Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение
высшего образования «Уральский федеральный университет
имени первого Президента России Б.Н. Ельцина»

Институт радиоэлектроники и информационных технологий - РТФ
Центр ускоренного обучения

УДК 004.4

Допустить к защите

Директор ЦУО ИРИТ-РТФ

_____ Обухов О.В.

«___»_____2024 г.

РАЗРАБОТКА АВТОМАТИЗИРОВАННОЙ СИСТЕМЫ ПОСТАНОВКИ ЗАДАЧ ДЛЯ ПЕРСОНАЛА ЧАСТНОЙ КОМПАНИИ

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
Пояснительная записка
09.04.03 58.29.21 009 ПЗ

Руководитель доц., к.ф.-м.н., Dr.rer.nat	О.Н. Александрова
Нормоконтролёр ст. пр.	О.В. Новосёлова
Студент гр. РИЗМ-320901к	В.А. Подчезерцев

Екатеринбург 2024

РЕФЕРАТ

Пояснит. зап. 159 стр., 64 рис., 11 таб., 36 источн., 1 прил.

СИСТЕМА, ЗАДАЧА, АРХИТЕКТУРА, ИНТЕРФЕЙС, ФРЕЙМВОРК, БАЗА ДАННЫХ, СЕРВЕР, СОТРУДНИК, ЧАСТНАЯ КОМПАНИЯ, БИЗНЕС.

Объектом исследования является автоматизированная система постановки задач для частной финансовой компании. Цель работы – разработка веб-приложения для эффективного управления задачами, контроля их выполнения и взаимодействия сотрудников.

В ходе работы была охарактеризована предметная область, выявлены проблемы управления задачами, сформулированы требования к системе и проведен анализ существующих решений. На основе анализа определены ключевые функции и выбрана архитектура системы. Проведен выбор технологий: Java и Spring Boot для серверной части, Angular Material для клиентской. Спроектированы структура базы данных, интерфейс пользователя, диаграммы вариантов использования, последовательности и блоков, а также обратным проектированием выполнена диаграмма классов, описывающая взаимодействия в серверной части.

На завершающем этапе работы была реализована система, включающая модули регистрации и авторизации пользователей, постановки, мониторинга и поиска задач, а также форма взаимодействия пользователей через комментарии к задачам. Для обеспечения безопасности был внедрен механизм валидации токенов на основе JSON Web Token (JWT).

В результате создано веб-приложение с монолитной архитектурой, базой данных развёрнутой СУБД MySQL 8.0 и интуитивно понятным интерфейсом. Система протестирована и готова к внедрению.

В перспективе планируется сбор обратной связи от сотрудников, дальнейшая оптимизация системы и добавление новых функций, направленных на улучшение производительности и удобства работы.

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ 7
ВВЕДЕНИЕ 8
1 Постановка задачи 9
1.1 Результаты обследования предметной области 9
1.1.1 Область деятельности исследуемого объекта 9
1.1.2 Модель взаимодействия исследуемого объекта с субъектами внешнего мира 10
1.1.3 Основные недостатки существующей технологии, которую использует исследуемый объект 13
1.1.4 Обзор аналогов, решающих задачи исследуемого объекта 14
1.1.5 Обоснование разработки системы для исследуемого объекта 17
1.2 Определение представителей заинтересованных сторон 17
1.3 Сбор требований 18
1.3.1 Планирование 18
1.3.2 Подготовка к встрече 19
1.3.3 Подготовка к интервью 19
1.3.4 Интервью 20
1.3.5 Опросы 22
1.3.6 Наблюдение за работой сотрудником компании, использующим Битрикс24. 25
1.3.7 Анализ документов 27
1.4 Требования к структуре системы в целом 29
1.4.1 Перечень подсистем, их назначение и основные характеристики 29
1.4.2 Требования к способам и средствам обеспечения информационного взаимодействия компонентов системы 29
1.4.3 Требования к характеристике взаимосвязей создаваемой системы со смежными системами 29
1.4.4 Требования к режимам функционирования системы 30
1.4.5 Перспективы развития, модернизации системы 30
1.5 Требования к функциям, выполняемым системой 30
1.5.1 Функции, подлежащие автоматизации 30
1.5.2 Временной регламент реализации функций 31
1.5.3 Требования одновременности выполнения группы функций 31
1.5.4 Критерии отказов 31
1.6 Требования к видам обеспечения 31
1.6.1 Требования к математическому обеспечению 31

1.6.2 Требования к информационному обеспечению 32
1.6.3 Требования к лингвистическому обеспечению 32
1.6.4 Требования к программному обеспечению (ПО) 32
1.6.5 Требования к техническому обеспечению 33
1.6.6 Требования к метрологическому обеспечению 33
1.6.7 Требования к методическому обеспечению 33
1.7 Общие технические требования к системе 33
1.7.1 Требования к численности и квалификации персонала и пользователей системы 33
1.7.2 Требования к надежности 33
1.7.3 Требования по безопасности 33
1.7.4 Требования к эргономике и технической эстетике 34
1.7.5 Требования к эксплуатации, техническому обслуживанию, ремонту и 37
хранению компонентов системы 37
1.7.6 Требования к защите информации от несанкционированного доступа 37
1.7.7 Требования по сохранности информации при авариях 37
1.7.8 Требования к патентной чистоте и патентоспособности 38
1.7.9 Дополнительные требования 38
2 Анализ поставленной задачи 39
2.1 Анализ требований к системе 39
2.1.1 Анализ требований к структуре в целом 39
2.1.2 Анализ требований к функциям, выполняемым системой 39
2.1.3 Анализ требований к видам обеспечения 40
2.1.4 Анализ общих технических требований к системе 40
2.2 Обоснование архитектуры системы 41
2.2.1 Сравнение монолитной и распределенной архитектур 41
2.2.3 Причины выбора монолитной многоуровневой архитектуры 43
2.2.2 Уровни монолитной многоуровневой архитектуры 44
2.3 Выбор инструментальных средств 44
2.4 Анализ требований к интерфейсу пользователя 45
2.4.1 Базовые бизнес-требования 45
2.4.2 Ольга (ключевой персонаж) 45
2.4.3 Бизнес-цели 46
2.4.4 Персональные цели 46
2.4.5 Сценарии работы 46
2.4.6 Типичный день 47

2.4.7 Состав и внутренняя структура ключевых задач персонажа 48
2.4.8 Требования персонажа к программе 48
2.4.9 Важность пользовательских требований 49
2.4.10 Информационная структура приложения, общая концепция 50
2.4.11 Реализация требований пользователя 51
3 Описание системы 54
3.1 Архитектура разрабатываемой системы 54
3.1.1 Аннотация Business Layer 55
3.1.2 Аннотация Application Layer 56
3.1.3 Аннотация Technology Layer 56
3.1.4 Построение диаграммы блоков 57
3.2 Структура программного обеспечения 59
3.2.1 Диаграмма вариантов использования 59
3.2.2 Диаграммы последовательности 60
3.2.3 Диаграммы классов 62
3.3 Описание базы данных 82
3.3.1 Таблицы, используемые в базе данных 83
3.3.2 Связи между таблицами в базе данных 85
3.3.3 Примеры SQL-запросов 85
3.3.4 Оптимизация работы базы данных 86
3.4. Обзор результатов тестирования 87
3.4.1 Автоматическое тестирование 87
3.4.2 Нагрузочное тестирование 87
4 Руководство администратора и пользователя 89
4.1 Требуемые ресурсы 89
4.2 Руководство пользователя статуса admin 89
4.3 Руководство пользователя статуса employee 93
4.4 Сообщения об ошибках системы 96
ЗАКЛЮЧЕНИЕ 101
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 102
ПРИЛОЖЕНИЕ А (обязательное) Исходный код программы 105
А.1 Frontend-часть 105
А.2 Backend-часть 145

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

Битриск24 – облачная CRM-система и сервис управления бизнесом

Таск-менеджер – автоматизированная система для управления задачами/процессами

ФРИСБИ – компания, занимающаяся финансовой логистикой

Admin – статус пользователя со расширенными функциями системы

Angular – фреймворк для разработки SPA-приложений на JavaScript/TypeScript

Angular Material – библиотека пользовательских интерфейсов для Angular

ArchiMate – язык моделирования архитектуры предприятия, который поддерживает описание, анализ и визуализацию архитектуры внутри и между бизнес-доменами

Backend – часть веб-сайта или программного обеспечения, которая не видна пользователям, включает в себя серверные процессы, такие как обработка запросов, доступ к базам данных и выполнение логики приложения.

Check Boxes – элементы интерфейса для выбора нескольких опций

Edit Field – поле ввода текста

Employee – статус пользователя со стандартным набором функций системы

E-mail – система обмена сообщениями через Интернет между пользователями

Frontend – часть веб-приложения или программы, с которой взаимодействуют пользователи, включает в себя все элементы интерфейса, такие как кнопки, формы и графические компоненты

IntelliJ IDEA – интегрированная среда разработки (IDE), созданная компанией JetBrains для разработки на языках программирования Java и других языка

Java – объектно-ориентированный язык программирования

MySQL – реляционная система управления базами данных, созданная компанией MySQL AB

Option Buttons – радиокнопки для выбора одного из нескольких вариантов

Radio Buttons – элементы интерфейса для выбора одного варианта

RESTful API – архитектурный стиль API, который используется для регламентирования взаимодействия между клиентской и серверной частью программного обеспечения.

Spring Boot – фреймворк для разработки приложений на Java

Spring Data JPA – модуль Spring для работы с базами данных

Spring Security – модуль Spring для реализации безопасности

Unit test –процесс позволяющий проверить на корректность отдельные модули исходного кода программы

Visual Studio Code – бесплатный редактор кода с открытым исходным кодом, разработанный Microsoft

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

БД – База данных

ГОСТ Р – национальный стандарт Российской Федерации

ГБ – Гигабайт, единица измерения объема данных

ГГц – Гигагерц, единица частоты

ИС – Информационная система

ПО – Программное обеспечение

СУБД – Система управления базами данных

API – Application Programming Interface программный интерфейс, с помощью которого приложения, веб-сервисы и программы обмениваются информацией.

CRUD – (Create, Read, Update, Delete) операции для работы с данными

CRM – (Customer Relationship Management) система управления взаимоотношениями с клиентами

DAL – (Data Access Layer) слой компьютерной программы, который предоставляет упрощенный доступ к данным, хранимым в постоянном хранилище какого-либо типа, таком как реляционная база данных

DTO – (Data Transfer Object) паттерн проектирования, используемый для переноса данных между слоями приложения, часто между клиентом и сервером или между разными слоями в бэкенде

ID – (Identifier) уникальный идентификатор

IP-ATC – (Internet Protocol – Automatic Telephone Station) IP-телефония

JWT – (JSON Web Token) формат токенов для аутентификации

MHz – мегагерц, единица измерения частоты

REST – (Representational State Transfe) передача состояния представления», «передача репрезентативного состояния» или «передача „самоописываемого “состояния»

SPA – (Single Page Application) одностраничное приложение

SQL – (Structured Query Language) язык структурированных запросов

UX – (User Experience) опыт пользователя, который он получает во время взаимодействия с разрабатываемым продуктом

UI – (User Interface) интерфейс пользователя

x86/x64 – основные архитектуры процессоров, используемых в современных компьютерах, которые определяют, как процессоры обрабатывают данные и взаимодействуют с памятью

ВВЕДЕНИЕ

Развитие цифровых технологий оказывает значительное влияние на экономику, позволяя компаниям внедрять современные инструменты для оптимизации бизнес-процессов. Одним из таких решений являются автоматизированные системы постановки задач, которые помогают эффективно управлять проектами и внутренними операциями, повышая продуктивность и упрощая координацию сотрудников.

Создание автоматизированной системы постановки задач для частной компании особенно актуально в условиях высоких требований к скорости и точности управления процессами. Частные компании нуждаются в строгом контроле задач, поддержании безопасности данных и автоматизации распределения ресурсов, что требует современных решений для повышения эффективности и конкурентоспособности.

Целью данной работы является проектирование и разработка веб-приложения, обеспечивающего постановку, мониторинг и выполнение задач с учетом особенностей бизнес-процессов компании. В рамках исследования были изучены требования к системе, проведен анализ предметной области и существующих решений, а также разработаны архитектура, структуры данных и интерфейсы системы.

Результаты работы включают проектирование и создание системы с ключевыми модулями: регистрация и авторизация пользователей, управление задачами, поиск задач, мониторинг их выполнения и взаимодействие сотрудников через комментарии. Система протестирована, ее функциональность подтверждена, и она готова к внедрению в рабочие процессы компании.

Работа демонстрирует решение актуальной задачи цифровизации управления задачами в частной компании, что способствует повышению ее продуктивности и конкурентоспособности.

1 Постановка задачи

Анализ существующих корпоративных систем [1] стал важным этапом научно-исследовательской работы, проведенной в рамках разработки автоматизированной системы постановки задач. На завершающем этапе проекта этот анализ позволил окончательно уточнить сильные и слабые стороны исследуемых решений, что стало основой для проверки соответствия разработанной системы поставленным целям и требованиям. Выявленные преимущества и недостатки существующих систем подтвердили обоснованность выбранного подхода и предоставили ценную информацию для будущих улучшений.

1.1 Результаты обследования предметной области

1.1.1 Область деятельности исследуемого объекта

Областью деятельности объекта являются процессы постановки задач.

В частной компании «ФРИСБИ» процессы постановки задач организованы с помощью информационно-программного обеспечения Битриск24.

Битрикс24 это облачная платформа для совместной работы и общения, которая предлагает ряд бизнес-инструментов, включая CRM, управление проектами, общение в команде и многое другое. Она предназначена для малого и среднего бизнеса и предлагает бесплатный план с базовыми функциями и платные планы с более продвинутыми возможностями.

Основные преимущества использования Битрикс24[2]:

– широкий спектр услуг (предоставляет полный набор бизнес-инструментов, которые позволяют командам обмениваться информацией и сотрудничать более эффективно);

– легкая настраиваемость (настраиваемые конкретные потребности бизнеса, благодаря возможности добавления пользовательских полей, рабочих процессов и интеграций);

– доступность (бесплатный тарифный план, а также доступные платные тарифные планы с гибкой ценовой политикой для любого бюджета);

– безопасность (использование передовых мер безопасности для защиты данных пользователей и обеспечения конфиденциальности всей информации).

Использование системы Битрикс24 в финансовой системе «ФРИСБИ предусмотрено посредством специальных программных модулей, которые позволяют осуществлять такие функции как:

– общения сотрудников (групповой част и видеозвонки);

– постановка задач (назначение, распределение);

– ведение статистики (ведение статистики за определенный период времени);
– CRM (Система управления взаимоотношениями с клиентами);
– почтовый клиент (корпоративная почта для сотрудников);
– база знаний (накопленные данные за определенный период времени).

1.1.2 Модель взаимодействия исследуемого объекта с субъектами внешнего мира

Общая схема внутренних связей частной компании представлена на рисунке 1.

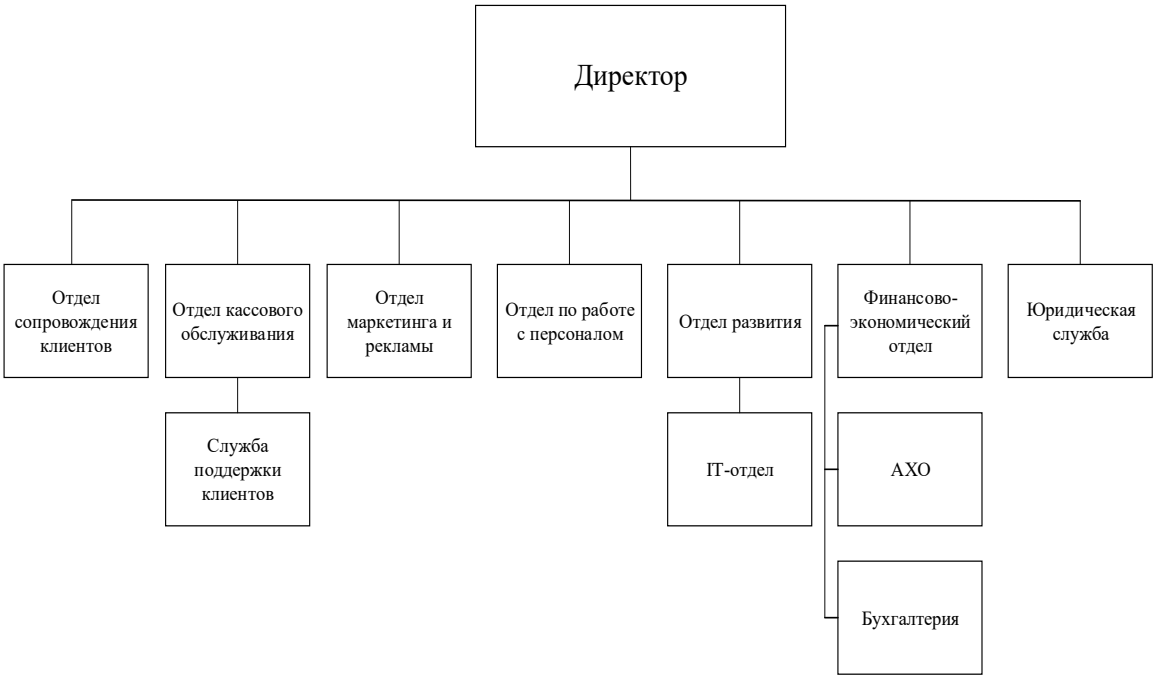
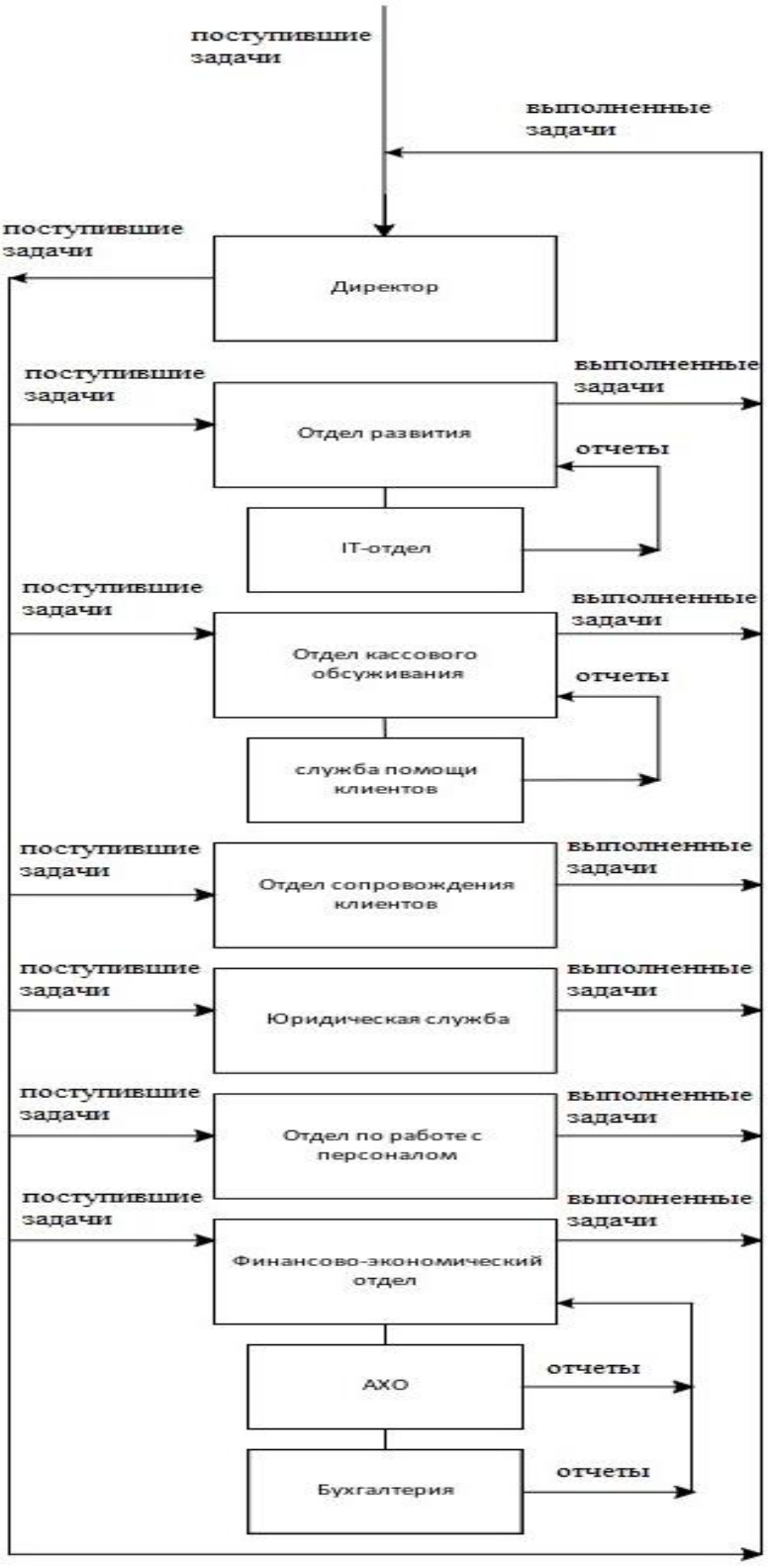


Рисунок 1 – Структура организации частной компании

Директор компании взаимодействует с различными субъектами, которые, с одной стороны, влияют на появления требований к задачам, а с другой – эти требования воплощают в реальность в процессе реализации задач. В данной структуре используется водопадная модель.

Каждый сотрудник компании является связующим звеном между постановкой и выполнением задач (Рис.2), что подчеркивает важность отслеживания хода реализации постановки задач для частной компании.



Общая схема распределения задач в частной компании представлена на рисунке 2.

Рисунок 2 – Структура распределения задач частной компании

Любая поступившая задача из вне сначала оценивается директором компании, а в дальнейшем поступает в программу распределения задач внутри компании, по специальному идентификатору задач.

1.1.3 Основные недостатки существующей технологии, которую использует исследуемый объект

Как уже было сказано ранее, на исследуемом объекте используется информационно-программное обеспечение Битрикс24. Данное ПО имеет ряд недостатков, которые затрудняют его использование:

– Ограниченная настройка на бесплатном плане, бесплатный тарифный план является отправной точкой, но имеет ограничения. Согласно опросу пользователей (основанному на данных интернет-форумов и обзоров), 45% респондентов отметили, что отсутствие возможности настройки критически сказывается на эффективности работы в бесплатной версии.

– Ограниченные интеграции в бесплатной версии интеграции ограничены, и только около 20% популярных приложений (например, внешние CRM, платежные системы) поддерживаются. В результате, по статистике, 38% малых предприятий вынуждены тратить дополнительные средства на доработку или искать обходные решения.

– Зависимость от подключения к Интернету, как облачная платформа, Битрикс24 зависит от стабильного интернет-соединения. По отзывам пользователей, средняя задержка при работе с задачами через нестабильное подключение составляет 2-5 секунд, что особенно ощутимо для команд, работающих в удаленных регионах. Около 27% пользователей указывают на регулярные проблемы с доступностью сервиса.

– Ограниченное хранилище в бесплатной версии предоставляется лишь 5 ГБ пространства, чего недостаточно для команд, активно работающих с документами и файлами. Примерно 32% пользователей сталкиваются с необходимостью перехода на платный план уже в первые 3-4 месяца использования.

– Ограниченные возможности электронной коммерции, если бизнес ориентирован на электронную коммерцию, инструменты Битрикс24 уступают другим платформам. Согласно исследованию пользовательских отзывов, 41% предпринимателей отметили, что инструменты аналитики и настройки онлайн-продаж недостаточны.

– Дополнительное обучение, из-за большого функционала платформы новым пользователям требуется значительное время для освоения. Среднее время обучения сотрудников на базовые операции составляет от 3 до 7 часов, что превышает аналогичные показатели для более простых платформ (например, Trello или Asana). По опросам, 35% пользователей сталкивались с необходимостью проведения дополнительных тренингов.

1.1.4 Обзор аналогов, решающих задачи исследуемого объекта

В настоящее время существует достаточное количество различных платформ для реализации рассматриваемой задачи. Рассмотрим преимущества и недостатки наиболее популярных из них:

1.1.4.1 Таск менеджер ASANA.

Asana [3] это веб-программа для управления задачами и совместной работы, которая позволяет командам планировать, отслеживать и управлять своей работой.

Преимущества Asana:

– удобный интерфейс;
– настраиваемые параметры проекта;
– возможность создавать и назначать задачи;
– отслеживать ход выполнения проекта;
– обновления в режиме реального времени и возможность интеграции с другими приложениями.
– Недостатки Asana:
– стоимость программы;
– отсутствие функции учета рабочего времени;
– ограниченные возможности настройки.

В целом, Asana –это мощный инструмент управления проектами, который может помочь командам оставаться организованными и эффективно сотрудничать, но вы должны оценить свои конкретные потребности и рабочие процессы, чтобы определить, подходит ли он для вашей команды.

