

const navigate = useNavigate();

const user = useSelector((state) => state.app.user);

const [tasks, setTasks] = useState([]);

const [show, setShow] = useState(false);

const [developers, setDevelopers] = useState([]);

const [clients, setClients] = useState([]);

const [name, setName] = useState('');

const [description, setDescription] = useState('');

const [price, setPrice] = useState(0);

const [developer, setDeveloper] = useState('');

const [client, setClient] = useState('');

const [date, setDate] = useState({});

const [addedTask, setAddedTask] = useState({});

const [deletedTask, setDeletedTask] = useState(null);

const handleNameChange = (event) => setName(event.target.value);

const handleDescriptionChange = (event) => setDescription(event.target.value);

const handlePriceChange = (event) => setPrice(event.target.value);

const handleDeveloperChange = (event) => setDeveloper(event.target.value);

const handleClientChange = (event) => setClient(event.target.value);

const handleDateChange = (event) => setDate(event);

const [getTasks] = useLazyQuery(taskQueries.GET\_TASKS, {

onCompleted: (data) => {

setTasks(data.getTasks.tasks);

},

fetchPolicy: 'network-only',

});

useEffect(() => {

getTasks({

variables: {

getTasksInput: { filterString: '', skip: 0, limit: -1 },

},

fetchPolicy: 'network-only',

});

}, [addedTask, deletedTask]);

const [getDevelopers] = useLazyQuery(userQueries.GET\_USERS, {

onCompleted: (data) => {

setDevelopers(data.getUsers.users);

},

fetchPolicy: 'network-only',

});

const [getClients] = useLazyQuery(userQueries.GET\_USERS, {

onCompleted: (data) => {

setClients(data.getUsers.users);

},

fetchPolicy: 'network-only',

});

const handleClose = () => setShow(false);

const handleShow = () => {

getDevelopers({

variables: {

getUsersInput: { filterString: 'DEVELOPER', skip: 0, limit: -1 },

},

});

getClients({

variables: {

getUsersInput: { filterString: 'CLIENT', skip: 0, limit: -1 },

},

});

setShow(true);

};

const [addTask] = useMutation(taskMutations.CREATE\_TASK, {

onCompleted: (data) => {

setAddedTask(data);

setShow(false);

},

});

const handleAddTask = () => {

addTask({

variables: {

createTaskInput: {

name: name,

description: description,

price: +price,

developers: [developer],

client: client,

startDate: `${date.from.year}-${date.from.month}-${date.from.day}`,

endDate: `${date.to.year}-${date.to.month}-${date.to.day}`,

status: 'NEW',

},

},

});

};

const [deleteTask] = useMutation(taskMutations.DELETE\_TASK, {

onCompleted: (data) => {

setDeletedTask(data);

},

});

const handleDeleteTask = (id) => {

deleteTask({

variables: {

taskId: id,

},

});

};

return (

<>

<Table striped bordered hover>

<thead>

<tr>

<th>#</th>

<th>Task name</th>

<th>Creator</th>

<th>Client</th>

<th>Developer</th>

<th>Status</th>

{user.role === 'MANAGER' && <th>Delete</th>}

</tr>

</thead>

<tbody>

{tasks.map((item, index) => (

<tr key={index}>

<td onClick={() => navigate('/task', { state: { item } })}>

{++index}

</td>

<td onClick={() => navigate('/task', { state: { item } })}>

{item.name}

</td>

<td onClick={() => navigate('/task', { state: { item } })}>

{item.creator.nickname}

</td>

<td onClick={() => navigate('/task', { state: { item } })}>

{item.client.nickname}

</td>

<td onClick={() => navigate('/task', { state: { item } })}>

{item.developers[0].nickname}

</td>

<td onClick={() => navigate('/task', { state: { item } })}>

{item.status}

</td>

{user.role === 'MANAGER' && (

<td>

<Button

variant="danger"

onClick={() => handleDeleteTask(item.\_id)}

>

Delete

</Button>

</td>

)}

</tr>

))}

<tr>

<td colSpan={7}>

<Button

variant="primary"

onClick={() => {

handleShow();