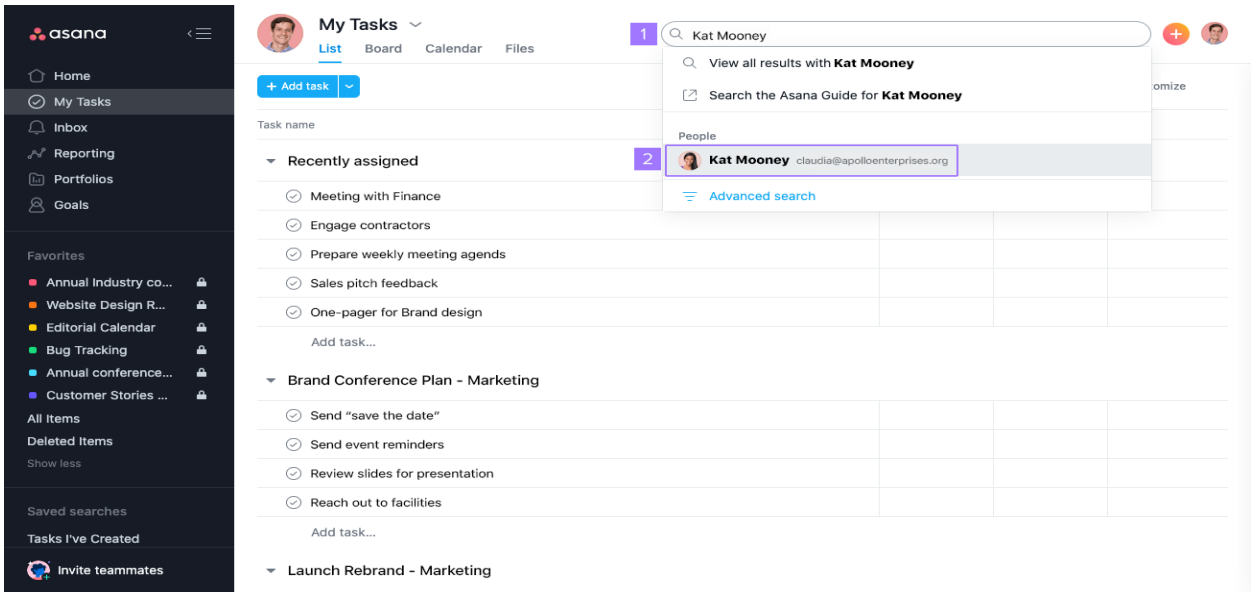


Рисунок 3 – Главная страница такс-менеджера Asana

1.1.4.2 Таск-менеджер TRELLO.

Trello [3] – это веб-инструмент, который помогает командам управлять проектами и задачами с помощью гибкого визуального подхода. Его главной особенностью является использование досок Канбан, где задачи представлены карточками, которые можно перемещать между различными этапами рабочего процесса.

Преимущества Trello:

– простой и удобный интерфейс;

– высокая гибкость (используется для широкого спектра задач и проектов;

– настраивается в соответствии с рабочими процессами команды;

– легко сотрудничать с членами команды благодаря обновлениям и уведомлениям в режиме реального времени;

– бесплатная версия с базовыми функциями.

– Недостатки Trello:

– ограниченная функциональность по сравнению с более продвинутыми инструментами управления проектами;

– может быть загромождён излишними карточками и списками;

– ограниченная отчетность и аналитика для отслеживания прогресса;

– нет встроенных функций отслеживания времени или распределения ресурсов;

– совместная работа может быть затруднена для больших команд или более сложных проектов.

В целом, Trello – это отличный инструмент для малых и средних команд, которые ищут простой и наглядный способ управления своими проектами и задачами.

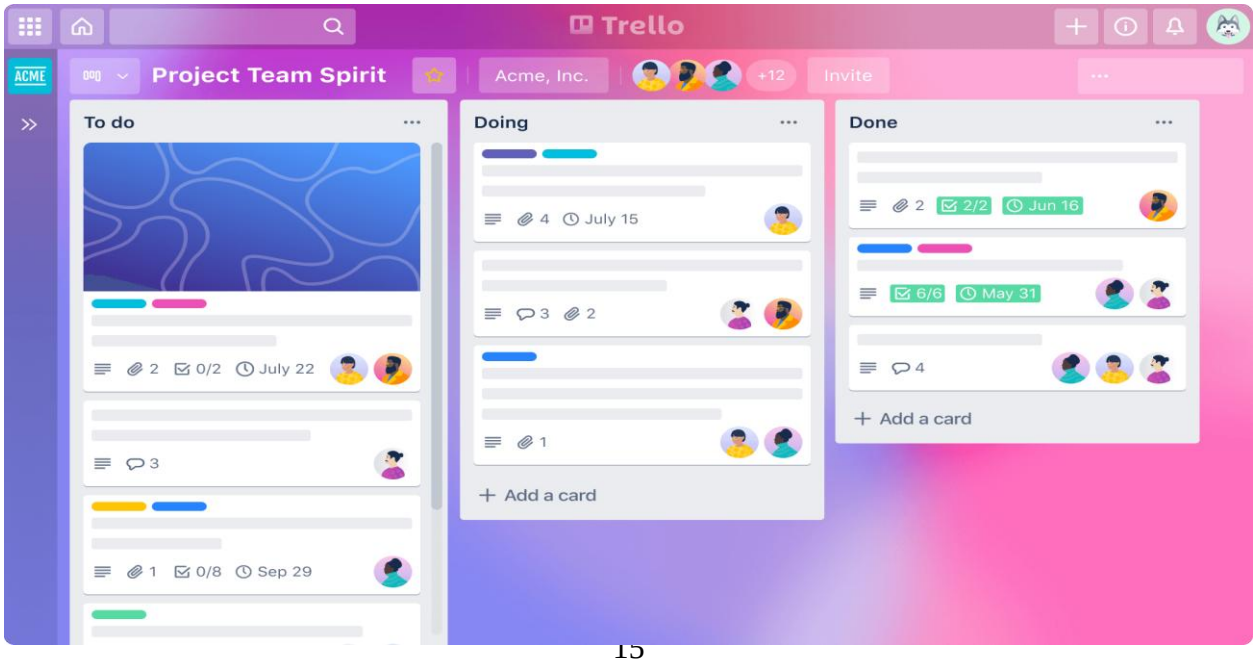


Рисунок 4 – Главная страница таск-менеджера Trello

1.1.4.3 Таск-менджер ClickUP.

ClickUp [4] – это инструмент управления проектами, который предлагает расширенные возможности управления задачами и множество функций, помогающих командам оставаться организованными и продуктивными.

Преимущества ClickUp:

– настраиваемость (высокая степень настройки);

– совместная работа (чат в режиме реального времени, комментарии и упоминания.)

– множество видов представления задач (представление списка, представление доски и представление календаря)

– интеграции (Google Drive, Trello, Slack и др)

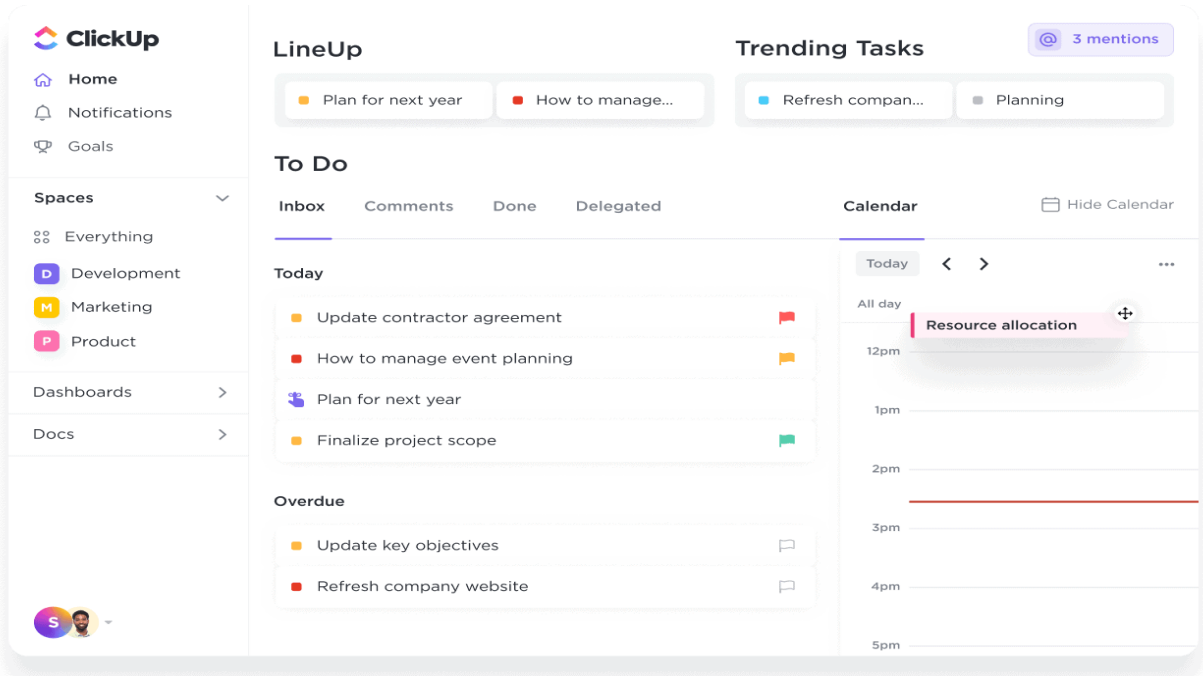
Недостаткам ClickUp

сложность (мощный набор функций может потребовать некоторого времени для освоения);

ценообразование (дорогие тарифные планы)

ограниченная отчетность (недостаточная аналитика);

В целом, ClickUp –это мощный инструмент управления проектами с широким набором функций и возможностей настройки. Хотя он может не подойти для каждой команды, он может быть полезным инструментом для команд со сложными потребностями в управлении



проектами.

Рисунок 5 – Главная страница такс-менеджера ClickUP

1.1.5 Обоснование разработки системы для исследуемого объекта

Рассмотрение существующих систем, а также системы Битриск24 позволяет сделать следующие выводы:

– существующие решения не позволяют в полной мере отразить то, что необходимо для постановки задач для частной финансовой компании «ФРИСБИ»;

– наличие излишней функциональности в существующих таск-менеджерах усложняет процесс постановки задачи;

– высокая стоимость премиум-подписки;

– отсутствие интуитивно-понятного интерфейса.

Учитывая данные выводы по сравнению наиболее популярных систем, а также системы Битриск24, можно сделать заключение о необходимости создания инструмента, отвечающего специфическим требованиям клиента, а именно частной компании «ФРИСБИ»

1.2 Определение представителей заинтересованных сторон

Определим, кто может быть заинтересован в использовании системы постановки задач для частной компании занимающейся финансовой логистикой, для этого составим таблицу 1.

Таблица 1 – Реестр представителей заинтересованных сторон

Роль	ФИО	Отдел	Должность	Цель/Влияние
Инициатор, разработчик приложения	Подчезерцев В.А.	-	-	Помощь в разработке инструмента для постановки задач
Пользователи		Сопровождения клиентов	Менеджеры по работе с клиентами	Получение нового удобного инструмента для постановки задач
Пользователи		Кассового обслуживания	Менеджеры по работе с кассирами	Получение нового удобного инструмента для постановки задач

Роль	ФИО	Отдел	Должность	Цель/Влияние
Пользователи		Маркетинга и рекламы	Дизайнеры и СММ менеджеры	Получение нового удобного инструмента для постановки задач
Пользователи		Отдел по работе с персоналом	Менеджеры по подбору персонала, специалисты по охране труда	Получение нового удобного инструмента для постановки задач
Пользователи		Развития	Менеджеры по развитию	Получение нового удобного инструмента для постановки задач
Пользователи		Финансово-экономический отдел	Экономисты	Получение нового удобного инструмента для постановки задач
Пользователи	Ольга	Юридический	Юрисконсульты	Получение нового удобного инструмента для постановки задач

1.3 Сбор требований

1.3.1 Планирование

На подготовку организационных вопросов отведем один день. Сюда входит планирование, проведение встреч и интервью [6] с одним из сотрудников IT-отдела компании, наблюдением за работой его работой.

Еще три дня будут потрачены на проведение опросов среди коллег разработчиков и

обработку их результатов. Один день будет отведен за наблюдением работы сотрудника в приложении Битрикс24. Два дня будут отведены на извлечение требований путем изучения документов о Битриксе24. Учитывая необходимость дополнительного времени в качестве буфера, общая продолжительность сбора требований составляет 10 дней.

1.3.2 Подготовка к встрече

В качестве места встречи выбран офис, переговорная комната. Время – обеденный перерыв, ланч. Выбранный формат общения – личная встреча один на один, поскольку при данном формате происходит меньше потерь информации.

1.3.3 Подготовка к интервью

Для интервью были подготовлены следующие вопросы:

– Как давно вы используете Битрикс24 в качестве менеджера задач в финансовой логистике нашей компании?

– Какие ключевые особенности "Битрикс24" вы считаете наиболее ценными для управления задачами и проектами?

– Можете ли вы рассказать о конкретных проблемах или ограничениях, с которыми вы столкнулись при использовании Битрикс24 для нужд нашей компании?

– По вашему мнению, в каких областях можно улучшить Битрикс24 для более эффективной поддержки наших процессов финансовой логистики?

– Есть ли какие-то конкретные функциональные возможности или интеграции, которые, по вашему мнению, отсутствуют в Битрикс24, что повысило бы его эффективность для нашей компании?

– Исследовали ли вы какие-либо обходные пути или настройки в Битрикс24 для решения конкретных задач в нашей финансовой логистике? Если да, то не могли бы вы привести примеры?

– Есть ли какие-либо специфические проблемы с производительностью или узкие места, с которыми вы столкнулись при использовании Битрикс24? Как, по вашему мнению, их можно устранить?

– Можете ли вы предложить какие-либо конкретные модули или расширения в Битрикс24, которые можно использовать для оптимизации наших процессов финансовой логистики?

– Поскольку мы стремимся разработать новое приложение в дополнение к "Битрикс24", какие возможности или функции, по вашему мнению, должны быть приоритетными для повышения общей эффективности решения?

– Наконец, исходя из вашего опыта работы с Битрикс24 и операциями финан-

совой логистики нашей компании, есть ли у вас другие предложения или идеи по улучшению программы или ее интеграции с другими системами для достижения лучших результатов?

1.3.4 Интервью

Ответы на интервью приведены в таблице 2.

Таблица 2 – Результаты интервью по сбору требований

Вопрос	**Ответ сотрудника IT-отдела**
Как давно вы используете Битрикс24 в качестве менеджера задач в финансовой логистике нашей компании?	Я использую Битрикс24 в качестве менеджера задач в нашей компании в течение последних двух лет.
Какие ключевые особенности "Битрикс24" вы считаете наиболее ценными для управления задачами и проектами?	Ключевые особенности "Битрикс24", которые я считаю наиболее ценными для управления задачами и проектами, включают в себя надежные возможности управления задачами, интегрированные средства коммуникации для совместной работы, а также возможность устанавливать сроки и отслеживать прогресс.
Можете ли вы рассказать о конкретных проблемах или ограничениях, с которыми вы столкнулись при использовании Битрикс24 для нужд нашей компании? По вашему мнению, в каких областях можно улучшить Битрикс24 для более эффективной поддержки наших процессов финансовой логистики?	Одной из проблем, с которой мы столкнулись, были ограниченные возможности настройки для определенных рабочих процессов, специфичных для нашей финансовой логистики. Мы также периодически сталкивались с проблемами производительности в периоды пиковой нагрузки, что приводило к замедлению времени отклика.
Есть ли какие-то конкретные функциональные возможности или интеграции, которые, по вашему мнению, отсутствуют в Битрикс24, что повысило бы его эффективность для нашей компании?	Чтобы лучше поддерживать наши процессы финансовой логистики, я считаю, что "Битрикс24" может выиграть от улучшения функций автоматизации, таких как автоматические уведомления и напоминания для критических задач.

Вопрос	Ответ сотрудника IT-отдела
Исследовали ли вы какие-либо обходные пути или настройки в Битрикс24 для решения конкретных задач в нашей финансовой логистике? Если да, то не могли бы вы привести примеры?	В настоящее время в Битрикс24 отсутствуют встроенные интеграции с некоторыми инструментами финансовой логистики, такими как бухгалтерские программы и системы управления цепочками поставок. Наличие таких интеграций значительно повысило бы эффективность и точность данных.
Есть ли какие-либо специфические проблемы с производительностью или узкие места, с которыми вы столкнулись при использовании	Да, для решения наших специфических задач мы настроили Битрикс24, создав пользовательские поля и формы для сбора соответствующих финансовых данных. Мы
столкнулись при использовании Битрикс24? Как, по вашему мнению, их можно устранить?	соответствующих финансовых данных. Мы также разработали специальные отчеты для создания показателей финансовой деятельности, специфичных для наших логистических операций.
Можете ли вы предложить какие-либо конкретные модули или расширения в Битрикс24, которые можно использовать для оптимизации наших процессов финансовой логистики?	Битрикс24 предлагает различные модули и расширения, которые могут оптимизировать процессы финансовой логистики. Я предлагаю изучить модуль CRM для управления взаимоотношениями с клиентами, модуль "Документооборот" для эффективной работы с документами и модуль "Отслеживание времени" для точного учета времени, затраченного на выполнение различных логистических задач.

Вопрос	Ответ сотрудника IT-отдела
Поскольку мы стремимся разработать новое приложение в дополнение к "Битрикс24", какие возможности или функции, по вашему мнению, должны быть приоритетными для повышения общей эффективности решения?	Для повышения общей эффективности решения я считаю, что приоритетными будут такие функции, как расширенная отчетность и аналитика данных финансовой логистики, бесшовная интеграция с внешними финансовыми системами, а также мобильное приложение для оперативного доступа. Кроме того, включение более динамичной и визуально привлекательной функции визуализации задач.
Наконец, исходя из вашего опыта работы с Битрикс24 и операциями финансовой логистики нашей компании, есть ли у вас другие предложения или идеи по улучшению программы или ее интеграции с другими системами для достижения лучших результатов?	В дополнение к упомянутым улучшениям, я рекомендую изучить возможность интеграции "Битрикс24" с нашими инструментами аналитики и отчетности, чтобы получить более глубокое представление о результатах нашей финансовой логистики. Такая интеграция позволит нам принимать решения на основе данных и выявлять области для оптимизации.

1.3.5 Опросы

Следующим шагом был проведен опрос среди коллег разработчиков ПО [7]. Вопросы анкеты подготовлены с учетом того, чтобы выяснить следующие аспекты:

- важность постановки задач;
- наиболее популярный способ фиксирования задач;
- преимущества используемого способа фиксации личных задач;
- недостатки используемого способа фиксирования личных задач;
- на сколько важно количество посторонних функций;

Результаты опроса показали следующие результаты:

– все опрошенные оценивают важность постановки ежедневных задач на высоком уровне (рисунок 6);

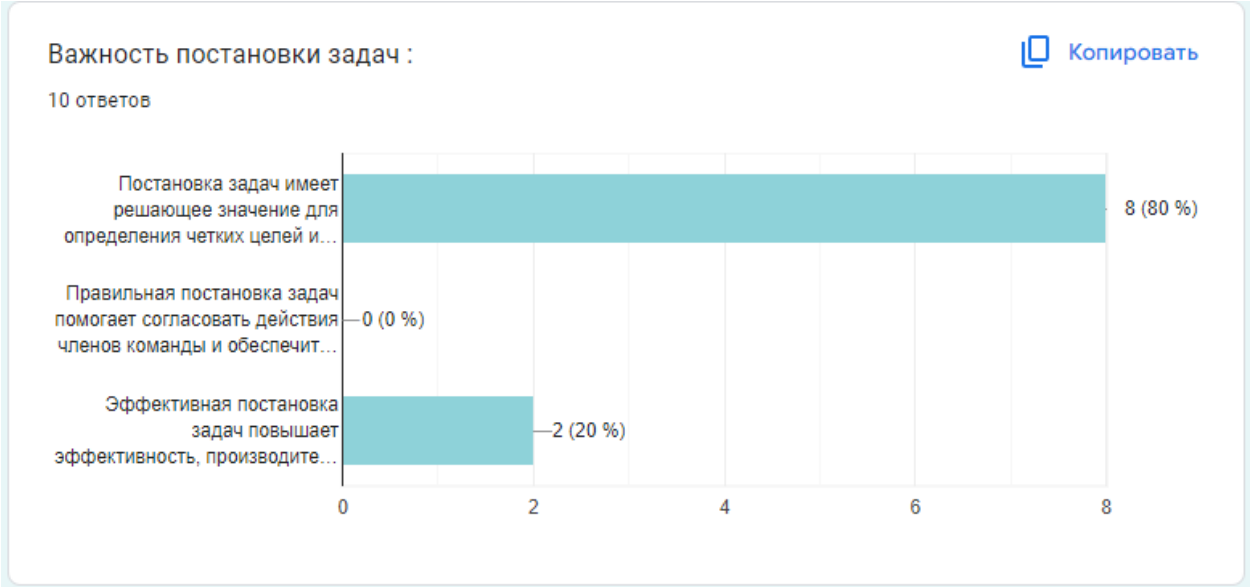


Рисунок 6 – Оценка важности постановки задач

– наиболее популярные способы постановки задач в частной компании – это использование комбинации программного обеспечения для управления проектами (например, Trello, Asana, ClikUP]) и платформ для командного общения (например, Slack, Microsoft Teams [7]). (рисунок 7);



Рисунок 7 – Самый популярный способ фиксации задач

– респонденты считают, что преимущества в постановке задач через приложение упорядоченная организация и расстановка приоритетов личных задач (рисунок 8);



Рисунок 8 – Преимущество метода, используемого для фиксации задач

– главные недостатки, на которые обратили внимание респонденты при фиксировании постановки задач через приложение – это балансировка личных и связанных с большим проектом задач может потребовать дополнительных усилий и управления временем (рисунок 9);



Рисунок 9 – Недостатки метода, используемого для фиксации задач

– как показывает опрос, для респондентов важно, чтобы приложение имело



количество посторонних функций должно быть сведено к минимуму, чтобы избежать беспорядка и сложности (рисунок 10).

Рисунок 10 – Важность количества посторонних функций

1.3.6 Наблюдение за работой сотрудником компании, использующим Битрикс24.

Формат наблюдения: онлайн, с помощью подключению к удаленному доступу из дома Наблюдение происходило за конкретным сотрудником компании, использующим Битрикс24 для получения, фиксирования и выполнения задач, а именно:

Сотрудник продемонстрировал его текущие задачи, их название, текущий статус, а также постановщика задачи и ответственного за ее исполнение (рисунок 11).

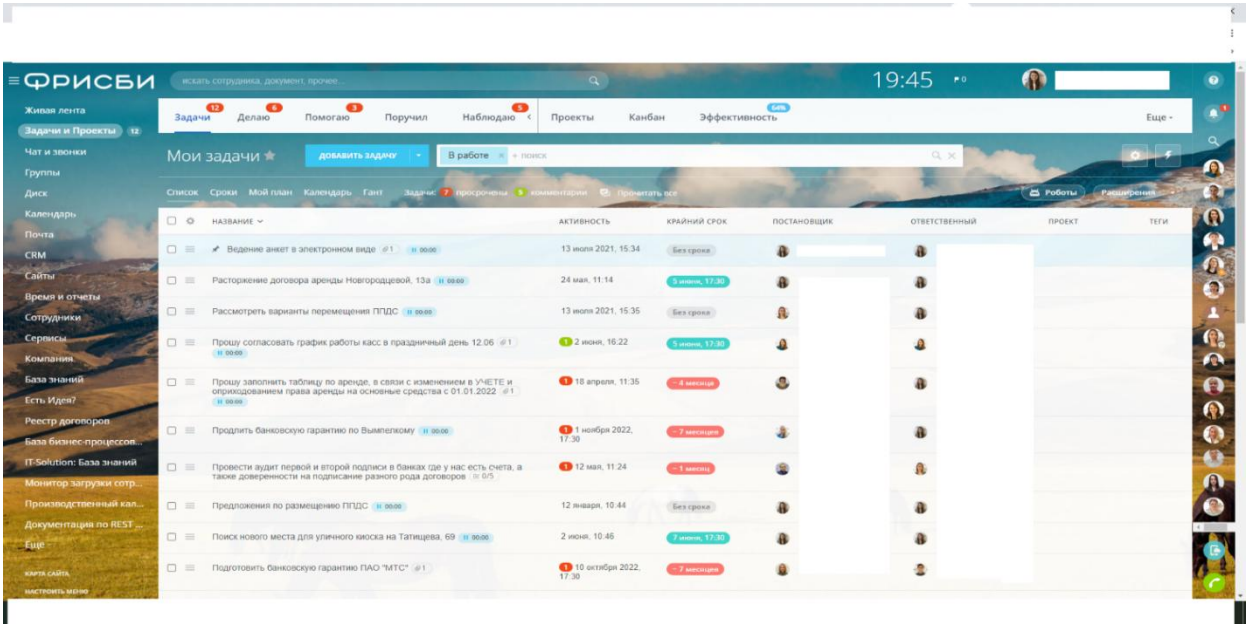
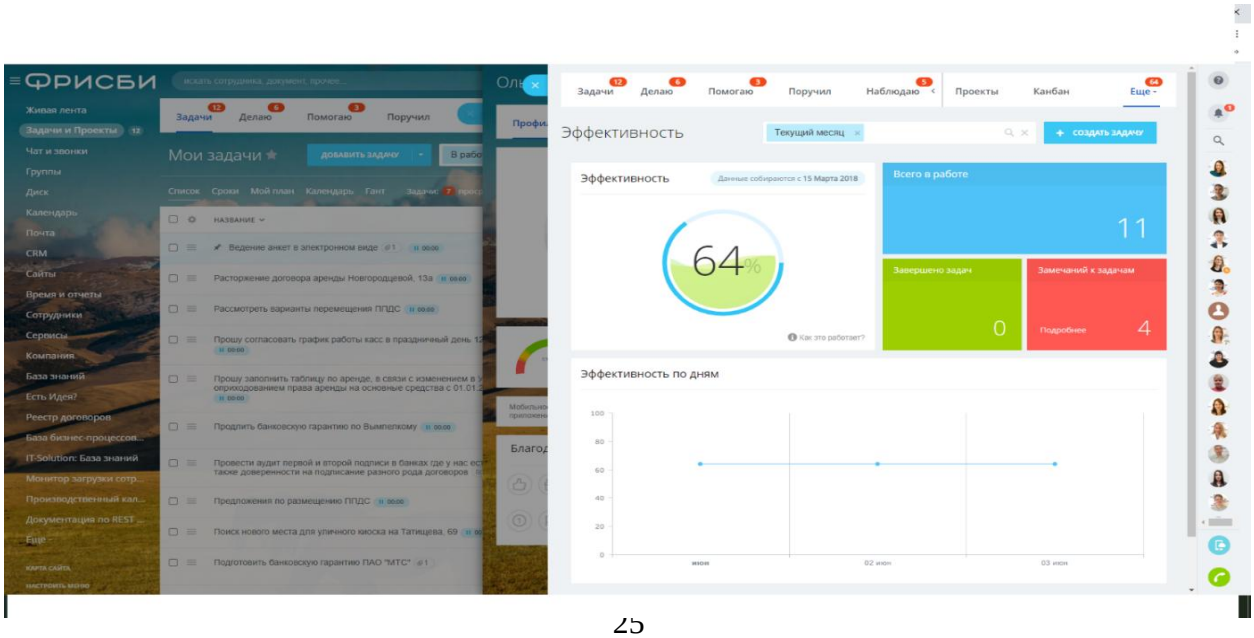


Рисунок 11 – Задачи сотрудника

Сотрудник продемонстрировал уровень своей эффективности за определенный



промежуток времени, показав количество выполненных и не выполненных задач (рисунок 12).

Рисунок 12 – Эффективность сотрудника

Сотрудник продемонстрировал выполнение одной из текущих задач, показав важные аспекты, а также комментарии от начальника отдела (рисунок 13).

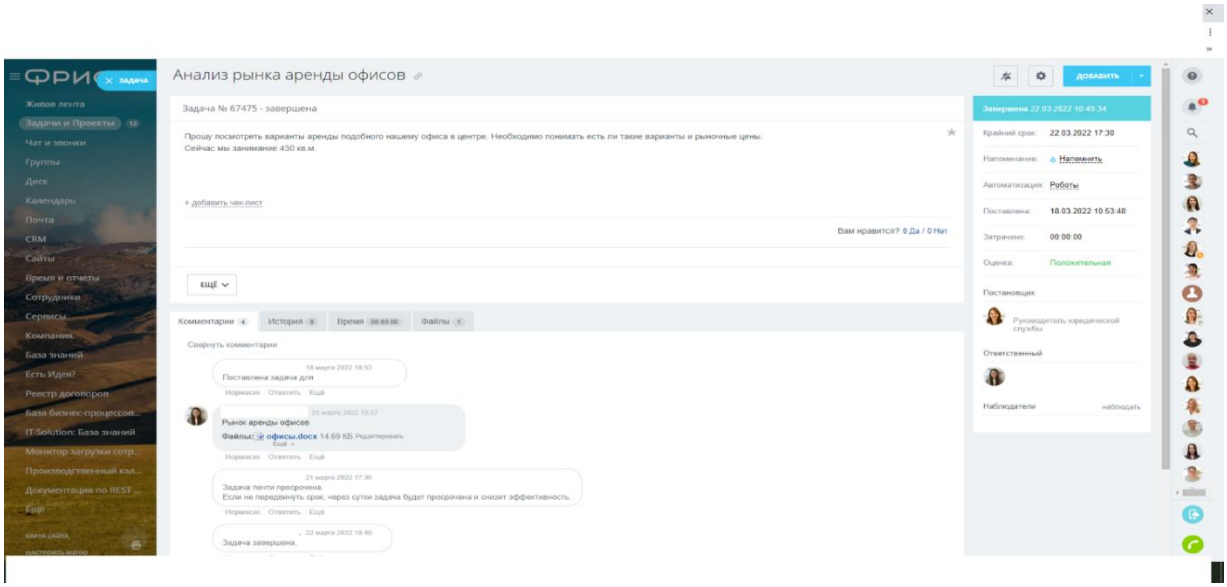
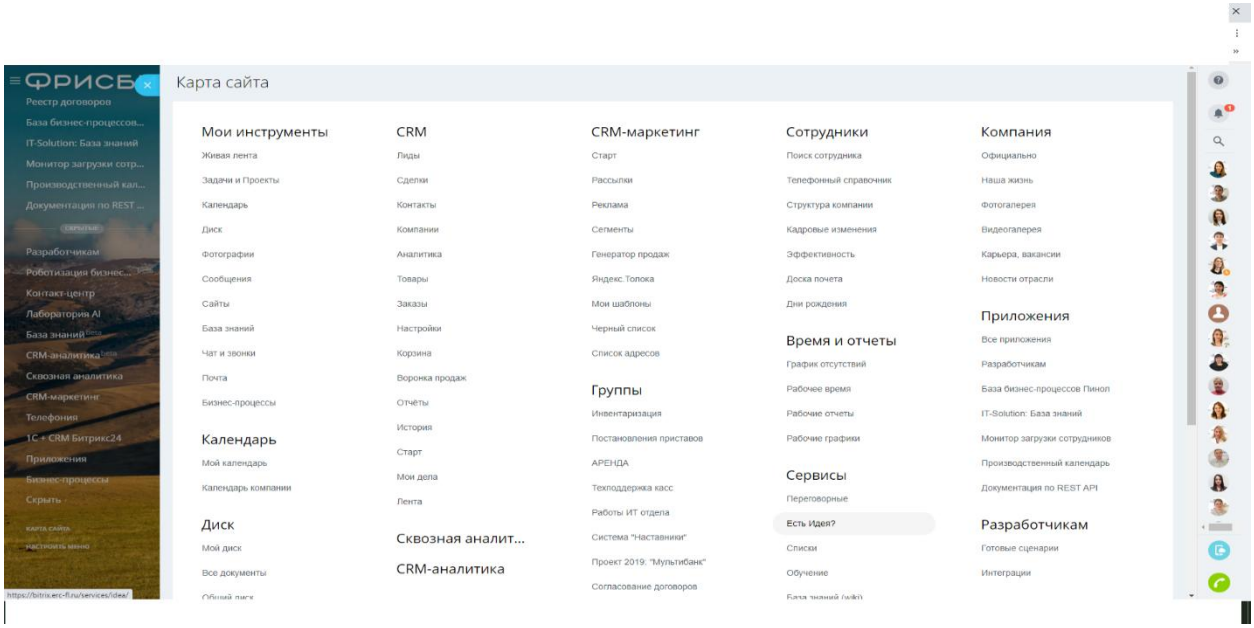


Рисунок 13 – Демонстрация выполнения текущей задачи

Сотрудник продемонстрировал карту сайта, на которой показано избыточное



количество функции наряду с постановкой задач (рисунок 14).

Рисунок 14 – Карта сайта

Таким образом, после наблюдения за работой сотрудника частной компании сложилось мнение, что у программы всё-таки имеются недостатки: из-за большого количества разнообразных функций в приложении может быть замедлена работа. Например, зависание одного из программных модулей, либо отсутствия подтверждения посылки сообщения

ощутимо сказывает на работе всех модулей этого программного обеспечения в том числе и модуля, отвечающего за постановку задач.

Разработка нового приложения, ориентированного непосредственно на управление задачами, могла бы удовлетворить эти специфические потребности. Приоритетами нового приложения могут стать удобный интерфейс, возможности настройки, бесшовная интеграция, расширенная отчетность и аналитика, автоматизация задач, расширенные полномочия пользователей. Уделяя внимание этим задачам, новое приложение обеспечит более специализированное и эффективное решение для операций финансовой логистики компании. Таким образом, необходимо полностью переделав данный модуль для Битрикс24, сделать его автономным.

1.3.7 Анализ документов

В данной лабораторной работе в качестве сбора требований об используемом ПО было выбрано непосредственно лицензионное соглашение Битрикс24 [8].

Настоящее лицензионное соглашение с конечным пользователем (далее –Соглашение) является юридически обязательным соглашением, заключаемым между компанией 1С-Битрикс и Вами –Конечным пользователем (физическим лицом, действующим от своего имени и в своих интересах, или юридическим лицом).

1С-Битрикс – Общество с ограниченной ответственностью «1С-Битрикс», 109544 Россия, г. Москва, б-р Энтузиастов, д.2, 13 эт., пом. 8-19.

Администратор (Администратор Портала) –физическое лицо, наделенное особыми правами по управлению Порталом, и либо являющееся Конечным пользователем, либо действующее от имени и в интересах Конечного пользователя.

Программа – программа для ЭВМ «1С-Битрикс24» (как в целом, так и ее компоненты), являющаяся представленной в объективной форме совокупностью данных и команд, в том числе, исходного текста, базы данных, аудиовизуальных произведений, включённых 1С-Битрикс в состав указанной программы для ЭВМ, а также любая документация по ее использованию.

Использование Программы – воспроизведение одной копии Программы посредством запуска в порядке, определенном пользовательской (технической) документацией и Соглашением.

Пользователь портала – любой Авторизованный пользователь или Экстранет-пользователь, который авторизовался в Программе как минимум один раз. Права Пользователя портала определяются исходя из функциональности Программы и настроек, определенных Администратором портала.

Авторизованный пользователь – пользователь Программы, в том числе Админи-

стратор портала, который в рамках настроек, определенных Администратором портала, может иметь доступ ко всей функциональности Программы. К указанным пользователям не относятся Экстранет-пользователи.

Экстранет-пользователь – пользователь Программы, который в рамках настроек, определенных Администратором портала, может иметь доступ только к разделу «Экстранет», созданному в целях использования модуля «Экстранет» в соответствии с п. 4.2 Соглашения.

Портал – информационный ресурс, являющийся совокупностью данных одной Программы с уникальным идентификатором, с помощью которого группируются объекты программы (информационные блоки, веб-формы, форумы, шаблоны, шаблоны писем и другие) для их совместного отображения и использования, обычно в одном внешнем виде, языке интерфейса, доменном имени или каталоге.

Пользовательские компоненты – разработанные и предлагаемые к установке третьими лицами расширения (компоненты) на условиях собственных соглашений об использовании, предоставляющие доступ к информации или сервисам третьих лиц, либо осуществляющие переход на ресурсы третьих лиц.

Техническая поддержка – мероприятия, осуществляемые 1С-Битрикс в установленных им пределах и объемах для обеспечения функционирования Программы, включая информационно-консультационную поддержку.

Договор – документ, на основании которого 1С-Битрикс (или иное лицо, имеющее соответствующие права) предоставил Конечному пользователю право использования Программы на возмездной основе.

Регистрация – действие, направленное на создание Учетной записи Администратора Портала в Программе.

Учетная запись – запись в системе 1С-Битрикс, хранящая данные, позволяющие идентифицировать Администратора Портала и Пользователя портала, использование которой осуществляется в соответствии с условиями Битрикс24.Network.

Личный кабинет – закрытая область Портала, позволяющая Администратору Портала осуществлять авторизацию новых пользователей, управлять их Профилями и настройками Портала. Для входа в Личный кабинет используются уникальное имя и пароль, определенные Администратором Портала при регистрации.

SIP-коннектор – дополнительный модуль Программы для ее интеграции с IP-АТС.

Демо-доступ – ограниченный период Использования Программы, в течение которого Администратору портала предоставляется доступ ко всем функциональным возможностям Программы в целях самостоятельного ознакомления, оценки и проверки их Адми-

нистратором портала.

Профиль – область/страница Портала, заполняемая Пользователем портала и содержащая данные, позволяющие его идентифицировать.

Сканер совместимости тарифных планов (Сканер совместимости) – инструмент, который предоставляет Администратору Портала возможность выявить те виды пользовательских данных, которые не могут быть перенесены при переходе на тип лицензии с меньшей функциональностью (низкий тип лицензии).

Подводя итог следует отметить, что для модернизации системы "Битрикс24" изучение лицензионного соглашения очень полезный ресурс ведь он дает чёткое понятие об администраторе ПО, также разграничивает конкретные роли и уровни доступа, определенные для Пользователя портала, Авторизованного пользователя и Пользователя экстрасети. Также лицензионное соглашение дает указания пользователям и разработчикам, которым важно следовать для обеспечения соответствия требованиям и дальнейшего использования функциональных возможностей программы. Соблюдая лицензионное соглашение и принимая новые обновления и функции, пользователи могут максимально использовать потенциал программы "Битрикс24".

1.4 Требования к структуре системы в целом

1.4.1 Перечень подсистем, их назначение и основные характеристики

1.4.1.1 Подсистема контроллера. Назначение подсистемы – отправлять запросы на изменение данных в БД после действий пользователя.

1.4.1.2 Подсистема модели. Назначение подсистемы – осуществлять изменения в БД согласно отправленным запросом от подсистемы контроллера.

1.4.1.3 Подсистема представления. Назначение подсистемы – отображать изменения в БД за счет отображения данных в интерфейсе пользователя.

1.4.2 Требования к способам и средствам обеспечения информационного взаимодействия компонентов системы

1.4.2.1 Информационное взаимодействие подсистем должно осуществляться через единое информационное пространство и посредством использования стандартизированных протоколов и форматов обмена данными.

1.4.2.1 Все компоненты подсистем должны функционировать в пределах одного устройства пользователя.

1.4.3 Требования к характеристике взаимосвязей создаваемой системы со смежными системами

Система должна быть функционально законченной и иметь возможность работать при подключении к серверу с данными.

1.4.4 Требования к режимам функционирования системы

Должна обеспечиваться работа в штатном режиме.

1.4.5 Перспективы развития, модернизации системы

При разработке системы должны быть предусмотрены возможности ее

последующей модернизации при минимальных временных и финансовых затратах по следующим направлениям:

- расширение прикладных функций;
- расширение системной платформы (перенос на мобильные устройства).

1.5 Требования к функциям, выполняемым системой

1.5.1 Функции, подлежащие автоматизации

1.5.1.1 Подсистема отображения данных (Data Display Subsystem)

Подсистема отображения данных предоставляет данные в формате, удобном для пользователя. Она отвечает за визуализацию всех элементов системы на уровне пользовательского интерфейса. Основные функции подсистемы:

- Отображение списка задач в виде карточек, таблиц или списков.
- Вывод дедлайнов и статусов задач.
- Визуализация структуры проекта, к которому относятся задачи.
- Возможность создания новых задач через формы ввода.
- Обеспечение удобного доступа к функциям редактирования задач, добавления комментариев и просмотра подробностей.

1.5.1.2 Подсистема управления задачами

Подсистема управления задачами выполняет ключевые операции по работе с задачами, включая их создание, изменение, удаление и мониторинг. Основные функции:

- Авторизация пользователей, подсистема выполняет аутентификацию и авторизацию на основе JSON Web Token (JWT).
- Просмотр текущих задач, отображение задач пользователя с указанием их статуса и связанных дедлайнов.
- Работа с задачами, для пользователей с ролью *admin* доступны CRUD-операции: создание, редактирование, удаление задач;
- Для пользователей с ролью *employee* доступен просмотр задач и изменение их статусов.
- Обеспечение безопасности, контроль доступа к данным и защита информации от несанкционированного доступа.
- Работа с комментариями, создание и просмотр комментариев к задачам для

авторизованных пользователей.

1.5.1.3 Подсистема управления данными (Data Management Subsystem)

Подсистема управления данными отвечает за взаимодействие с данными на уровне системы, обеспечивая их хранение, обработку и защиту. Основные функции:

– Сбор и хранение данных, вводимых пользователями.
– Подготовка данных для последующей визуализации.
– Шифрование данных для обеспечения их безопасности.
– Контроль доступа к данным через уровни прав пользователей.

1.5.1.4 Подсистема базы данных (Data Base Subsystem)

Подсистема базы данных выполняет функцию хранения всех данных, используемых системой. Основные данные в виде таблиц, к примеру такие как:

Tasks – хранение информации обо всех задачах пользователей, включая их статусы, дедлайны и авторов.

Users – информация обо всех зарегистрированных пользователях, включая их роли (admin или employee).

Comments – хранение комментариев, прикрепленных к задачам. Подсистема базы данных организует надежное хранение данных и их доступность для других компонентов системы.

1.5.2 Временной регламент реализации функций

Система должна обеспечить для пользователя по всем операциям время отклика не более трех секунд (при успешном соединении с сервером) с вероятностью 0,98.

1.5.3 Требования одновременности выполнения группы функций

Система не должна иметь принципиальных ограничений на количество одновременно работающих пользователей.

1.5.4 Критерии отказов

1.5.4.1 Критерии отказов для каждой разрабатываемой функции должны быть разработаны на этапе “Разработка подсистемы интеграции” совместно с полномочными представителями заказчика.

1.5.4.2 Должно быть обеспечено сохранение всей функциональности системы, доступной через программные интерфейсы, при отказе на уровне пользователя.

1.6 Требования к видам обеспечения

1.6.1 Требования к математическому обеспечению

Требования к математическому обеспечению не предъявляются.

1.6.2 Требования к информационному обеспечению

1.6.2.1 Структура хранения данных в системе должна состоять из следующих основных областей:

– область временного хранения данных;

– область постоянного хранения данных.

1.6.2.2 Информация в базе данных системы должна сохраняться при возникновении аварийных ситуаций, связанных со сбоями электропитания.

1.6.3 Требования к лингвистическому обеспечению

1.6.3.1 При реализации системы должны применяться следующие языки программирования: SQL, Java и др.

1.6.3.2 Для реализации алгоритмов манипулирования данными в хранилище данных необходимо использовать стандартный язык запроса к данным SQL.

1.6.3.3 Для организации диалога системы с пользователем должен применяться графический пользовательский интерфейс.

1.6.4 Требования к программному обеспечению (ПО)

1.6.4.1 Требования к клиентскому программному обеспечению:

Windows 10 или выше, с установленным клиентом Яндекс браузера.

1.6.4.2 Требования к обеспечению качества программного обеспечения.

1.6.4.2.1 Функциональность должна обеспечиваться выполнением подсистемами всех их функций.

1.6.4.2.2 Надежность должна обеспечиваться за счет предупреждения ошибок – недопущения ошибок в готовом ПО.

1.6.4.2.3 Эффективность должна обеспечиваться за счет принятия подходящих, верных решений на разных этапах разработки ПО и системы в целом.

1.6.4.2.4 Сопровождаемость должна обеспечиваться за счет:

– высокого качества документации по сопровождению;

– использования в программном тексте описания объектов и комментариев;

– использования осмысленных (мнемонических) и устойчиво различимых имен объектов;

– избеганием создания фрагментов текстов программ с неочевидным или скрытым смыслом.

1.6.4.3 На каждом этапе в разработке системы должна проводиться проверка правильности принятых решений по разработке и применению готовых программных средств.

1.6.4.4 Необходимость согласования разрабатываемых программных средств с фондом

алгоритмов и программ отсутствует.

1.6.5 Требования к техническому обеспечению

1.6.5.1 Минимальные требования к клиентскому устройству должны быть следующие:

– процессор 2 ГГц или выше;

– оперативная память не менее 4 Гб.

1.6.6 Требования к метрологическому обеспечению

Требования к метрологическому обеспечению не предъявляются.

1.6.7 Требования к методическому обеспечению

1.6.7.1 Исполнитель должен предоставлять пользователю эксплуатационную документацию – руководство пользователя.

1.6.7.2 Требования к оформлению руководства пользователя приведены в п. 9.

1.7 Общие технические требования к системе

1.7.1 Требования к численности и квалификации персонала и пользователей системы

1.7.1.1 Пользователями системы должны выступать, сотрудники предприятия, прошедшие обучение использованию Системы.

1.7.1.2 В состав персонала, выполняющего техническую эксплуатацию системы, должны входить:

– администратор системы;

– группа технической поддержки.

1.7.1.3 Администратор системы должен владеть общими принципами построения системы, способами архивации и аварийного восстановления БД, настройкой системы.

1.7.1.4 Группа технической поддержки должна знать настройку программной части, обладать знаниями и умением классифицировать и устранять возникающие ошибки.

1.7.1.5 Режим работы обслуживающего персонала должен определяться режимом работы организации.

1.7.1.6 Все пользователи системы должны быть «уверенными пользователями» операционной системы Windows.

1.7.2 Требования к надежности

Надежность создаваемой системы должна обеспечиваться:

– выбором отказоустойчивого оборудования;

– оперативным исправлением возникающих ошибок.

1.7.3 Требования по безопасности

Для обеспечения безопасности пользовательские данные не будут использоваться приложением.

1.7.4 Требования к эргономике и технической эстетике

1.7.4.1 В системе должен быть предусмотрен один вид пользовательского интерфейса – отдельный интерфейс пользователя.

1.7.4.2 При проектировании пользовательского интерфейса должны быть учтены правила графического дизайна.

1.7.4.3 При проектировании окна должно быть учтено, при каком разрешении, а также размерах устройства и шрифтов будут работать пользователи.

1.7.4.4 Заголовки окон должны быть короткие и адекватные содержимому окна.

1.7.4.5 Заголовки должны соответствовать названиям элементов, при помощи которых окна были вызваны.

1.7.4.6 Если окно вызывается элементом, не имеющим явного названия, в заголовке окна должно отражаться название экранной формы.

1.7.4.7 Тип окна (модальное, немодальное, возможность минимизации / максимизации) должен быть выбран осознанно, в соответствии с задачами пользователей.

1.7.4.8 Управляющие и информационные элементы должны быть расположены достаточно далеко друг от друга.

1.7.4.9 Информация в окне должна быть адекватно сгруппирована (связанные элементы объединены в группы).

1.7.4.10 Кнопки должны находиться в секции, на которую они оказывают непосредственное воздействие.

1.7.4.11 Терминационные кнопки, управляющие окном, должны быть расположены либо снизу в ряд, либо справа в колонку.

1.7.4.12 Переход от элемента к элементу внутри окна должен осуществляться сверху вниз и слева направо.

1.7.4.13 В диалоговых окнах должно отсутствовать меню или инструментальные панели.

1.7.4.14 Диалоговые окна должны открываться не в центре экрана, а в центре текущего действия пользователя.

1.7.4.15 Пункты главного меню должны иметь адекватные названия.

1.7.4.16 Первая буква в названии пунктов должна быть заглавная.

1.7.4.17 Все пункты первого уровня должны активизировать выпадающее меню.

1.7.4.18 Раскрывающиеся меню и элементы основного меню второго уровня должны начинаться с заглавной буквы.

1.7.4.19 Если в меню используются пиктограммы, они должны быть расположены слева от названия пункта меню.

1.7.4.20 Все списки должны содержать более одного элемента.

1.7.4.21 Высота меню не должна превышать размер экрана, чтобы меню не нужно было прокручивать.

1.7.4.22 Пункты меню должны быть адекватно сгруппированы, разделители в меню – осмысленно использованы.

1.7.4.23 Пункты меню должны быть расположены в порядке связанности выполняемых функций, частоты использования и важности.

1.7.4.24 Должно быть использовано не более двух подуровней меню.

1.7.4.25 Название пункта меню должно соответствовать названию вызываемого окна.

1.7.4.26 Пункты меню, открывающие диалоговые окна, должны быть обозначены в конце многоточием (...).

1.7.4.27 Недоступные пункты меню должны быть обозначены серым цветом шрифта.

1.7.4.28 Каждому пункту всплывающего меню должен соответствовать аналогичный пункт в основном меню.

1.7.4.29 Элементы должны быть упорядочены и сгруппированы в соответствии с задачами пользователей.

1.7.4.30 Для стандартных действий должны использоваться общепринятые графические элементы.

1.7.4.31 Число переключателей (Check Boxes) в одном окне должно быть не более десяти.

1.7.4.32 Переключатели должны быть сгруппированы и каждой группе присвоено название.