}}

>

Add task

</Button>

</td>

</tr>

</tbody>

</Table>

<Modal show={show} onHide={handleClose} centered>

<Modal.Header closeButton>

<Modal.Title>Modal window</Modal.Title>

</Modal.Header>

<Modal.Body>

<Form>

<Form.Group className="mb-3" controlId="exampleForm.ControlInput1">

<Form.Label>Name</Form.Label>

<Form.Control

type="text"

placeholder="Enter name"

autoFocus

onChange={handleNameChange}

/>

</Form.Group>

<Form.Group className="mb-3" controlId="exampleForm.ControlInput2">

<Form.Label>Description</Form.Label>

<Form.Control

as="textarea"

rows={3}

placeholder="Enter description"

onChange={handleDescriptionChange}

/>

</Form.Group>

<Form.Group className="mb-3" controlId="exampleForm.ControlInput3">

<Form.Label>Price</Form.Label>

<Form.Control

type="number"

placeholder="Enter price"

onChange={handlePriceChange}

/>

</Form.Group>

<Form.Group className="mb-3" controlId="exampleForm.ControlInput4">

<Form.Label>Developer</Form.Label>

<Form.Select

aria-label="Default select example"

onChange={handleDeveloperChange}

>

<option>Select developer</option>

{developers.map((item, index) => (

<option value={item.\_id} key={index}>

{item.nickname}

</option>

))}

</Form.Select>

</Form.Group>

<Form.Group className="mb-3" controlId="exampleForm.ControlInput5">

<Form.Label>Client</Form.Label>

<Form.Select

aria-label="Default select example"

onChange={handleClientChange}

>

<option>Select client</option>

{clients.map((item, index) => (

<option value={item.\_id} key={index}>

{item.nickname}

</option>

))}

</Form.Select>

</Form.Group>

<Form.Group className="mb-3" controlId="exampleForm.ControlInput6">

<Form.Label>Period</Form.Label>

<DtPicker type={'range'} onChange={handleDateChange} />

</Form.Group>

</Form>

</Modal.Body>

<Modal.Footer>

<Button variant="secondary" onClick={handleClose}>

Cancel

</Button>

<Button variant="primary" onClick={handleAddTask}>

Save

</Button>

</Modal.Footer>

</Modal>

</>

);

};

export default TasksView;

# ПРИЛОЖЕНИЕ Л. Реализация компонента GraphQLProvider

const GraphQlProvider = ({ children }) => {

const token = useSelector((state) => state.app.accessToken);

const dispatch = useDispatch();

const uploadLink = createHttpLink({

uri: HTTP\_API\_URL,

});

const wsLink = new GraphQLWsLink(

createClient({

url: WS\_API\_URL,

connectionParams: {

authToken: `Bearer ${token}`,

},

})

);

const authLink = setContext((\_, { headers }) => {

return {

headers: {

...headers,

Authorization: `Bearer ${token}`,

},

};

});

const httpLinkWithAuth = authLink.concat(uploadLink);

const splitLink = split(

({ query }) => {

const definition = getMainDefinition(query);

return definition.kind === 'OperationDefinition' && definition.operation === 'subscription';

},

wsLink,

httpLinkWithAuth

);

const errorLink = onError((errorResponse) => {

console.log(errorResponse);

const error = errorHandler(errorResponse);

dispatch(setError(error));

});

const client = new ApolloClient({

link: errorLink.concat(splitLink),

cache: new InMemoryCache({

addTypename: false,

}),

headers: {

authorization: `Bearer ${token}`,

},

defaultOptions: {

query: {

errorPolicy: 'all',

},

mutate: {

errorPolicy: 'all',

},

},

});

return <ApolloProvider client={client}>{children}</ApolloProvider>;

};

export default GraphQlProvider;