1.7.4.33 Внутри группы переключатели должны быть расположены строго вертикально.

1.7.4.34 Переключатели не должны применяться для частого, оперативного использования.

1.7.4.35 В названиях должна использоваться только позитивная, утвердительная форма.

1.7.4.36 Командные кнопки должны иметь краткие и ясные названия.

1.7.4.37 В каждом диалоге должно использоваться не более шести кнопок.

1.7.4.38 Кнопки, выполняющие в разных диалогах идентичные функции, должны иметь одинаковые названия.

1.7.4.39 Типовые кнопки должны иметь общепринятые названия.

1.7.4.40 Кнопки, вызывающие продолжение диалога во вложенных формах, должны быть обозначены многоточием (...).

1.7.4.41 Недоступные кнопки должны иметь соответствующие атрибуты (серый цвет

шрифта и т. п.).

1.7.4.42 Опасные для пользователя кнопки не должны являться кнопками по умолчанию.

1.7.4.43 Редактируемые поля со списком (Combo Box) должны иметь функцию авто выбора.

1.7.4.44 Раскрывающиеся списки по высоте выводимого на экран списка должны быть ограничены 3 – 8 элементами.

1.7.4.45 Если список содержит более 50 элементов, должен использоваться фильтр или режим поиска.

1.7.4.46 Если все элементы не умещаются в одном фрагменте списка, должна автоматически появляться полоса прокрутки.

1.7.4.47 Каждая группа элементов должна иметь осмысленное название, помимо рамки отделена от других групп и элементов свободным пространством.

1.7.4.48 В одной группе должно использоваться не более шести кнопок выбора (Option Buttons или Radio Buttons).

1.7.4.49 В пределах группы кнопки должны быть расположены по вертикали.

1.7.4.50 Не должно быть состояния, когда ни одна кнопка не выбрана.

1.7.4.51 Последовательность расположения кнопок в группе должна учитывать частоту использования.

1.7.4.52 Названия вкладок должны быть выровнены по центру.

1.7.4.53 Каждой вкладке должно быть присвоено осмысленное название.

1.7.4.54 Количество рядов закладок не должно превышать двух.

1.7.4.55 Все связанные между собой данные должны находиться внутри одной закладки.

1.7.4.56 Кнопки, относящиеся ко всему блоку закладок, должны быть расположены за пределами блока закладок.

1.7.4.57 Для недоступных полей ввода (Text Box или Edit Field) должен использоваться серый цвет (название, текст и фон поля).

1.7.4.58 Высота всех текстовых полей в окне должна быть одинакова.

1.7.4.59 Содержимое полей должно быть выровнено по левому краю, за исключением полей с числовыми значениями (например, для вывода денежных сумм).

1.7.4.60 Длина поля должна быть не меньше длины вводимых в него данных.

1.7.4.61 Если в поле вводится численное значение, то границы диапазона должны выводиться во всплывающей подсказке.

1.7.4.62 При открытии окна фокус ввода должен попадать на элемент внутри окна.

1.7.4.63 Схема табуляции должна соответствовать очередности заполнения полей (слева направо, сверху вниз).

1.7.4.64 Командные кнопки должны быть включены в табуляцию.

1.7.4.65 Невидимые и недоступные элементы должны быть исключены из схемы табуляции.

1.7.4.66 Направление теней во всех пиктограммах должно быть одинаково: слева сверху.

1.7.4.67 Система, завершив какую-либо длительную операцию, должна об этом оповещать пользователя через сообщение на экране монитора.

1.7.4.68 Необходимо избегать взаимодействия с пользователем в середине длительных операций.

1.7.4.69 Если избежать взаимодействия с пользователем не удается, необходимо предусмотреть действие по умолчанию при отсутствии реакции пользователя в течение 2–3 минут.

1.7.4.70 При разработке мобильного приложения должен быть обеспечен уровень соответствия АА (средний) документу ГОСТ Р 52872–2019.

1.7.5 Требования к эксплуатации, техническому обслуживанию, ремонту и хранению компонентов системы

1.7.5.1 Работа системы должна обеспечиваться только в штатном режиме.

1.7.5.2 При выходе системы из строя, самой системой, а в случае ее полного отказа – системным администратором должны быть приняты меры по блокировке системы для доступа пользователей, с последующим немедленным резервным копированием информации из БД «как есть».

1.7.5.3 Авторские права на программное обеспечение, на архитектуру системы и модели данных принадлежат разработчику и защищаются действующим законодательством.

1.7.5.4 Пользователь не имеет права передавать программное обеспечение третьим лицам, самостоятельно или через третьих лиц использовать систему в качестве образца для разработки аналогичного программного обеспечения.

1.7.6 Требования к защите информации от несанкционированного доступа

Доступ к системе и хранимой в ней информации может предоставляться любым авторизованным пользователям.

1.7.7 Требования по сохранности информации при авариях

1.7.7.1 Система должна обеспечивать сохранность накопленной информации при любых авариях, не влекущих за собой поломку носителя (например, потеря питания, отказ технических средств и т. п.).

1.7.7.2 Должна быть обеспечена логическая целостность данных за счет использования транзакций.

1.7.7.3 Все команды, определенные в одной транзакции, должны обрабатываться как единое целое. Таким образом, будет обеспечена гарантия, что зафиксированными могут оказаться либо все изменения, выполненные транзакцией, либо данные будут восстановлены в состояние, в котором они были до начала транзакции.

1.7.8 Требования к патентной чистоте и патентоспособности

1.7.8.1 По всем техническим и программным средствам, применяемым в системе, должны соблюдаться условия лицензионных соглашений.

1.7.8.2 По всем техническим и программным средствам, применяемым в системе, должна обеспечиваться патентная чистота.

1.7.9 Дополнительные требования

Система должна эксплуатироваться на уже имеющихся у пользователей устройствах.

2 Анализ поставленной задачи

2.1 Анализ требований к системе

2.1.1 Анализ требований к структуре в целом

Раздел "Требования к структуре системы в целом" определяет общую организацию подсистем, их назначение и характеристики. Основные компоненты системы включают подсистему контроллера, модели и представления. Эти компоненты взаимодействуют для обеспечения корректной работы системы. Такой подход позволяет обеспечить надежность, масштабируемость и возможность дальнейшей модернизации. Основные преимущества структуры включают четкое разделение обязанностей между компонентами и возможность упрощенной интеграции новых модулей [9].

2.1.2 Анализ требований к функциям, выполняемым системой

Анализ требований к автоматизируемым функциям показывает, что система должна обеспечивать полноценное управление задачами, включая их создание, изменение, удаление и мониторинг и включать в себя следующие подсистемы:

а) Подсистема отображения данных, обеспечивает визуализацию задач и структуры проекта в удобном для пользователя формате (карточки, таблицы). Также реализуется возможность работы с формами ввода и редактирования.

б) Подсистема управления задачами, отвечает за CRUD-операции с задачами для администраторов, а для сотрудников с ролью employee – за просмотр и обновление статусов задач.

в) Подсистема управления данными**,** взаимодействует с данными, обеспечивая их хранение, обработку и защиту.

г) Подсистема базы данных**,** хранит задачи, пользователей и комментарии. Эти требования обеспечивают функциональную полноту системы, делая её удобной и безопасной для использования.

2.1.2.1 Временной регламент реализации функций

Система должна обеспечивать скорость отклика не более трех секунд при стабильном соединении. Это требование повышает удобство использования системы и снижает задержки при выполнении операций.

2.1.2.2 Одновременность выполнения группы функций

Система проектируется с учетом возможности многопользовательской работы без ограничений на число одновременно активных пользователей. Это гарантирует стабильность работы в условиях растущей нагрузки.

2.1.2.3 Критерии отказов

Требования предполагают сохранение функциональности системы при отказах на уровне пользователя. Это достигается через устойчивую архитектуру и использование надежных механизмов обработки ошибок.

2.1.3 Анализ требований к видам обеспечения

2.1.3.1 Математическое обеспечение

Отдельные математические алгоритмы в разработке системы не используются, так как задачи, возлагаемые на систему, не требуют специфических математических расчетов.

2.1.3.2 Информационное обеспечение

Требования к информационному обеспечению предполагают создание областей временного и постоянного хранения данных. В базе данных должна быть реализована защита от потерь данных при авариях, что достигается использованием транзакций и регулярного резервного копирования.

2.1.3.3 Лингвистическое обеспечение

Для реализации системы применяются языки программирования Java и SQL. Пользовательский интерфейс разрабатывается с использованием графических инструментов, поддерживающих удобный диалог между пользователем и системой.

2.1.3.4 Программное обеспечение

Система должна быть совместима с Windows 10 и выше, а также с браузером Яндекс. ПО должно обеспечивать высокую функциональность, надежность, эффективность и сопровождаемость, что закладывается на всех этапах разработки.

2.1.3.5 Техническое обеспечение

Клиентские устройства должны иметь процессор не менее 2 ГГц и оперативную память не менее 4 ГБ. Эти требования соответствуют задачам системы и обеспечивают её стабильную работу.

2.1.3.6 Методическое обеспечение

Пользователи должны получать руководство, описывающее принципы работы системы. Это способствует быстрому освоению системы сотрудниками компании.

2.1.4 Анализ общих технических требований к системе

2.1.4.1 Требования к численности и квалификации персонала

Система ориентирована на использование обученными сотрудниками и технической поддержкой. Администратор должен владеть принципами работы с базой данных и настройкой системы.

2.1.4.2 Надежность

Надежность системы достигается использованием отказоустойчивого оборудования

и оперативным исправлением ошибок, что минимизирует риски отказа.

2.1.4.3 Безопасность

Система защищает пользовательские данные от несанкционированного доступа через механизм авторизации и безопасное хранение информации.

2.1.4.4 Эргономика и техническая эстетика

Пользовательский интерфейс разрабатывается с учетом удобства использования и современных стандартов графического дизайна. Обеспечивается логичная структура меню и полей ввода.

2.1.4.5 Эксплуатация и обслуживание

Система должна работать в штатном режиме, с возможностью оперативного резервного копирования данных в случае сбоев. Авторские права на ПО принадлежат разработчику, что предотвращает несанкционированное использование.

2.1.4.6 Сохранность информации при авариях

Использование транзакций обеспечивает целостность данных. В случае аварийных ситуаций система должна гарантировать восстановление информации до последнего сохраненного состояния.

2.1.4.7 Патентная чистота

ПО и применяемые технологии соответствуют лицензионным соглашениям, что исключает нарушение авторских прав.

2.2 Обоснование архитектуры системы

2.2.1 Сравнение монолитной и распределенной архитектур

При проектировании программного обеспечения важным этапом является выбор архитектуры, которая будет определять структуру системы, принципы взаимодействия ее компонентов и подходы к масштабированию. Для определения оптимального подхода необходимо рассмотреть основные архитектурные подходы, такие как монолитная и распределенная архитектуры, с учетом их преимуществ и недостатков.

Монолитная архитектура – представляет собой единое приложение, которое содержит все функциональные компоненты – пользовательский интерфейс, бизнес-логику и взаимодействие с базой данных [10].

Преимущества:

– Простота разработки и развертывания: вся система разрабатывается как одно приложение.

– Низкая сложность настройки: в отличие от распределенных систем, не требуется организация взаимодействия между независимыми сервисами.

– Легкость тестирования: отсутствие необходимости координировать компоненты или проверять сетевые взаимодействия между сервисами.

– Производительность: при грамотной реализации монолитные системы могут демонстрировать высокую скорость работы благодаря отсутствию сетевой задержки между компонентами.

Недостатки:

– Ограниченная масштабируемость: вся система масштабируется целиком, даже если нагрузки требуют оптимизации только отдельных модулей.

– Сложность сопровождения при большом объеме кода: при увеличении функциональности системы структура монолита может стать запутанной.

Распределенная архитектура – включает множество отдельных сервисов (микросервисов), которые взаимодействуют между собой через сеть.

Преимущества:

– Высокая гибкость: возможность масштабировать отдельные модули или сервисы.

– Независимость разработки: разные команды могут параллельно разрабатывать и обновлять отдельные модули.

– Надежность: отказ одного сервиса не приводит к полной остановке системы.

Недостатки:

– Сложность разработки: требуется настройка взаимодействия сервисов, использование API и решение вопросов совместимости.

– Зависимость от сети: производительность системы может снижаться из-за сетевой задержки.

– Увеличение затрат: разработка и обслуживание распределенной системы требуют больше ресурсов.

Обоснование выбора монолитной архитектуры

Для данного проекта была выбрана монолитная архитектура, так как она лучше соответствует текущим задачам системы:

Простота реализации: Проект ограничен по времени и ресурсам. Монолитный подход позволяет быстрее достичь результата.

Структурированность: Многоуровневая архитектура обеспечивает логическое разделение на слои (например, слой отображения, бизнес-логики и данных), что облегчает сопровождение.

Низкие эксплуатационные затраты: Отсутствие необходимости координации между

сервисами позволяет упростить процесс развертывания и поддержки.

Подготовка к возможному переходу на микросервисы: В будущем, при увеличении масштабов проекта, монолитная система может быть реорганизована в распределенную архитектуру.

2.2.3 Причины выбора монолитной многоуровневой архитектуры

Причинами выбора монолитной многоуровневой архитектуры является:

а) Централизованность и согласованность, единая структура обеспечивает целостность данных и согласованность всех процессов в рамках одного приложения.

б) Логическое разделение ответственности, деление системы на уровни четко разграничивает задачи каждого из них, например:

1) UI-уровень отвечает за интерфейс.
2) Бизнес-логика – за обработку данных.
3) DAL – за управление базой данных.

в) Простота разработки и развертывания, вся система разрабатывается как одно приложение, что значительно упрощает процесс развертывания и тестирования. Не требуется управление несколькими сервисами.

г) Гибкость и подготовка к масштабируемости логическое разделение на уровни позволяет адаптировать архитектуру в будущем. При необходимости монолит можно рефакторить в микросервисную архитектуру.

д) Следование принципам проектирования, как указано в учебнике Марка Ричардса [10], многоуровневая архитектура обеспечивает соблюдение принципа Separation of Concerns (разделение обязанностей), что упрощает дальнейшее сопровождение и масштабирование.

В совокупности, выбор монолитной архитектуры обусловлен следующими факторами**:**

а) Экономическая целесообразность. Для небольшой компании монолитная архитектура является наиболее подходящим вариантом с точки зрения затрат на разработку, внедрение и эксплуатацию. Она не требует дополнительных расходов на инфраструктуру, организацию взаимодействия между микросервисами, а также найм специалистов с опытом работы с распределенными системами.

б) Скорость реализации проекта. Разработка монолитного приложения занимает меньше времени, так как отсутствует необходимость проектировать и поддерживать сложные механизмы взаимодействия между сервисами. Это позволяет быстрее внедрить систему в эксплуатацию и начать её использование.

в) Соответствие текущим требованиям. На данном этапе развития компании объ-

ем данных, нагрузка и количество пользователей не создают предпосылок для возникновения узких мест, характерных для монолитных систем.

Таким образом, выбранная архитектура обеспечивает баланс между функциональностью, затратами и требованиями компании. Вместе с тем, логическое разделение системы на уровни создаёт основу для её дальнейшей модернизации, включая возможный переход на микросервисную архитектуру в случае необходимости [11].

2.2.2 Уровни монолитной многоуровневой архитектуры

В рамках данного проекта система реализована с разделением на три основных уровня:

а) Уровень пользовательского интерфейса (UI) этот уровень отвечает за взаимодействие с пользователем. Этот уровень включает в себя все элементы frontend, такие как веб-страницы, формы и интерфейсы для ввода/вывода данных. Уровень UI абстрагирован от бизнес-логики, что упрощает изменение внешнего вида без влияния на внутреннюю функциональность.

б) Уровень бизнес-логики этот уровень является центральным уровнем системы, который отвечает за обработку данных и реализацию бизнес-правил. Здесь происходят все расчеты, проверки и основное взаимодействие между слоями данных и пользовательским интерфейсом. Бизнес-логика изолирована, что позволяет гибко изменять правила работы системы.

в) Уровень доступа к данным (DAL) этот уровень управляет взаимодействием с базой данных. Он реализует функции чтения, записи, обновления и удаления данных. Изолирование операций с данными обеспечивает надежность и защищает бизнес-логику от изменений в структуре хранения.

2.3 Выбор инструментальных средств

Для реализации серверной части приложения был выбран язык программирования **Java** [12] в сочетании с фреймворком **Spring Boot** [13]. Java обеспечивает высокую производительность и стабильность, а Spring Boot позволяет быстро разрабатывать **RESTful API** [14], которые необходимы для взаимодействия клиентской и серверной частей. Дополнительно был использован **Spring Security** [15] для обеспечения безопасности системы, что особенно важно в финансовой сфере.

В качестве базы данных выбрана реляционная СУБД **MySQL 8.0**. Она отличается надежностью, высокой скоростью обработки запросов и поддержкой транзакций, что гарантирует целостность данных. **MySQL** [16] также хорошо интегрируется с Java через **Spring Data JPA**, что упрощает управление данными.

Для клиентской части приложения был использован **Angular** совместно с **Angular**

Material [17]. Angular позволяет создавать одностраничные приложения (SPA) с высокой производительностью, а Angular Material предоставляет готовые компоненты для создания удобного и современного пользовательского интерфейса [18].

Для обеспечения безопасности пользовательских данных и аутентификации был внедрен механизм валидации токенов с использованием **JSON Web Token (JWT)** [19]. Эта технология позволяет гарантировать, что доступ к данным имеет только авторизованный пользователь, что критически важно для систем, работающих с конфиденциальной информацией.

Интеграция всех компонентов осуществлялась в средах разработки **IntelliJ IDEA** для backend и **Visual Studio Code** для frontend. Эти IDE обеспечивают удобство написания и тестирования кода, а также поддержку инструментов контроля версий, таких как Git.

Таким образом, выбранные инструментальные средства полностью соответствуют требованиям к разрабатываемой системе и позволяют создать надежное, масштабируемое и безопасное решение.

2.4 Анализ требований к интерфейсу пользователя

Анализ проводиться с использованием метода персонажей А. Купера [20]. Интерфейс пользователя должен включать следующие элементы: регистрацию и авторизацию сотрудников, управление профилем пользователя, создание и управление задачами, поиск задач по различным критериям, контроль за сроками выполнения задач, а также систему уведомлений о статусе выполнения и важных обновлениях по задачам.

2.4.1 Базовые бизнес-требования

ИС должна обеспечивать [21]:

- быстрое фиксирование задач;
- внесение изменений в ранее поставленную задачу;
- возможность поиска задач;
- возможность в выборе исполнителя;
- возможность установки срока задачи;
- возможность установки статуса задачи;
- возможность комментирования текущих задач;
- возможность изменение статуса задачи;
- возможность удаления задачи;

2.4.2 Ольга (ключевой персонаж)

Имя: Ольга (рисунок 15).

Возраст: 28 лет.

Образование: высшее.

Должность: Начальник юридического отдела

Может работать удаленно, но в основном в офисе, распределяет задачи отдела между работниками, часто говорит по телефону с контрагентами.

2.4.3 Бизнес-цели

Бизнес цели ключевого персонажа включают в себя следующие пункты:

– успеть выполнить все запланированные на день дела;

– иметь представление о статусе всех задач;

– эффективно управлять своим временем;

– быстро реагировать на возникающие задачи.

2.4.4 Персональные цели

Персональные цели ключевого персонажа включают в себя:

– соблюдение жизненного баланса работа-жизнь;

– эффективное распределение своего рабочего времени;

– трата меньшего количества времени на изучение и анализ задач.



Рисунок 15 – Ольга, ключевой персонаж

2.4.5 Сценарии работы

Ежедневные действия ключевого персонажа на работе:

– провести мониторинг активных задач работников отдела;

– внести изменения в уже поставленные задачи;

– изменить статус задач;

– следить за сроками задач сотрудников;

– выбрать исполнителей для новый задач.

2.4.6 Типичный день

Ольга, начальник юридического отдела в частной компании, управляет различными юридическими проектами и распределяет задачи среди сотрудников. Однако она сталкивается с проблемой сложной и громоздкой системы управления задачами, интерфейс которой затрудняет эффективное распределение и контроль за выполнением задач.

Каждое утро Ольга начинает с проверки электронной почты, чтобы просмотреть входящие запросы и определить приоритетность задач для своего отдела. К сожалению, существующая система управления задачами усложняет её работу, что затрудняет быструю постановку задач для команды. Ольге приходится вручную отслеживать статус задач, а также вводить обновления по их выполнению.

Вскоре после начала рабочего дня Ольге звонит контрагент, срочно запрашивая отправку согласованной версии договора аренды и указывая на необходимость оплаты обеспечительного платежа в течение пяти рабочих дней с момента подписания. Ольга быстро заносит задачу на лист бумаги, чтобы не потерять важную информацию, а затем поручает одному из юристов подготовить документы и передать их экономисту, ответственному за оплату. Несмотря на трудности с системой, она оперативно расставляет приоритеты и контролирует выполнение задач через электронную почту.

Тем временем Ольга получает неожиданный звонок от инженера, отвечающего за техническое обслуживание пунктов приема платежей. Инженер сообщает о необходимости срочно согласовать работы по монтажу интернет-кабеля с арендодателем. Ольга понимает, что требуется отправить важный документ в течение дня, чтобы не допустить задержек, но система управления задачами не позволяет ей быстро проверить статус документа. Она делает пометку в ежедневнике, напоминая себе вернуться к этому вопросу во время обеда и проследить, чтобы нужный документ был отправлен вовремя.

Во время короткого перерыва на кофе Ольга получает сообщение от одного из юристов, который сообщает о необходимости встретиться и обсудить проект закрытия некоторых пунктов приема платежей. Несмотря на ограничения системы, Ольга решает подготовить юридическую справку по условиям досрочного расторжения договоров аренды. Ей приходится вернуться к рабочему месту и собрать информацию вручную, так как громоздкая система управления задачами затрудняет получение нужных данных.

К обеду Ольга проверяет электронную почту за предыдущий день и просматривает задачи команды на доске. Она замечает, что некоторые задачи потеряли актуальность, а другие требуют пересмотра сроков. Разочарованная ограничениями системы, Ольга решает обновить и перераспределить задачи на доске вручную, чтобы избежать сложностей с системой.

Несмотря на трудности, Ольга остаётся сосредоточенной на эффективном управлении своим отделом и постоянно ищет способы оптимизировать рабочий процесс, чтобы поддерживать высокую производительность команды [22].

2.4.7 Состав и внутренняя структура ключевых задач персонажа

Структура ключевых задач персонажа изображена в таблице №3.

Таблица 3 – Состав и внутренняя структура ключевых задач персонажа

№	Задача	Действия	Контекст	Критерий выполнения
1	Узнать активные задачи	Посмотреть на список задач	Произвольный	Пользователь узнал активные задачи
2	Зафиксировать задачу	Записать новую задачу, выбрать исполнителя	В процессе разговора с коллегами, при общении по телефону	Пользователь узнал о своей задаче
3	Внести изменения в ранее поставленную задачу	Указать новую информацию, сохранить изменения	В процессе разговора с коллегами, при общении по телефону	Пользователь узнал об изменениях
4	Внести задачу по направлению	Выбрать нового исполнителя	В процессе разговора с коллегами, при общении по телефону	Пользователь узнал о своей задаче
5	Изменить статус задачи	Внести изменения в статусе для пользователя	В процессе разговора с коллегами, при общении по телефону	Пользователь узнал об изменении статуса
6	Найти задачу по наименованию договора	Ввести название договора	Произвольный	Пользователь увидел карточку задачи
7	Оставить комментарии для исполнителя	В определенной задаче добавить к описанию важные данные	Произвольный	Пользователь увидит комментарии

2.4.8 Требования персонажа к программе

Программа должна:

– показывать активные задачи работников отдела;

– позволять быстро фиксировать и распределять задачи внутри отдела;

– отображать статус задачи;
– отображать исполнителя задачи;
– отображать сроки выполнения задачи;
– отображать комментарии по задачи;
– позволять осуществлять поиск по наименованию задач;
– осуществлять детальный просмотр задач;
– позволять менять статус, дату исполнения и исполнителя;
– позволять быстро удалять выполненные пользователем задачи.

2.4.9 Важность пользовательских требований

В таблице 4 показана характеристика значений важности.

Таблица 4 – Характеристика используемых значений важности

Значение критерия	**Характеристика критерия**
5	Персонаж испытывает необходимость в данной возможности
4	Персонаж не отказался бы от такой возможности
3	Персонажу не мешает данное требование
2	Персонажу мешает данное требование
1	Данное требование значительно затрудняет работу персонажа

В таблице 5 отражены требования и важность пользовательских требований

Таблица 5 – Требования и важность пользовательских требований

Требование	**Итого**
Наглядно показывать активные задачи	5
Подробное изучение поставленных задач	5
Отображение выполненных задач	5
Выполнять поиск задач	5

2.4.9.1 Частота востребования функциональности

В таблице 6 отображена характеристика использования значений частот.

Таблица 6 – Используемые значения частот

Значение	**Характеристика**
3	Используется персонажем несколько раз в неделю
2	Используется персонажем 1-2 в месяц
1	Используется персонажем 1-2 раз в квартал

В таблице 7 изображена функциональность приложения и частота использования ключевым персонажем Ольгой.

Таблица 7 – Частота востребования функциональности

Требование	Итого
Возможность показывать активные задачи	3
Возможность подробно изучить новые задачи	3
Возможность отображения выполненных задач	2
Возможность поиска задач	3

2.4.10 Информационная структура приложения, общая концепция

Исходя из требований к проектируемому приложению, предлагает следующая информационная структура:

– таблица задач (список активных задач в виде карточек задач, кнопка разместить задачу, кнопка подробнее о задаче, кнопка редактирования карточки задачи, кнопка удаления задачи, статус задачи, выход);

– экран разместить задачу (поле для ввода: Заголовка задачи, Описания задачи, Срок исполнения задачи, Выбор приоритета задачи, Выбор исполнителя задачи, кнопка разместить задачу, выход);

– экран подробнее о задаче (форма карточки с задачей, поле для ввода комментария, кнопка написать комментарий, кнопка навигации к таблице задач, выход);

– экран редактирования задачи (поле для ввода: Заголовка задачи, Описания задачи, Срок исполнения задачи, Выбор приоритета задачи, Выбор статуса задачи, Выбор исполнителя задачи, кнопка обновить задачу, выход);

– экран поиска (поле для ввода искомого значения, кнопки навигации таблица задач, выход).

Общая концепция приложения представлена на рисунке 16.

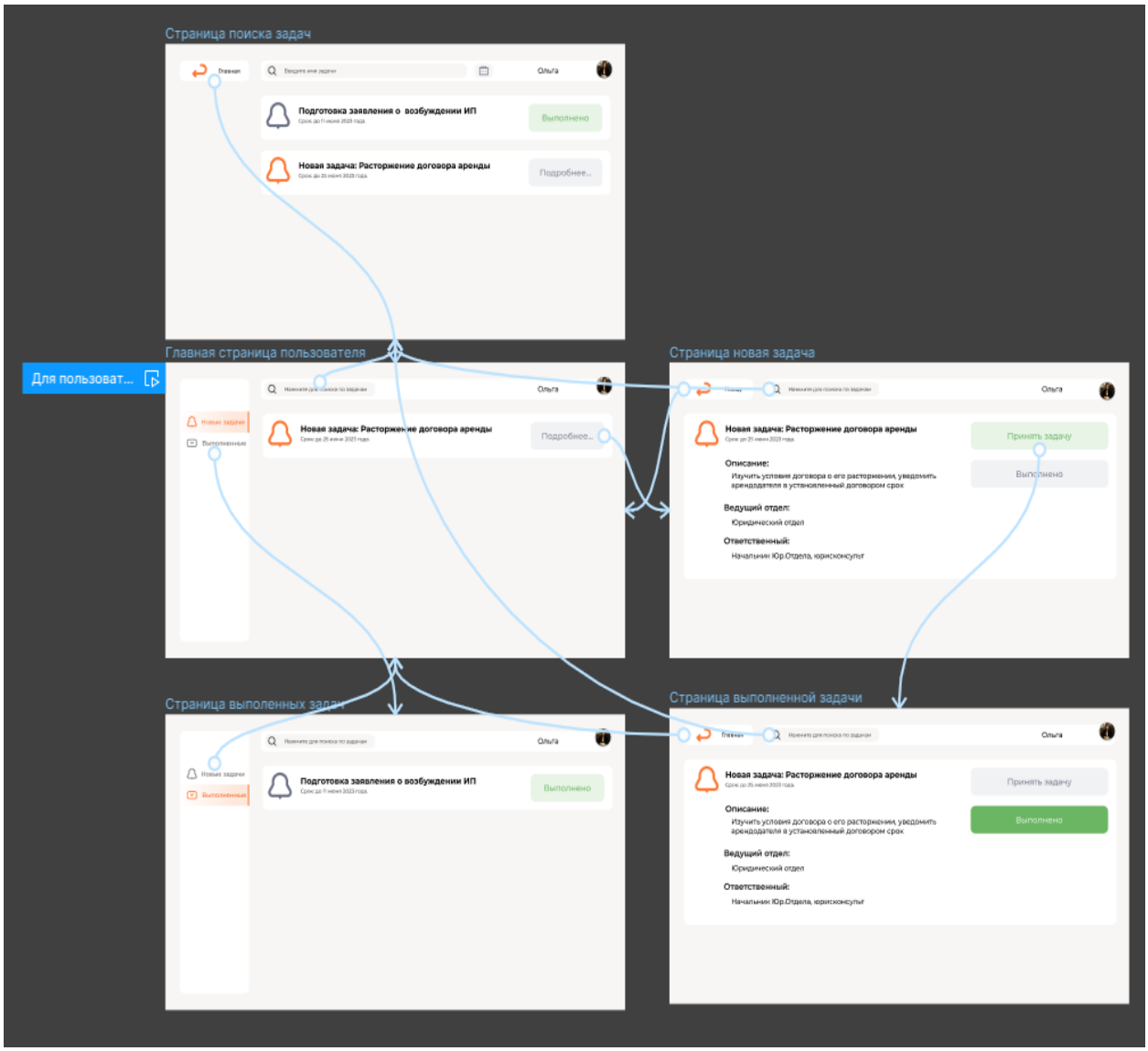


Рисунок – 16 Общая концепция приложения

2.4.11 Реализация требований пользователя

2.4.11.1. Наглядное представление активных задач

Выполнение требования наглядного представления активных задач представлено на рисунке 17. При отображении главной страницы пользователь видит все активные задачи.

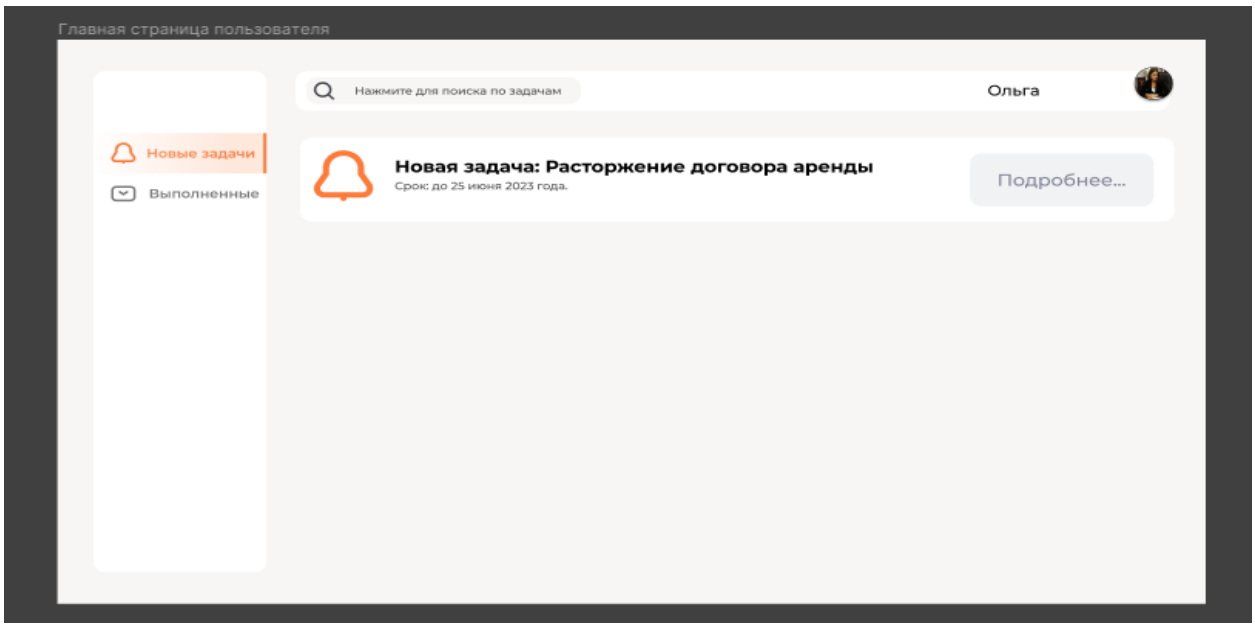


Рисунок – 17 Экран активных задач пользователя

2.4.11.2. Наглядное представление описание задачи

Выполнение требования подробного изучения поставленной задачи представлено на рисунке 18. При отображении страницы новая задач пользователь видит ее описание.

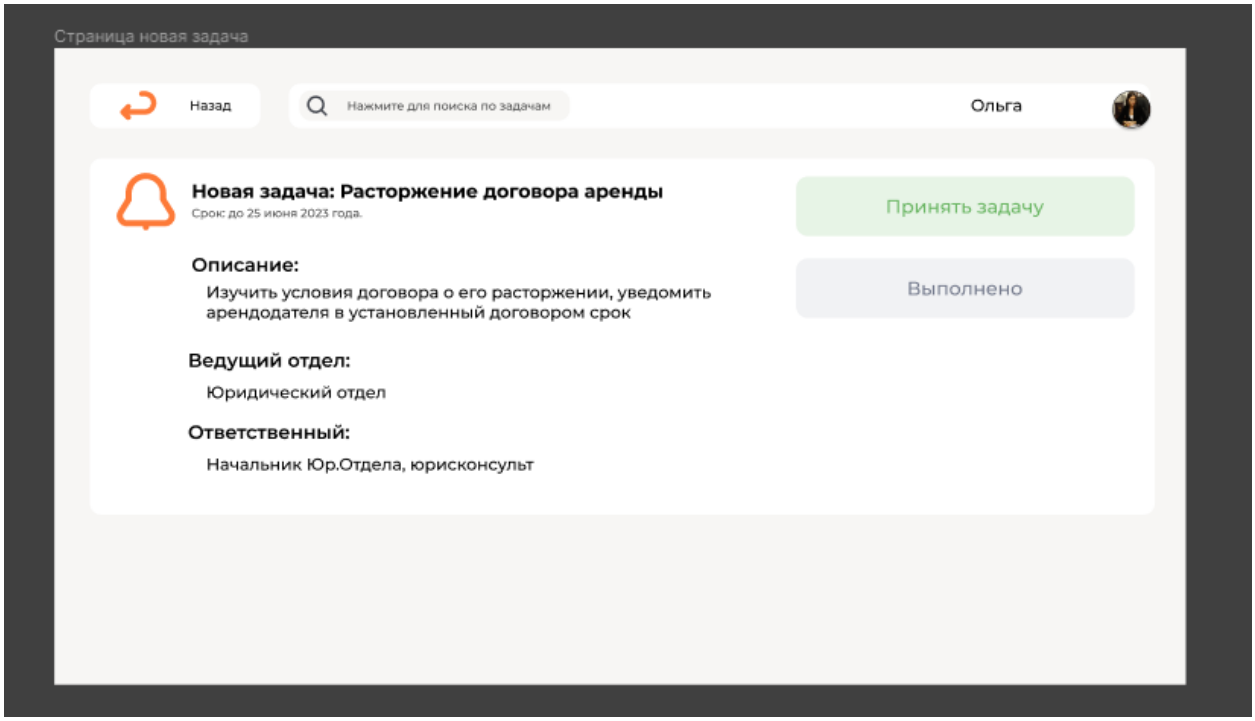


Рисунок – 18 Экран описания задачи пользователя

2.4.11.3. Наглядное представление выполненных задач

Выполнение требования отображения выполненных задач представлено на рисунке 19. При отображении страницы выполненных задач пользователь видит, выполненные ранее задачи.

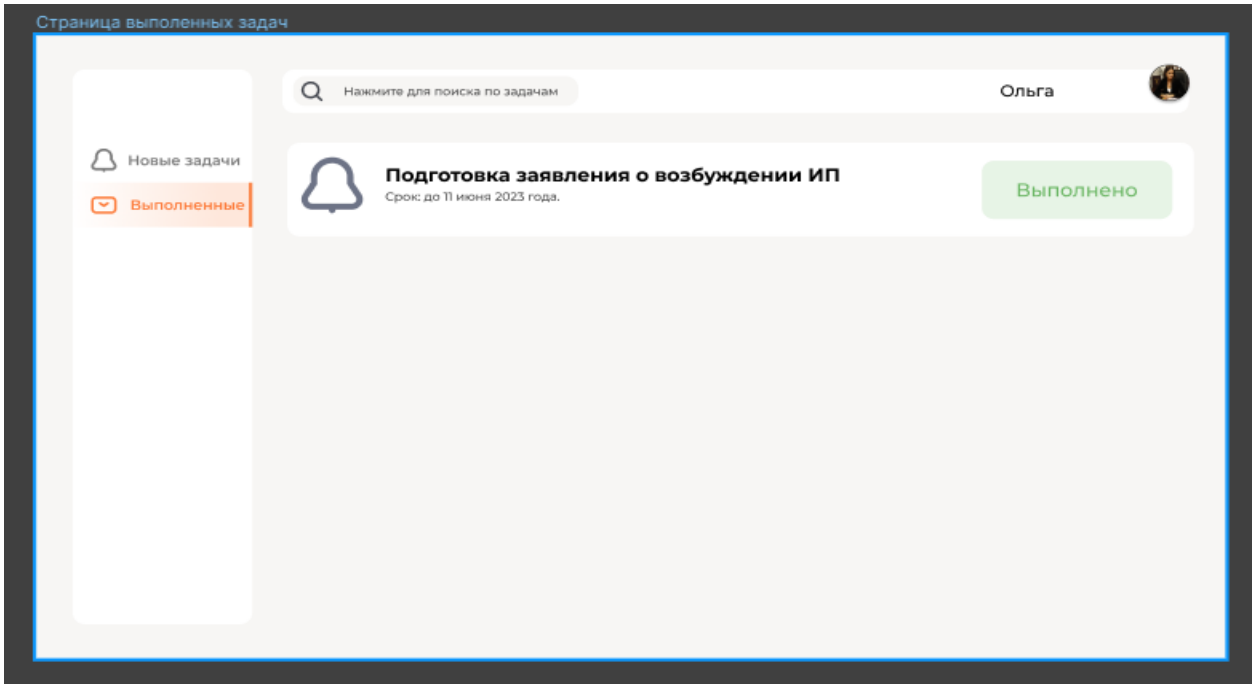


Рисунок – 19 Экран выполненных задач пользователя

2.4.11.4. Наглядное представление поиска задач.

Выполнение требования отображения поиска задач представлено на рисунке 20. При отображении страницы поиска задач пользователь видит все задачи.

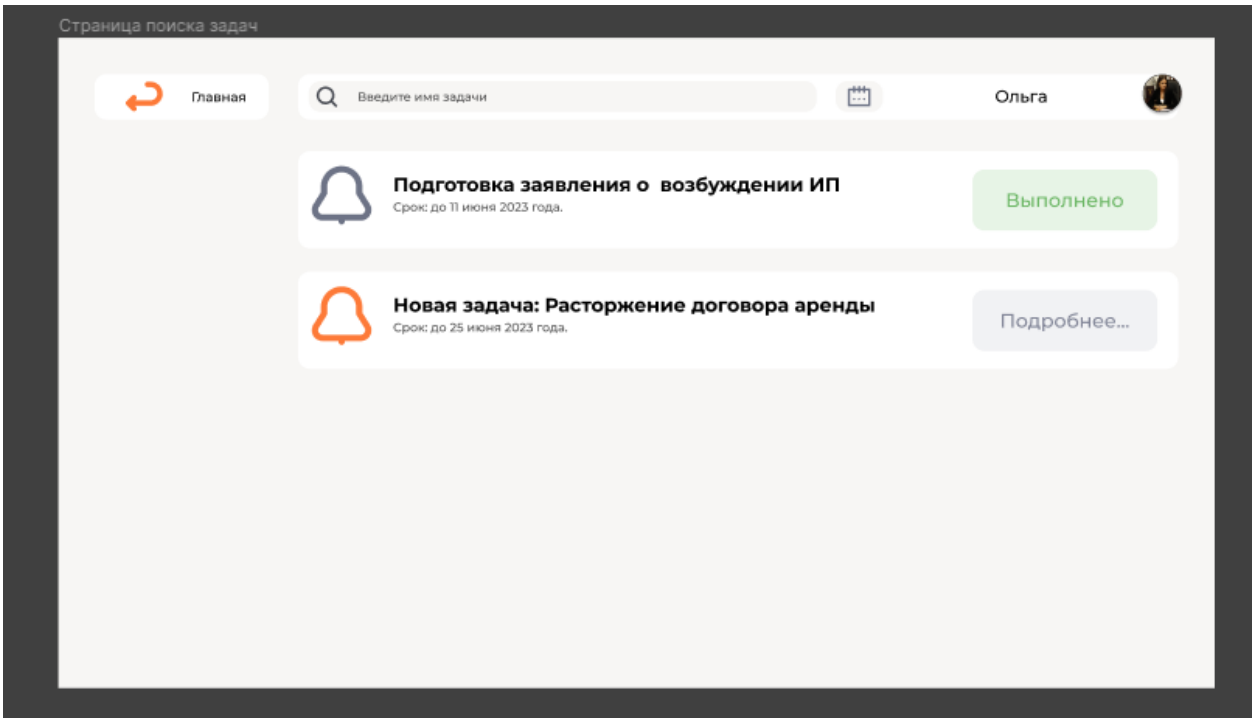


Рисунок – 20 Экран поиска задач пользователя

3 Описание системы

3.1 Архитектура разрабатываемой системы

Архитектура монолитного приложения представляет собой подход к разработке программного обеспечения, при котором все компоненты приложения находятся в одном целом. В отличие от других архитектурных подходов, таких как микросервисная архитектура, монолитная архитектура подразумевает использование одной основной кодовой базы для всего приложения [23].

В данной системе был выбран модульный вариант монолитной архитектуры такой подход к проектированию приложений, позволяет, с одной стороны, отложить во времени операционную сложность использования микросервисов, а с другой –избежать превращения монолитной системы в большой комок грязи.

Архитектура разрабатываемой системы была выполнена на языке ArchiMate [24, 25] и представлена на Рисунке 21.

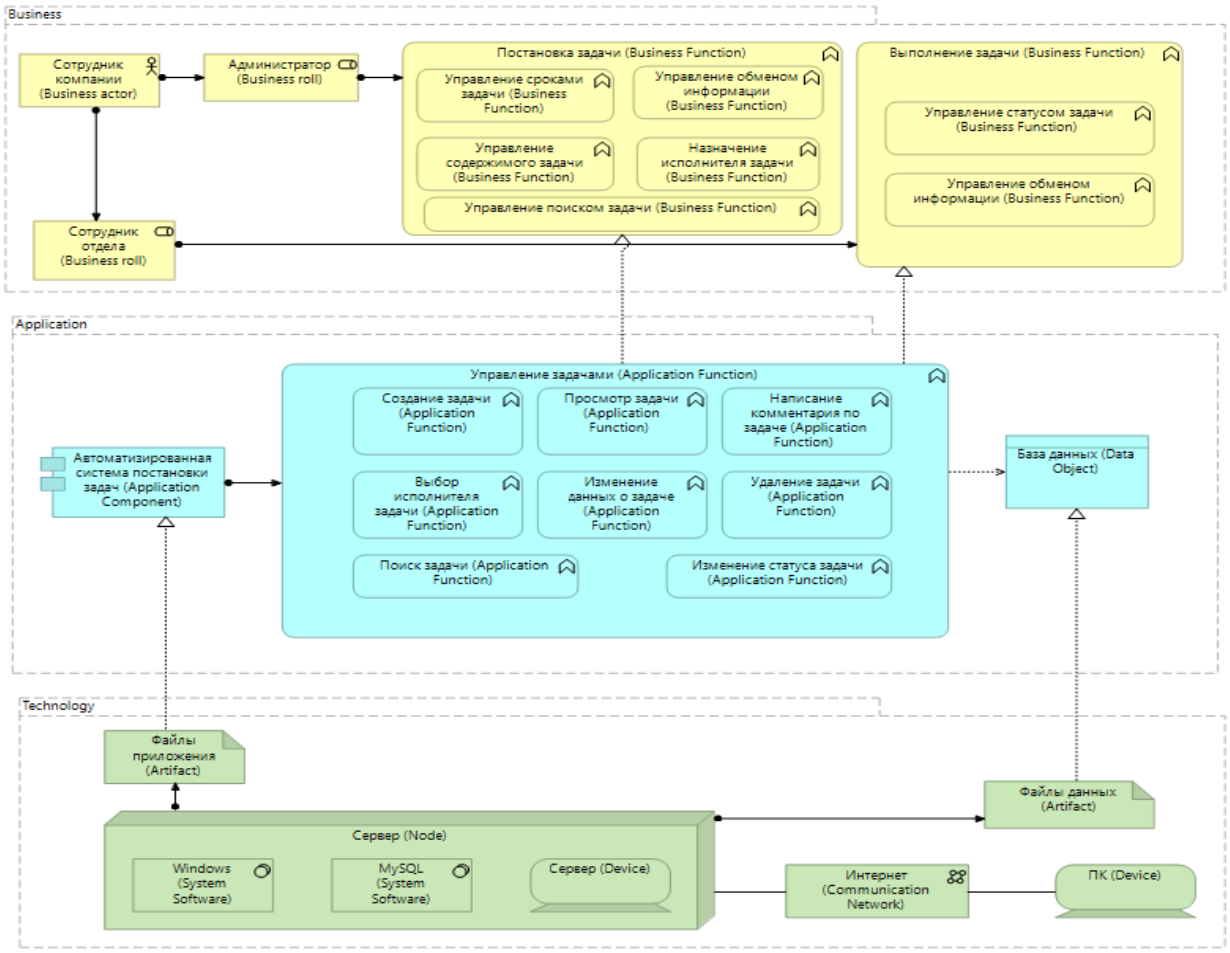


Рисунок – 21 Архитектура разрабатываемой системы

Описание результата

Результатом работы является создание системной архитектуры автоматизированной системы постановки задач, что изображено на рисунке 21.

Technology Layer состоит из таких элементов как: Node, System Software, Device, Artifact, Communication Network.

Application Layer включает в себя: Application Component, Application Interface, Application Service, Application Function, Data Object.

Business Layer состоит из: Business Actor, Business Role Business Function, Business Object.

3.1.1 Аннотация Business Layer

Сотрудник компании (Business Actor) – неавторизованный сотрудник компании.

Администратор (Business Role) – начальник отдела имеет возможность просмотра текущих задач пользователей, создание задачи, назначение исполнителя задачи, управление сроками задачи, редактирование задачи, удаления задачи, поиска задач, также оставлять комментарий по текущим задачам пользователям.

Сотрудник отдела (Business Role) – рядовой сотрудник отдела финансовой компании, имеет возможность просмотра текущей задачи, изменять статус задачи, а также оставлять комментарий по текущей задаче.

Постановка задачи (Business Function) – Процесс, выполняемый непосредственно начальниками отделов, направленный на достижение определенного результата в ограниченных временных рамках и с ограниченными ресурсами.

Выполнение задачи (Business Function) – Процесс, выполняемый непосредственно сотрудниками отделов, направленный на достижение определенного результата в ограниченных временных рамках и с ограниченными ресурсами.

Управление сроками задач (Business Function) – Процесс, включающий в себя разработку плана работ, формирование календарного графика, выполнения его пунктов и мониторинг его соблюдения.

Управление содержанием задачи (Business Function) – Процесс, определяющая работы, которые должны быть выполнены, чтобы создать и передать заказчику продукт проекта с необходимыми свойствами и функциональностью.

Назначения исполнителя задачи (Business Function) – Процесс обеспечения распределения задач согласно их специфическим требованиям.

Управление обменом информацией (Business Function) – Обеспечение своевременного сбора, генерации, распределения и хранения необходимой информации о задаче. Определение информационных и коммуникационных нужд для сотрудников компании. Своевременное предоставление необходимой информации сотрудникам компании.

Управление поиском задач (Business Function) – Обеспечение поиска задач по заданным критериям.

3.1.2 Аннотация Application Layer

Автоматизированная система постановки задач (Application Component) – приложение таск-менеджер, предназначенное для начальников отделов.

Управление задачами (Application Function) – функционал таск-менеджера, позволяющий начальнику отдела управлять задачами.

Создание задачи (Application Function) – подфункционал, позволяющий создать задачу при помощи заполнения полей формы.

Просмотр задачи (Application Function) – подфункционал, позволяющий осуществлять просмотр поставленной задачи.

Написание комментария по задаче (Application Function) – подфункционал, позволяющий создавать комментарии по задачам при помощи заполнения полей формы.

Выбор исполнителя задачи (Application Function) – подфункционал, позволяющий выбрать исполнителя задачи согласно ее специфике.

Изменение данных о задаче (Application Function) – подфункционал, позволяющий вносить изменения в уже поставленные задачи.

Удаление задачи (Application Function) – подфункционал, позволяющий удалять выполненные/активные задачи из базы данных.

Поиск задачи (Application Function) – подфункционал, позволяющий искать ранее поставленные задачи по заданным критериям.

Изменения статуса задачи (Application Function) – подфункционал, позволяющий исполнителю задачи уведомлять о ее выполнении.

База данных (Data Object) – данные, полученные после заполнения формы постановки задачи.

3.1.3 Аннотация Technology Layer

Файлы приложения (Artifact) - разработанное программное обеспечение.

Сервер (Node) – вычислительный ресурс.

Windows (System Software) – операционная система, на основе которой разрабатывается приложение.

MySQL –свободная объектно-реляционная система управления базами данных, осуществляющая корректное распределение хранения данных на сервере, создание, редактирование и удаление записей, находящихся в базе, идентификацию и обработку отдельных данных, быстрый поиск нужной информации при помощи языка SQL.

Server (Device) – выделенный компьютер для выполнения сервисного программного обеспечения.

Internet (Communication Network) – сеть Интернет.

ПК (Device) – устройство пользователя.

Файлы данных (Artifact) – данные, полученные с сервера.

3.1.4 Построение диаграммы блоков

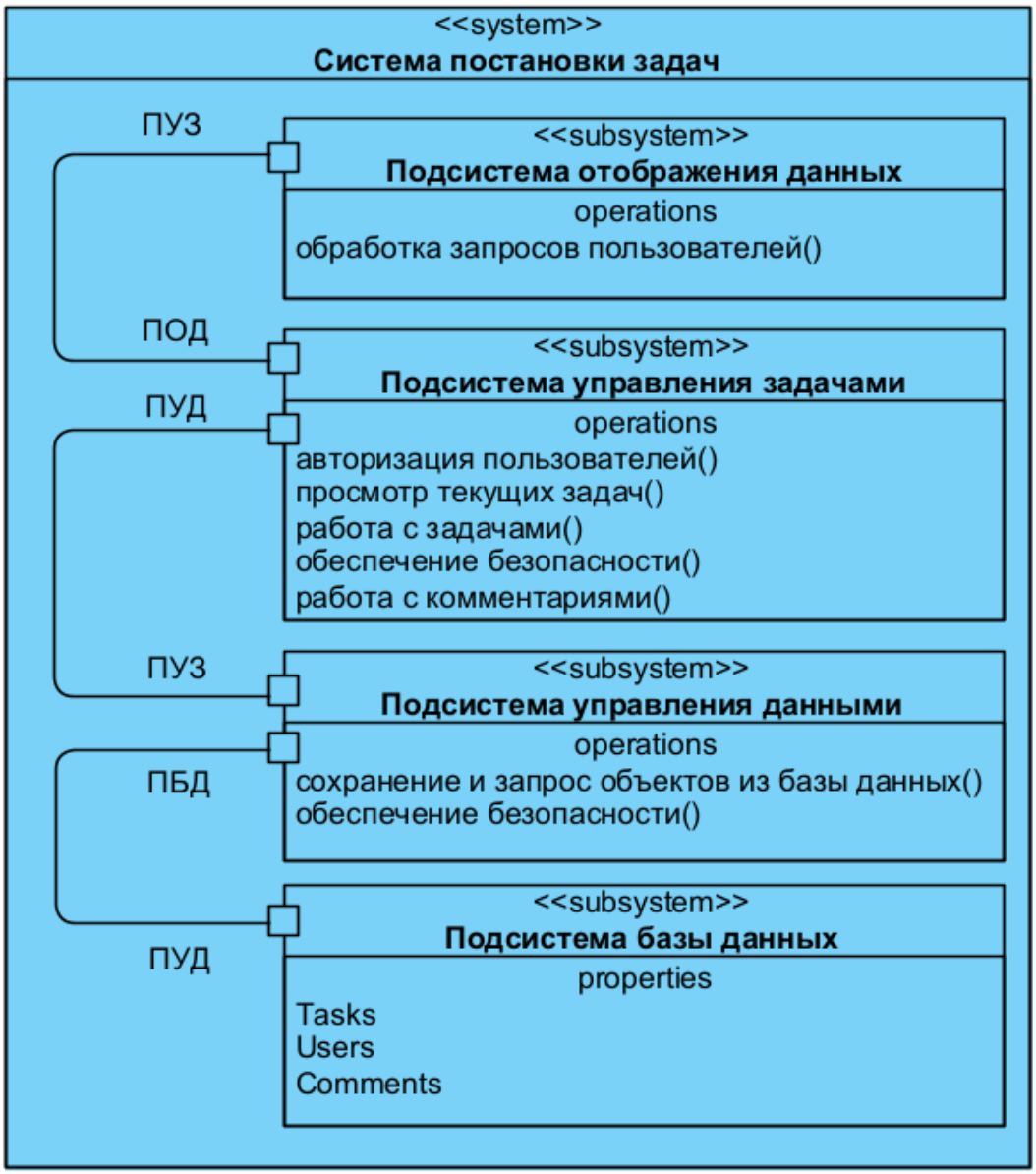


Диаграмма блоков представлена на рисунке 22.

Рисунок 22 – Диаграмма блоков

Разрабатываемая система строится по монолитной архитектуре.

ПУЗ (Порт управления задачами):

– Этот порт обеспечивает взаимодействие между подсистемой управления задачами (ПУД) и внешними компонентами или другими подсистемами.

– Отвечает за авторизацию пользователей, просмотр текущих задач, добавле-

ние и изменение задач, а также обработку комментариев.

– Основное назначение –передача команд от пользователя к системе.

ПОД (Порт отображения данных):

– Служит для обмена данными между подсистемой отображения данных и другими модулями системы.

– Обеспечивает доступ к информации, которая отображается пользователю, включая данные о задачах и комментариях.

– Отвечает за обратную связь от подсистемы управления данными (ПУД) к пользовательскому интерфейсу.

ПБД (Порт базы данных):

– Используется подсистемой управления данными для взаимодействия с базой данных.

– Выполняет функции сохранения, запроса и обновления объектов, таких как задачи, пользователи и комментарии.

– Обеспечивает безопасность данных при их сохранении и извлечении.

ПУД (Порт управления данными):

– Связывает подсистему управления данными с внешними подсистемами.

– Отвечает за централизованное управление данными, включая выполнение бизнес-логики и обеспечение безопасности.

3.1.4.1 Подсистема отображения данных в режиме реального времени. Подсистема отображения данных может показывать данные в том виде, в котором они поступают от других компонентов системы, к примеру в различных форматах, формы для ввода текста, кнопки для ввода информации, карточки задач, и т.д.;

3.1.4.2 Подсистема управления задачами (Task Management Subsystem) –это компонент системы, который отвечает за управление задачами и их выполнение. Она может выполнять следующие функции:

– Авторизация пользователей. Подсистема управления задачами проводит аутентификацию и авторизацию пользователей.

– Просмотр текущих задач. Подсистема управления задачами может выводить на экран текущие задачи пользователей.

– Работа с задачами. Подсистема управления задачами предоставляет пользователю статуса admin выполнять CRUD операции такие как (создание, просмотр, изменение, удаление), пользователю статуса employee выполнять операции просмотра задач, изменение статуса задачи.

– Обеспечение безопасности [25]. Подсистема управления задачами может обеспечивать безопасность системы, например, путем контроля доступа к данным и защиты от несанкционированного доступа к задачам на основе JSON Web Token (JWT).

– Работа с комментариями. Подсистема управления задачами может обеспечивать написания комментариев по задачам авторизованным пользователям.

3.1.4.3 Подсистема управления данными (Data Management Subsystem) –это компонент системы, который отвечает за управление данными и их хранение. Она может выполнять следующие функции:

– Сбор, хранение и обработка данных. Подсистема управления данными может собирать, хранить и обрабатывать данные, вводимые пользователями

– Обеспечение безопасности. Подсистема управления данными может обеспечивать безопасность системы, например, путем шифрования данных и контроля доступа к ним.

3.1.4.4 Подсистема базы данных (Data Base Subsystem) –это компонент системы, который отвечает за хранение данных в базе данных. Она хранит данные следующих полей: Task (все задачи пользователей), Users (все зарегистрированные пользователи в системе), Comments (все комментарии пользователей по задачам).

3.2 Структура программного обеспечения

3.2.1 Диаграмма вариантов использования

Диаграмма вариантов использования [27] для системы постановки задачами представлена на рисунке 23.

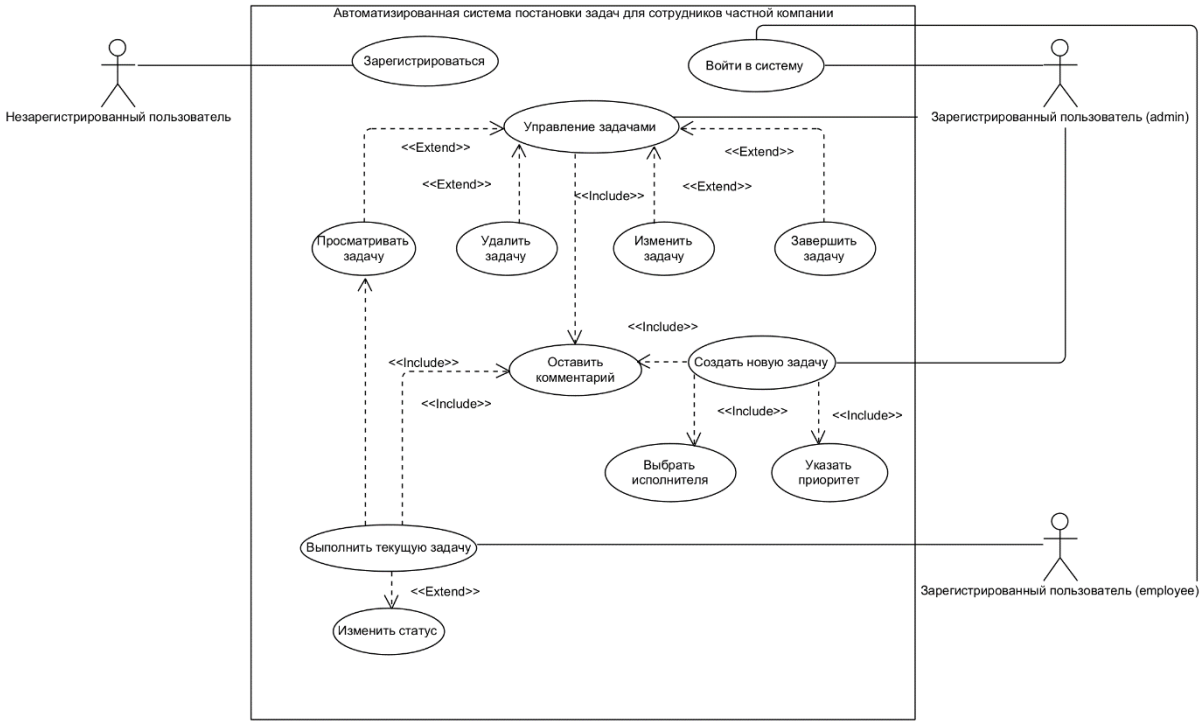


Рисунок 23 – Диаграмма вариантов использования

Действующими лицами являются Неавторизированный пользователь, Авторизированный пользователь статуса admin, Авторизованный пользователь статуса empoloyee, Приложение. Неавторизированный пользователь имеет возможность зарегистрироваться в системе. Авторизированный пользователь статуса admin может войти в систему, посмотреть список задач, создать новую задачу, выбрать исполнителя задачи, посмотреть задачу, изменить существующую задачу, удалить задачу, оставить комментарий по задаче. Авторизированный пользователь статуса employee может войти в систему, посмотреть личный список задач, поменять статус задачи, оставить комментарий по задаче.

3.2.2 Диаграммы последовательности

Диаграмма последовательности [28] представлена на рисунке 24. После того, как пользователь статуса admin вошел в систему, он может увидеть актуальный список всех задач пользователей. Далее есть шесть вариантов развития событий, в зависимости от того, что хочет сделать с задачами пользователь статуса admin: создать новую, просмотреть задачи пользователей, изменить существующую, удалить задачу, найти задачу по названию, написать комментарий по задаче. Для этого на диаграмму последовательности добавляем альтернативный комбинированный фрагмент взаимодействия.

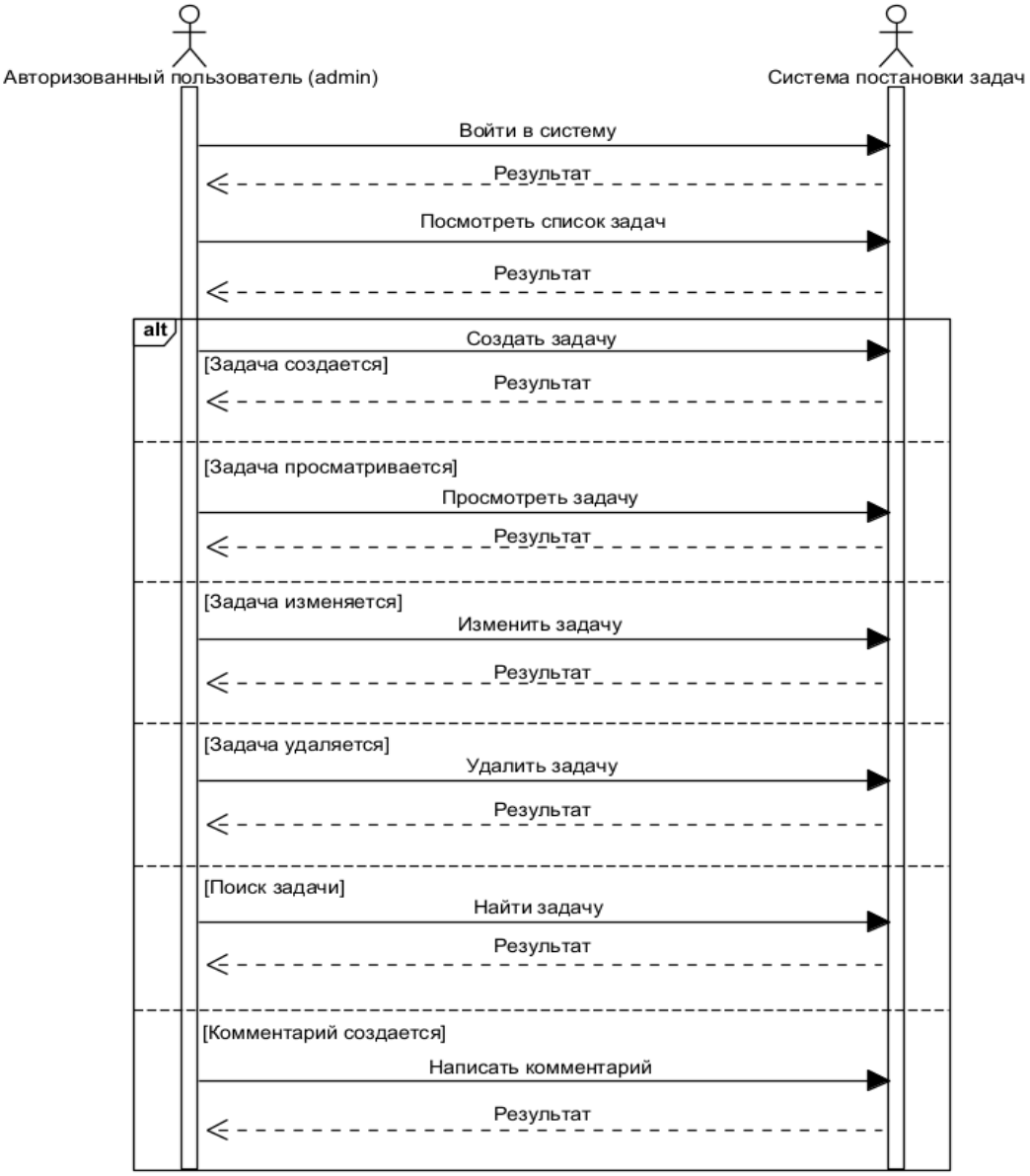


Рисунок 24 – Диаграмма последовательности для авторизованного пользователя admin

Диаграмма последовательности представлена на рисунке 25 После того, как пользователь статуса employee вошел в систему, он может увидеть свои текущие задачи. Далее есть три варианта развития событий, в зависимости от того, что хочет сделать с задачами пользователь статуса employee: просмотреть текущую задачу, изменить статус задачи, написать комментарий по задаче. Для этого на диаграмму последовательности добавляем альтернативный комбинированный фрагмент взаимодействия.

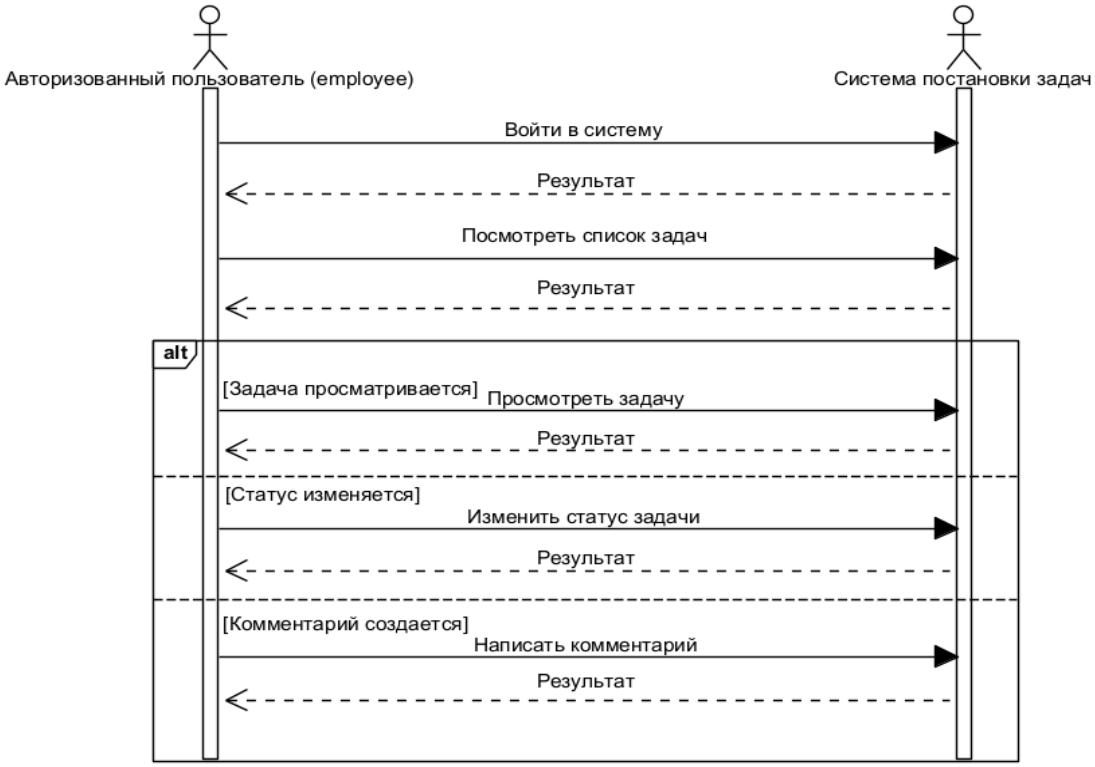
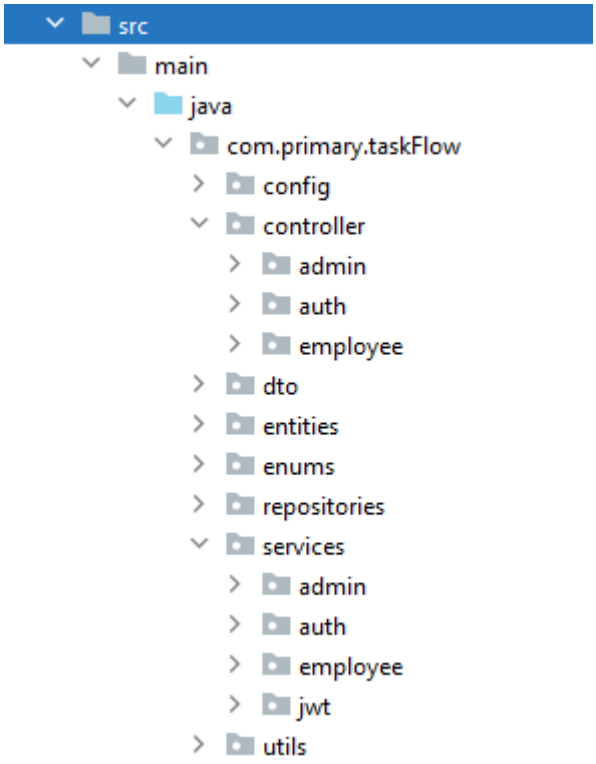


Рисунок 25 – Диаграмма последовательности для авторизованного пользователя employee

3.2.3 Диаграммы классов

Для создания правильных диаграмм классов backend сервиса системы нужно для начала изучить структуру пакетов [29]. На рисунке 26 представлена структура пакетов системы с



вложенными пакетами классов.

Рисунок 26 – Структура пакетов backend-сервиса

Далее просмотрим все пакеты и вложенных пакеты для создания диаграммы классов и описание сущностей входящих в них методом обратного проектирования в среде разработки intellij idea 2024.1

3.2.3.1 Пакет config (настройка авторизации и аутентификации).

На рисунке 27 представлена диаграмма классов пакета config.

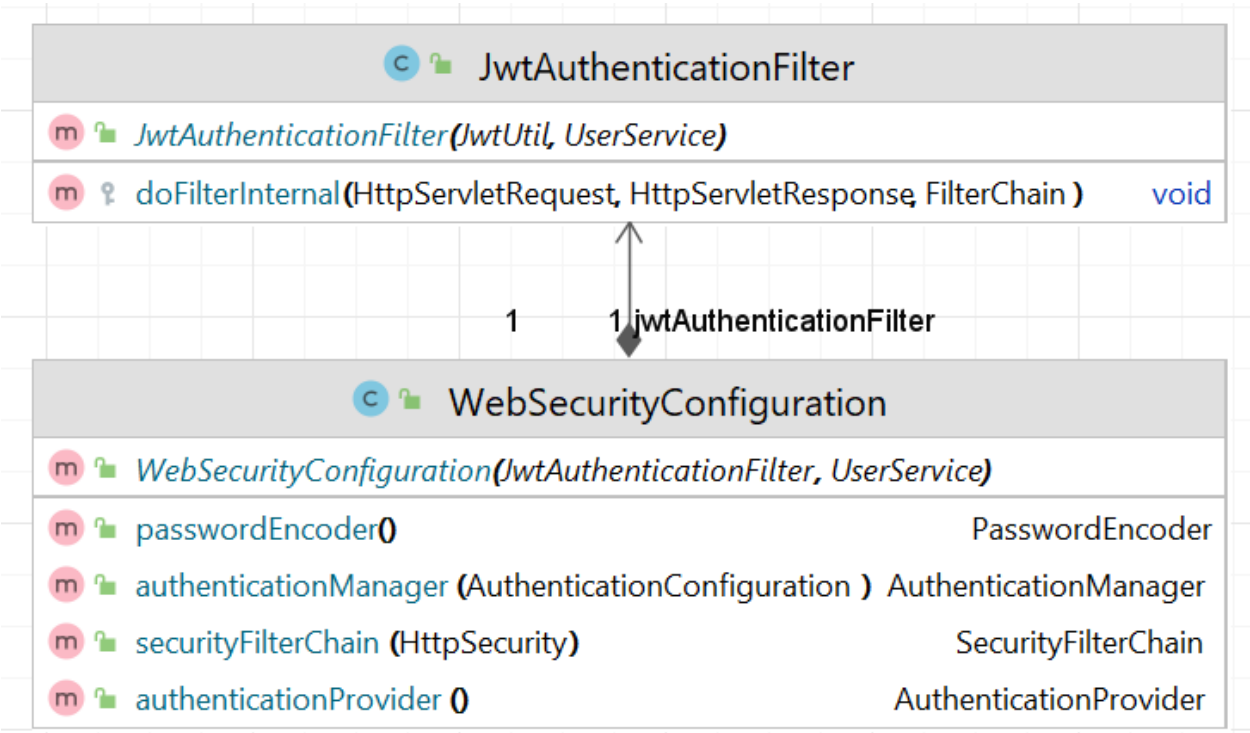


Рисунок 27 – Диаграмма классов пакета config

`WebSecurityConfiguration`:

Этот класс настраивает безопасность приложения, включая фильтры для JWT и конфигурации авторизации.

Методы:

`securityFilterChain(HttpSecurity)` –конфигурация цепочки безопасности;

`authenticationManager(AuthenticationConfiguration)` –настройка менеджера аутентификации для работы с JWT.

`passwordEncoder()` –хранение паролей для проверки;

`authenticationProvider()` –настройка провайдера аутентификации.

3.2.3.2. Пакет controller (управление запросами):

На рисунке 28 представлен пакет controller созданный для управления основными запросами в системе, он содержит в себе такие пакеты как AdminController, AuthController, EmployeeController.

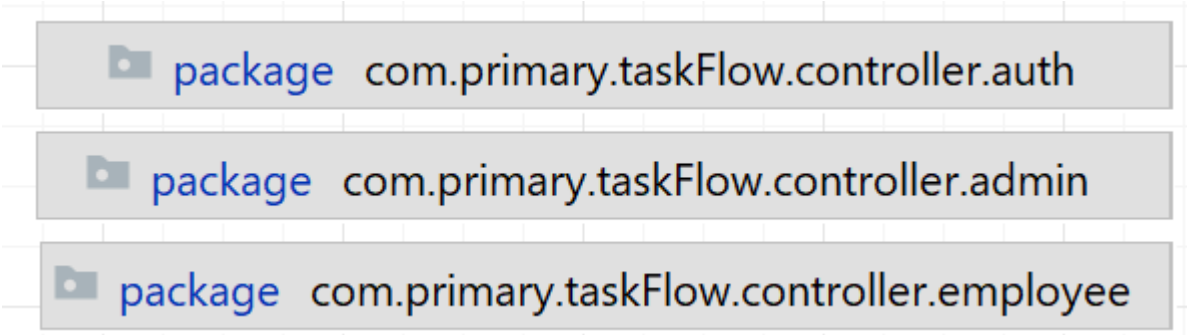


Рисунок 28 – Пакеты входящие в пакет controller

На рисунке 29 представлено описание класса AdminController.

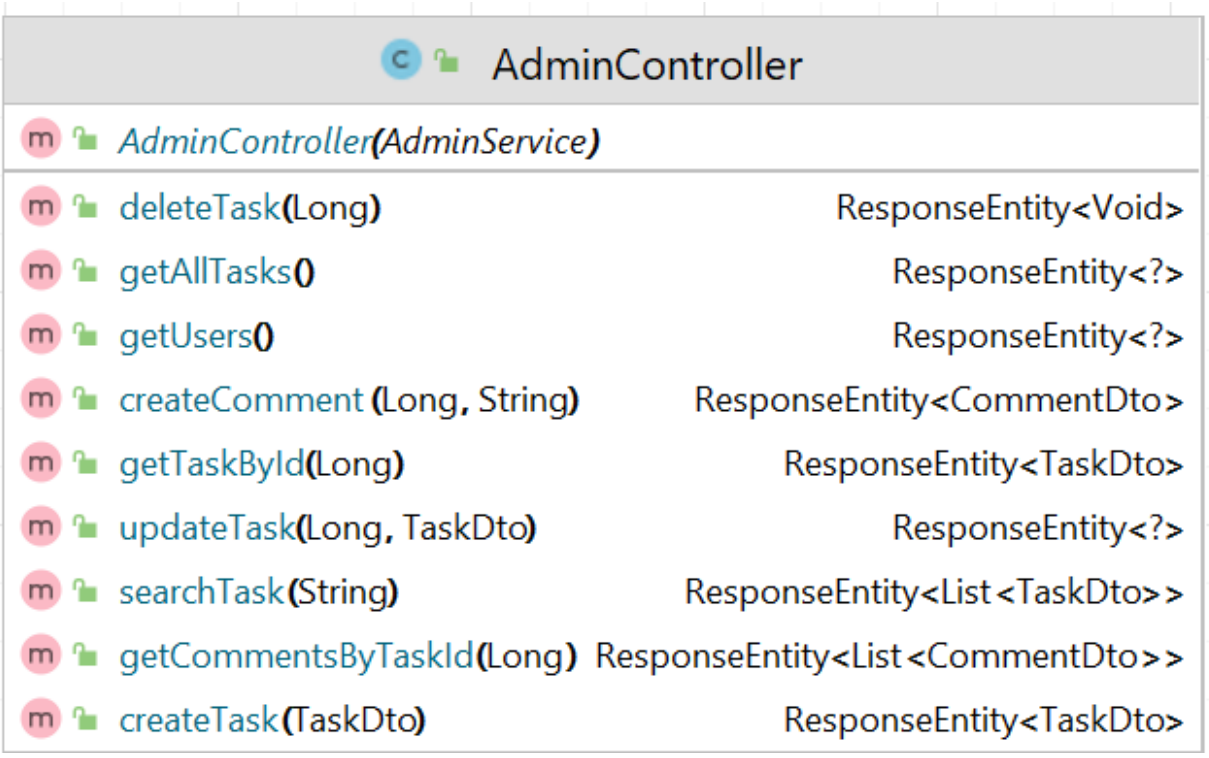


Рисунок 29 – Описание класса AdminController

Управляет задачами и комментариями. Предоставляет методы для CRUD операций с задачами.

Методы:

`getUsers()` – вывод активных пользователей системы;

`getCommentsByTaskid (Long)` –вывод комментариев по ID задачи;

`updateTask (Long)` –обновление задачи;

`searchTask(String)` –поиск задачи;

`createComment(Long, String)` –создание комментария к задаче.

`deleteTask(Long)` –удаление задачи;

`getTaskById(Long)` –получение задачи по ID;

`getAllTasks()` –получение всех задач в системе;

`createTask(TaskDto)` –создание задачи.

На рисунке 30 представлено описание класса AuthController.

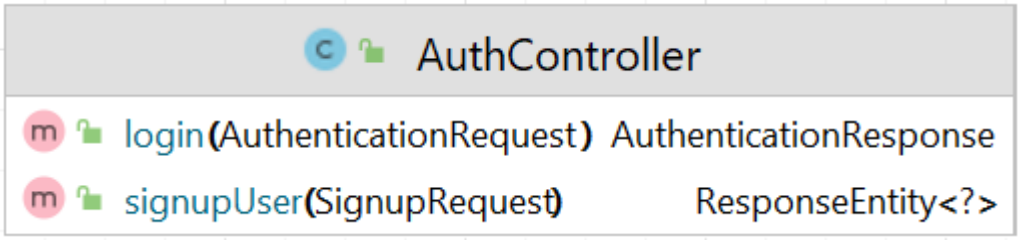


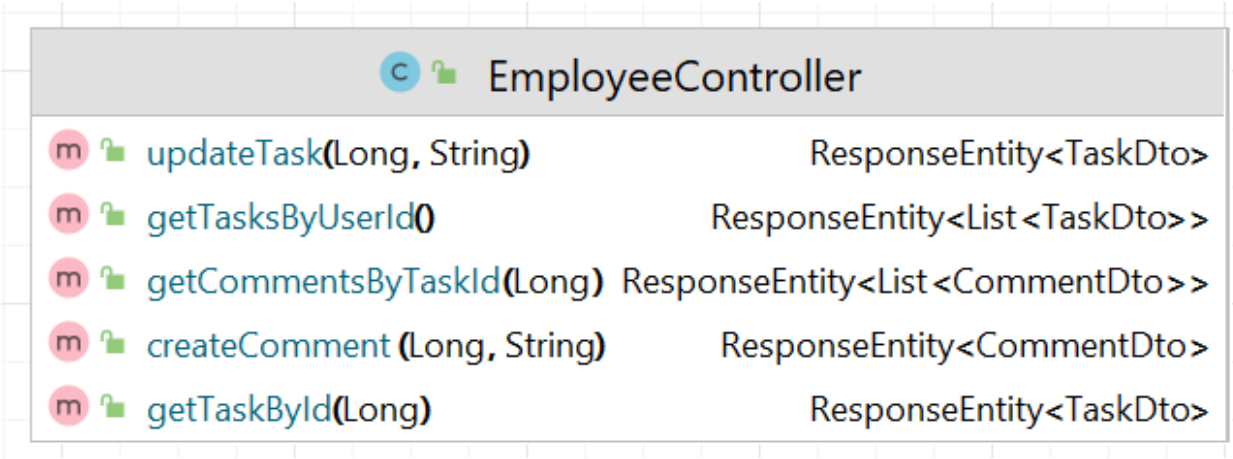
Рисунок 30 – Описание класса AuthController

Контроллер для управления аутентификацией и регистрацией пользователей.

Методы:

`login(AuthenticationRequest)` –авторизация и генерация JWT;

`signupUser(SignupRequest)` –регистрация пользователя.



На рисунке 31 представлено описание класса EmployeeController:

Рисунок 31 – Описание класса EmployeeController

Управляет задачами сотрудников и комментариями.

Методы:

`getTaskByUserId()`–получение задачи по ID пользователя;

`updateTask(Long)` –обновление задачи.

`getTaskById(Long)` –получение задачи по ID;

`createComment(Long, String)` –добавление комментария;

`getCommentsByTaskid(Long)` –получение комментариев по ID задачи.

3.2.3.3 Пакет dto (передача данных).

На рисунке 32 представлена диаграмма классов пакета dto. Пакет dto (Data Transfer Object) используется для передачи данных между слоями и содержит в себе такие классы как: TaskDto. CommentDto, UserDto, AuthenticationResponse, AuthenticationRequest, SignupRequest.



Рисунок 32 – Диаграмма классов пакета dto

Класс TaskDto:

DTO для передачи данных о задаче.

Методы:

getTaskStatus()/setTaskStatus(TaskStatus)

Получение/установка статуса задачи.

getId()/setId(Long)

Получение/установка идентификатора задачи.

getTitle()/setTitle(String)

Получение/установка заголовка задачи.

getDescription()/setDescription(String)

Получение/установка описания задачи.

getPriority()/setPriority(String)

Получение/установка приоритета задачи.

getDueDate()/setDueDate(Date)

Получение/установка крайнего срока выполнения задачи.

getEmployeeId()/setEmployeeId(Long)

Получение/установка идентификатора сотрудника, ответственного за задачу.

getEmployeeName()/setEmployeeName(String)

Получение/установка имени сотрудника.

equals(Object)/hashCode()/toString()

Переопределённые методы для корректного сравнения и представления.

Класс CommentDto:

DTO для передачи данных о комментариях.

Методы:

getId()/setId(Long)

Получение/установка идентификатора комментария.

getContent()/setContent(String)

Получение/установка содержимого комментария.

getCreatedAt()/setCreatedAt(Date)

Получение/установка даты создания комментария.

getTaskId()/setTaskId(Long)

Получение/установка идентификатора задачи, к которой привязан комментарий.

getUserId()/setUserId(Long)

Получение/установка идентификатора пользователя, оставившего комментарий.

getPostedBy()/setPostedBy(String)

Получение/установка имени автора комментария.

equals(Object)/hashCode()/toString()

Переопределённые методы.

Класс UserDto:

DTO для передачи данных о пользователях.

Методы:

getId()/setId(Long)

Получение/установка идентификатора пользователя.

getName()/setName(String)

Получение/установка имени пользователя.

getEmail()/setEmail(String)

Получение/установка электронной почты пользователя.

getPassword()/setPassword(String)

Получение/установка пароля (возможно, для создания пользователя или изменения данных).

getUserRole()/setUserRole(UserRole)

Получение/установка роли пользователя.

equals(Object)/hashCode()/toString()

Методы для сравнения и отображения.

Класс AuthenticationRequest:

DTO для обработки запросов на авторизацию.

Методы:

getEmail()/setEmail(String)

Получение/установка электронной почты для авторизации.

getPassword()/setPassword(String)

Получение/установка пароля.

equals(Object)/hashCode()/toString()

Методы для корректного сравнения и отображения.

Класс AuthenticationResponse:

DTO для ответа на авторизацию.

Методы:

getJwt()/setJwt(String)

Получение/установка JWT-токена.

getUserId()/setUserId(Long)

Получение/установка идентификатора пользователя.

getUserRole()/setUserRole(UserRole)

Получение/установка роли авторизованного пользователя.

equals(Object)/hashCode()/toString()

Методы для отображения данных.

Класс SignupRequest:

DTO для обработки запросов на регистрацию нового пользователя.

Методы:

getName()/setName(String)

Получение/установка имени.

getEmail()/setEmail(String)

Получение/установка электронной почты.

getPassword()/setPassword(String)

Получение/установка пароля.

equals(Object)/hashCode()/toString()

Методы для обработки данных.

3.2.3.4 Пакет entities (сущности).

На рисунке 33 представлена диаграмма классов пакета Entities в которой содержатся такие классы как: Task, User, Comment основные сущности в системе

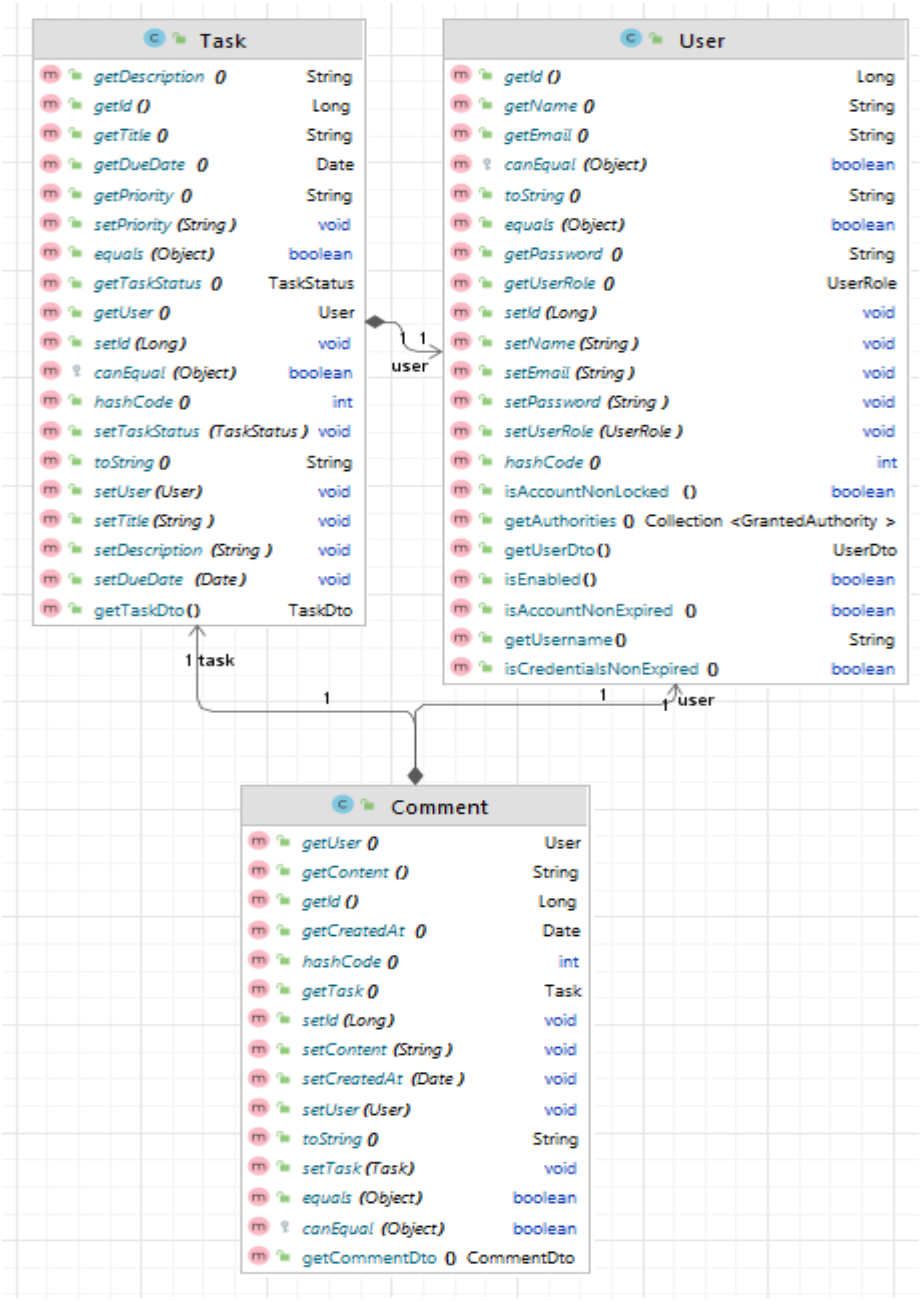


Рисунок 33 – Диаграмма классов пакета Entities

Класс Task:

Класс, представляющий сущность задачи в системе.

Поля:

`id (Long)`: Уникальный идентификатор задачи.

`title (String)`: Заголовок задачи.

`description (String)`: Описание задачи.

`priority (String)`: Приоритет задачи (например, высокий, средний, низкий).

`dueDate (Date)`: Срок завершения задачи.

`taskStatus (TaskStatus)`: Текущий статус задачи (например, "в процессе",

"выполнена").

`user (User)`: Ссылка на объект пользователя, которому назначена задача.

Методы:

Геттеры и сеттеры для всех полей.

`setTaskDto(TaskDto taskDto)`: Метод для обновления задачи на основе DTO.

`getTaskDto() (TaskDto)`: Возвращает объект DTO, содержащий данные задачи.

Переопределения методов equals(Object), hashCode() и toString().

Класс User:

Класс, представляющий сущность пользователя в системе.

Поля:

`id (Long)`: Уникальный идентификатор пользователя.

`email (String)`: Электронная почта пользователя.

`name (String)`: Имя пользователя.

`password (String)`: Хэшированный пароль пользователя.

`userRole (UserRole)`: Роль пользователя (например, "администратор", "исполнитель").

`isAccountNonExpired (boolean)`: Флаг, указывающий, не истек ли срок действия учетной записи.

`isAccountNonLocked (boolean)`: Флаг, указывающий, не заблокирована ли учетная запись.

`isCredentialsNonExpired (boolean)`: Флаг, указывающий, не истек ли срок действия учетных данных.

`isEnabled (boolean)`: Флаг, указывающий, активен ли пользователь.

`authorities (Collection<GrantedAuthority>)`: Коллекция, содержащая полномочия пользователя.

`tasks (List<Task>)`: Список задач, связанных с пользователем.

Методы:

Геттеры и сеттеры для всех полей.

`getUserDto() (UserDto)`: Метод для преобразования объекта пользователя в DTO.

Переопределения методов equals(Object), hashCode() и toString().

Класс Comment:

Класс, представляющий сущность комментария в системе.

Поля:

`id (Long)`: Уникальный идентификатор комментария.

`content (String)`: Содержимое комментария.

`createdAt (Date)`: Дата и время создания комментария.

`task (Task)`: Ссылка на задачу, к которой относится комментарий.

`user (User)`: Ссылка на пользователя, оставившего комментарий.

Методы:

Геттеры и сеттеры для всех полей.

`setCommentDto(CommentDto commentDto)`: Метод для обновления комментария на основе DTO.

`getCommentDto() (CommentDto)`: Метод, возвращающий объект DTO, содержащий данные комментария.

Переопределения методов `equals(Object)`, `hashCode()` и `toString()`.

3.2.3.5 Пакет enums (константы).

На рисунке 34 представлена диаграмма классов пакета enums в которой находятся такие классы как: UserRole, TaskStatus использующиеся в системе как основные константы.

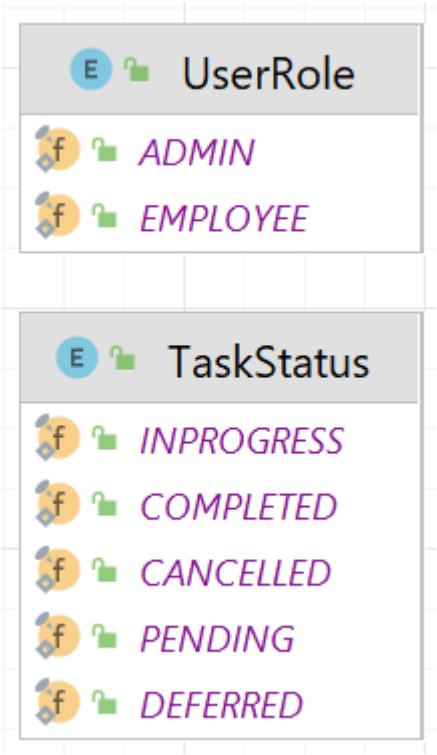


Рисунок 34 – Диаграмма классов пакета enums

Класс UserRole:

Перечисление ролей пользователей:

`ADMIN` – пользователь с расширенным набором функции;

`EMPLOYEE` – пользователь с ограниченным набором функции.

Класс TaskStatus:

Перечисление статусов задачи:

`PENDING` – задач в ожидании;

`INPROGRESS` – задача в процессе выполнения;

`COMPLETED` – задача выполнена;

`DEFERRED` – задача отложена;

`CANCELED` – задача отменена;

3.2.3.6 Пакет repositories (репозиторий интерфейсов).

На рисунке 35 представлена диаграмма классов пакета repositories, которая содержит в себе такие классы как CommentRepository, UserRepository, TaskRepository использующиеся в системе как промежуточные интерфейсы.

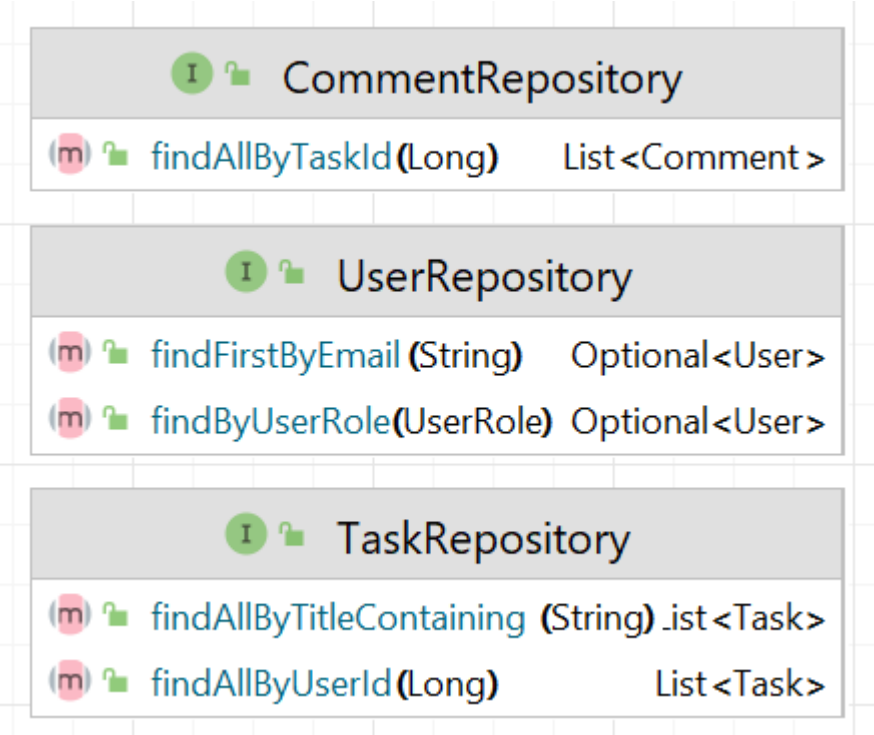


Рисунок 35 – Диаграмма классов пакета repositories

Интерфейс CommentRepository:

Репозиторий для работы с комментариями.

Методы:

`findByTaskId(Long)` –получение комментариев по идентификатору задачи.

Интерфейс UserRepository:

Репозиторий для работы с пользователями.

Методы:

`findFirstByEmail(String)` –поиск пользователя по email;

`findByUserRole(UserRole)` –поиск пользователя по роли.

Интерфейс TaskRepository:

Репозиторий для работы с задачами.

Методы:

`findAllByUser(User)` –поиск всех задач для пользователя;

`findAllByUser(User)` –поиск всех содержаний заголовков.

3.2.3.7 Пакет services (сервисы).

На рисунке 36 представлен пакет services созданный для реализации основных функций системе и содержит в себе такие пакеты как AdminController, AuthController, EmployeeController, jwt.

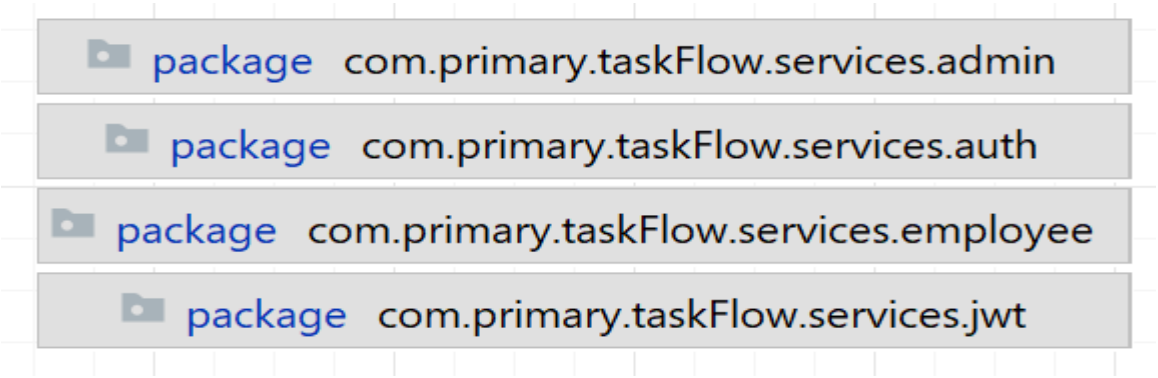


Рисунок 36 – Пакеты входящие в пакет services

На рисунке 37 представлена диаграмма классов AdminServices.

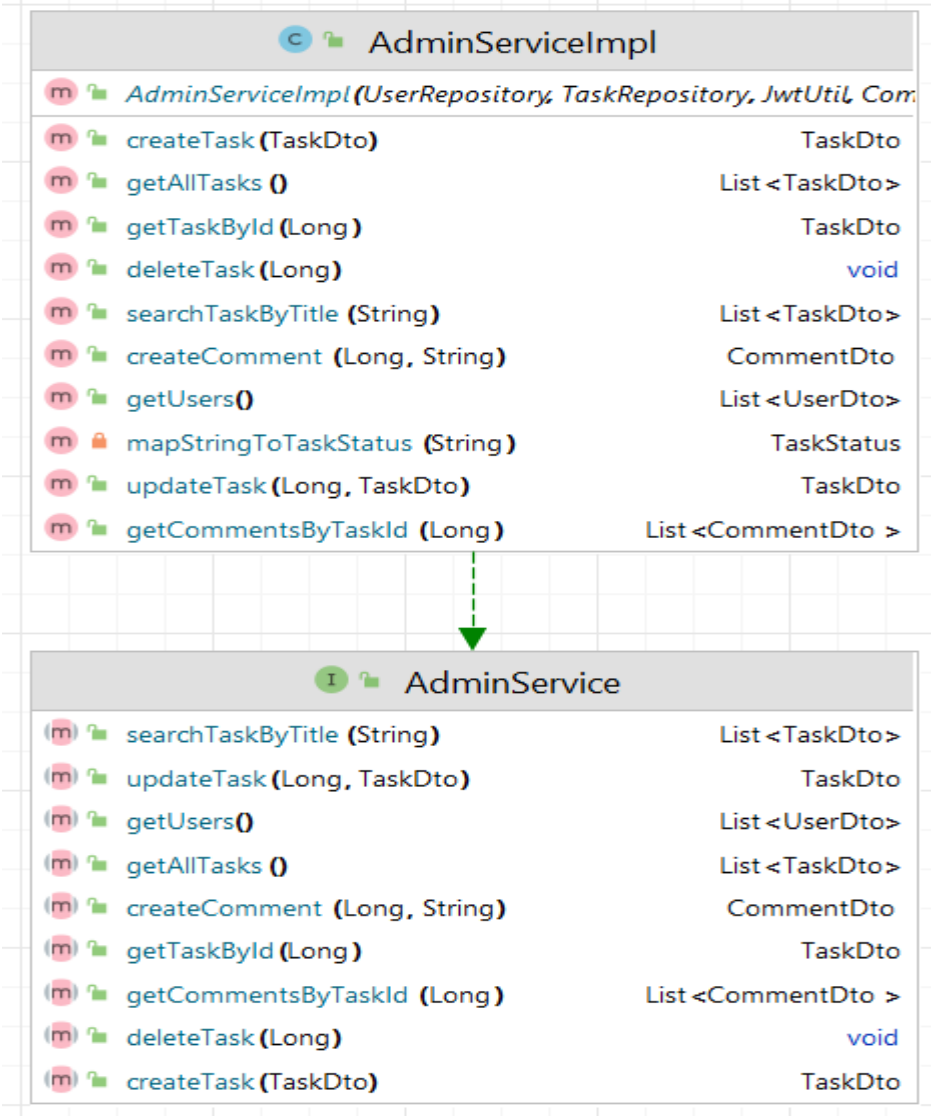


Рисунок 37 – Диаграмма классов пакета AdminServices

Интерфейс AdminService:

Интерфейс, задающий контракт для работы с административными операциями в системе.

Методы:

`createTask(TaskDto) (TaskDto)`

Создаёт новую задачу на основе объекта TaskDto.

Параметры: объект TaskDto.

Возвращает: созданную задачу.

`updateTask(Long id, TaskDto taskDto) (TaskDto)`

Обновляет существующую задачу по её идентификатору.

Параметры:

id –идентификатор задачи.

taskDto –объект с новыми данными для обновления.

Возвращает: обновлённую задачу.

`getAllTasks() (List<TaskDto>)`

Возвращает список всех задач в системе.

Возвращает: список объектов `TaskDto.`

`getTaskById(Long id) (TaskDto)`

Возвращает задачу по её идентификатору.

Параметры: id –идентификатор задачи.

Возвращает: объект TaskDto.

`getCommentsByTaskId(Long taskId) (List<CommentDto>)`

Возвращает список комментариев для указанной задачи.

Параметры: taskId –идентификатор задачи.

Возвращает: список объектов CommentDto.

`getUsers() (List<UserDto>)`

Возвращает список всех пользователей системы.

Возвращает: список объектов UserDto.

`createComment(Long taskId, String content) (CommentDto)`

Создаёт комментарий для определённой задачи.

Параметры:

taskId –идентификатор задачи.

content –текст комментария.

Возвращает: созданный объект CommentDto.

`searchTaskByTitle(String title) (List<TaskDto>)`

Выполняет поиск задач по заголовку.

Параметры: title –строка для поиска.

Возвращает: список задач, соответствующих запросу.

`deleteTask(Long id) (void)`

Удаляет задачу по её идентификатору.

Параметры: id –идентификатор задачи.

Класс AdminServiceImpl:

Класс, реализующий интерфейс AdminService. Он обеспечивает логику взаимодействия с репозиториями и компонентами системы.

Поля (конструктор с зависимостями):

UserRepository: Репозиторий для работы с пользователями.

TaskRepository: Репозиторий для работы с задачами.

JwtUtil: Утилита для работы с JWT-токенами.

CommentRepository: Репозиторий для работы с комментариями.

Методы:

Методы в этом классе соответствуют методам интерфейса AdminService, дополненные реализацией логики. Основные действия:

`mapStringToTaskStatus(String) (TaskStatus)`

Конвертирует строку в объект TaskStatus. Используется для валидации или обработки статусов задач.

`deleteTask(Long id)`

Удаляет задачу по её идентификатору. Метод включает в себя проверки и логику, связанную с удалением связанных данных.

`updateTask(Long id, TaskDto taskDto) (TaskDto)`

Реализует логику обновления задачи, включая проверки существования задачи и валидацию данных.

`getCommentsByTaskId(Long taskId) (List<CommentDto>)`

Получает комментарии, связанные с указанной задачей, из репозитория.

`searchTaskByTitle(String title) (List<TaskDto>)`

Выполняет поиск задач, которые соответствуют указанному заголовку. Может включать в себя использование фильтров в репозитории.

`createComment(Long taskId, String content) (CommentDto)`

Создаёт комментарий для определённой задачи, проверяя, существует ли указанная задача.

Остальные методы аналогичны описанию интерфейса AdminService.

На рисунке 38 представлена диаграмма классов пакета AuthServices.

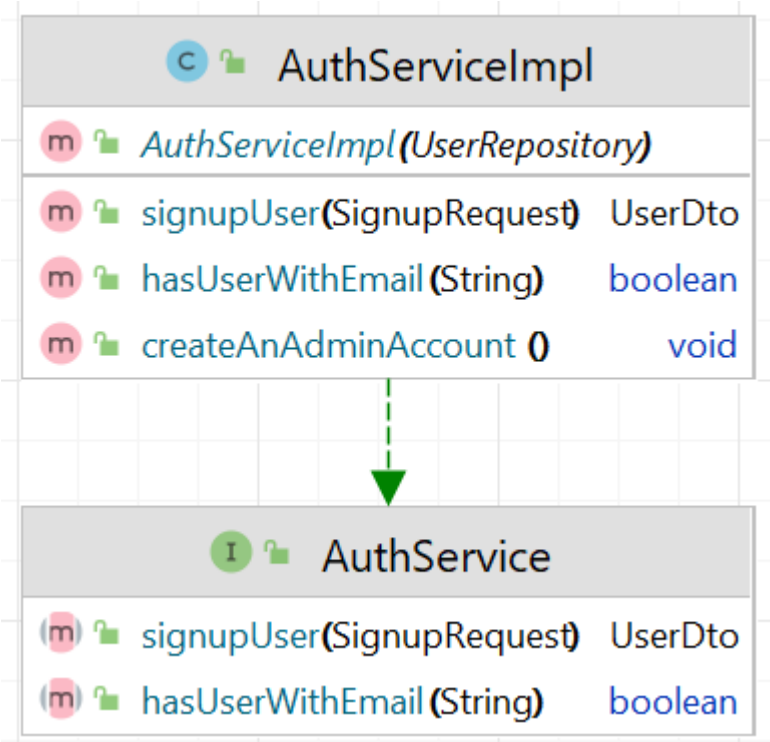


Рисунок 38 – Диаграмма классов пакета AuthServices

AuthServices: пакет, задающий контракт для работы с административными операциями в системе.

Класс AuthServiceImpl:

Методы:

`createAnAdminAccount() : void`

Создает учетную запись администратора. Логика связана с установлением прав администратора и сохранением данных в репозитории.

`signupUser(SignupRequest) : UserDto`

Реализует регистрацию нового пользователя. Метод принимает запрос с информацией о пользователе, выполняет проверки, преобразование данных и возвращает объект UserDto.

`hasUserWithEmail(String) : boolean`

Проверяет, существует ли пользователь с указанным email. Используется для валидации уникальности email.

Интерфейс AuthService:

Интерфейс, предоставляющий контракт для реализации функциональности аутентификации и регистрации.

Методы:

`signupUser(SignupRequest) : UserDto`

Метод для регистрации пользователя. Логика предоставляется классом, который

реализует этот интерфейс.

`hasUserWithEmail(String) : boolean`

Метод для проверки существования пользователя с заданным email.

На рисунке 39 представлена диаграмма классов пакета EmployeeSevices.

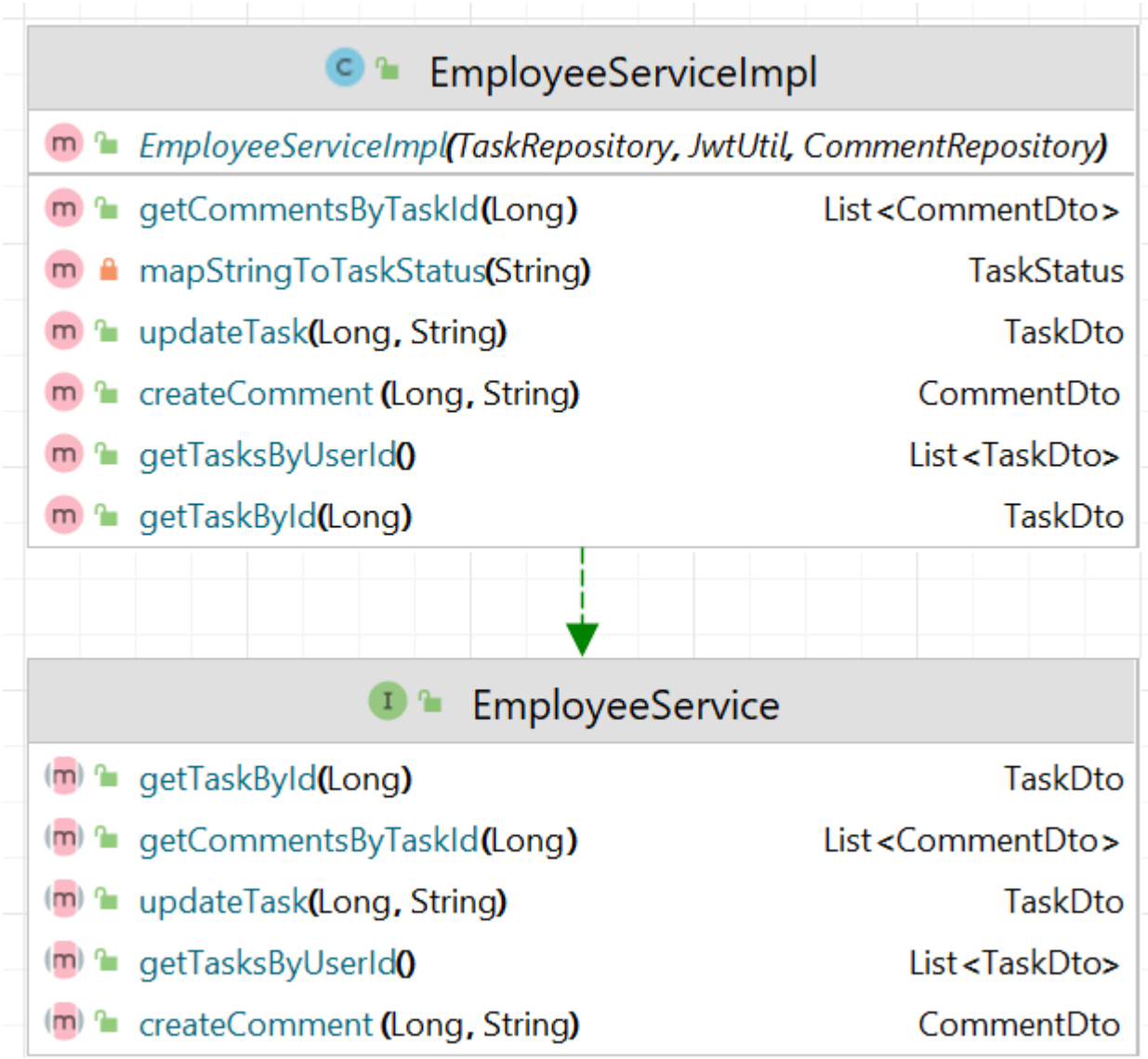


Рисунок 39 – Диаграмма классов пакета EmployeeServices

EmployeeService, пакет предназначен для обработки операций, связанных с задачами и комментариями.

Класс EmployeeServiceImpl:

Методы:

`getCommentsByTaskId(Long) : List<CommentDto>`

Возвращает список комментариев, связанных с конкретной задачей.

`getTaskById(Long) : TaskDto`

Получает задачу по её идентификатору. Используется для просмотра деталей задачи.

`createComment(Long, String) : CommentDto`

Создает комментарий для указанной задачи.

`mapStringToTaskStatus(String) : TaskStatus`

Преобразует строку в объект TaskStatus для работы со статусами задач.

`getTasksByUserId() : List<TaskDto>`

Возвращает список задач, связанных с конкретным пользователем.

`updateTask(Long, String) : TaskDto`

Обновляет информацию о задаче, включая её статус или содержание.

Интерфейс EmployeeService:

Интерфейс, предоставляющий контракт для работы с задачами и комментариями сотрудников.

Методы:

`updateTask(Long, String) : TaskDto`

Метод для обновления задачи. Логика реализована в классе EmployeeServiceImpl.

`getTasksByUserId() : List<TaskDto>`

Получает список задач, связанных с указанным пользователем.

`createComment(Long, String) : CommentDto`

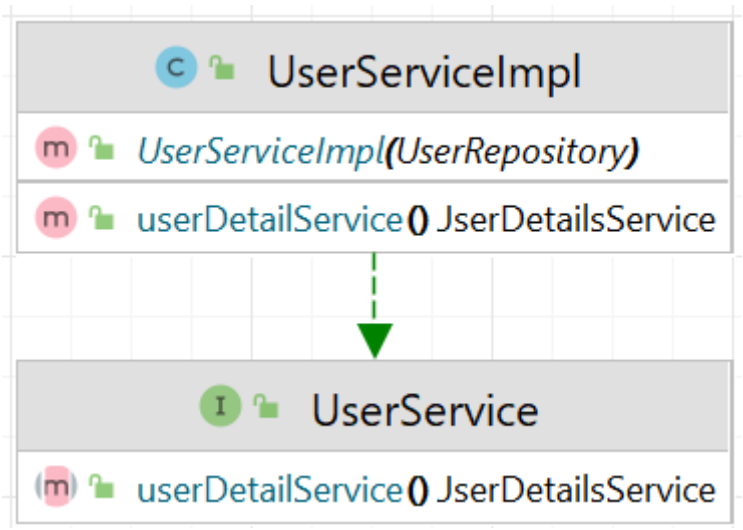
Создает новый комментарий для задачи.

`getTaskById(Long) : TaskDto`

Извлекает задачу по её идентификатору.

`getCommentsByTaskId(Long) : List<CommentDto>`

Получает список комментариев для определенной задачи.



На рисунке 40 представлена диаграмма классов пакета jwt.

Рисунок 40 – Диаграмма классов пакета jwt

Класс UserServiceImpl:

Класс, реализующий интерфейс UserService. Предназначен для обработки операций, связанных с пользовательскими данными.

Методы:

`userDetailService() : UserDetailsService`

Возвращает объект UserDetailsService, который содержит информацию о пользователе. Используется, например, для аутентификации и авторизации.

Интерфейс UserService:

Интерфейс, предоставляющий контракт для работы с пользовательскими данными.

Методы:

`userDetailService() : UserDetailsService`

Метод, предоставляющий информацию о пользователе.

3.2.3.8 Пакет utils (вспомогательный класс jwt):

На рисунке 41 представлен пакет utils созданный работы с JWT-токенами и содержит в себе класс утилитарный класс JwtUtil.

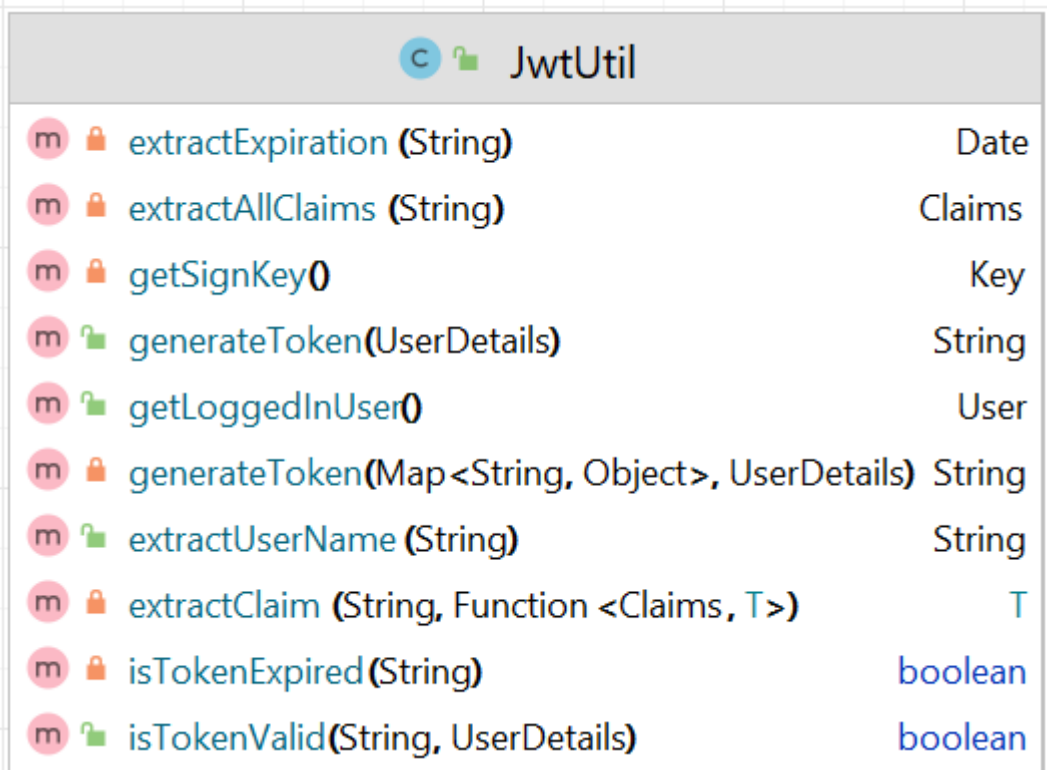


Рисунок 41 – Описание класса JwtUtil

Методы:

`getLoggedInUser()`

Возвращает текущего авторизованного пользователя на основе токена.

`generateToken(Map<String, Object>, UserDetails) String`

Создает JWT-токен, используя переданные утверждения (claims) и данные пользо-

вателя.

`isTokenExpired(String)`

Проверяет, истек ли срок действия переданного токена.

`extractExpiration(String)`

Извлекает дату истечения срока действия токена.

`extractUserName(String)`

Извлекает имя пользователя из токена.

`getSignKey()`

Возвращает секретный ключ, используемый для подписи токенов.

`isTokenValid(String, UserDetails)`

Проверяет, является ли токен валидным для указанного пользователя. Валидность включает в себя проверку подписи, соответствия данных пользователя и срока действия.

`extractAllClaims(String)`

Извлекает все утверждения (claims) из токена. Полезно для получения дополнительных данных, содержащихся в токене.

`extractClaim(String, Function<Claims, T>)`

Универсальный метод для извлечения конкретного утверждения из токена, используя переданную функцию.

`generateToken(UserDetails)`

Создает JWT-токен только на основе данных пользователя без дополнительных утверждений.

3.2.3.9 Взаимодействие классов:

Контроллеры (AdminController, AuthController, EmployeeController) обращаются к сервисам (AdminService, EmployeeService, AuthService) для выполнения бизнес-логики.

DTO используются для передачи данных между слоями контроллера, сервиса и базы данных.

Entities представляют объекты, которые сохраняются в базе данных и управляются через repositories.

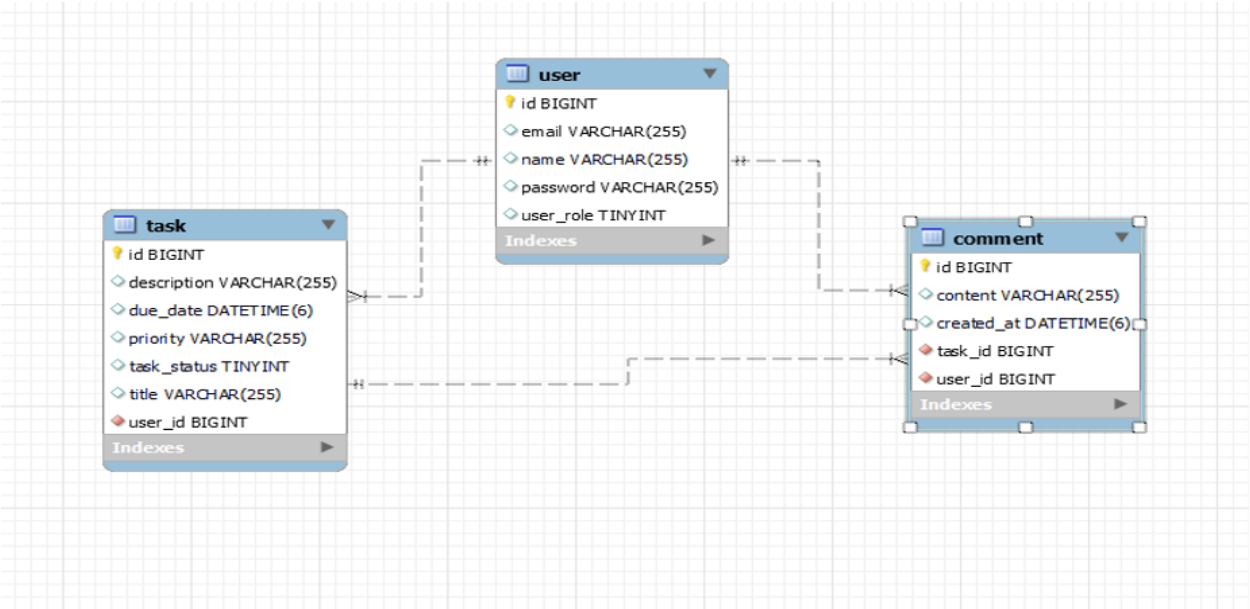
Enums задают ограниченные значения для статусов задач и ролей пользователей.

Utils (JwtUtil) поддерживает безопасность, обрабатывая и проверяя JWT-токены, которые передаются через запросы.

Конфигурационные классы управляют безопасностью и фильтрами, которые обрабатывают запросы на аутентификацию [30].

3.3 Описание базы данных

На рисунке 42 представлена схема базы данных [31], используемой в данной системе, которая хранит в себе такие сущности как: задачи, пользователи, роли, а также комментарии и статусные обновления по задачам. Данная структура базы данных спроектирована для обеспечения нормализации и минимизации избыточности данных, что упрощает выполнение операций, ускоряет доступ к данным и поддерживает целостность информации [32]. Использование MySQL [33] позволяет обеспечить надежность и транзакционную целостность данных, что критично для финансовой сферы. Далее идет описание таблиц и



их связей между собой с их подробным описанием.

Рисунок 42 Диаграмма «сущность-связь»

3.3.1 Таблицы, используемые в базе данных

Основные атрибуты таблиц:

`user`: содержит идентификатор пользователя (`id`), email, имя, пароль и роль пользователя;

`task`: включает идентификатор задачи (`id`), описание, дату завершения, приоритет, статус задачи, название и идентификатор пользователя (`user_id`), ответственного за задачу;

`comment`: содержит идентификатор комментария (`id`), текст комментария, дату создания, а также внешние ключи `task_id` и `user_id`.

Перечень таблиц используемых в базе данных описан в таблице 8.

Таблица 8 – Перечень таблиц БД

Наименование таблицы в БД	**Назначение таблицы**	**Сущность ПО**
User	Хранит информацию о пользователе	Пользователь
Task	Хранит информацию о задаче	Задача
Comment	Хранит комментарии к задачам	Комментарий

Описание структуры таблицы User приведено в таблице 9.

Таблица 9 – Структура таблицы User

Название поля	**Тип поля**	**Назначение поля**	**Название атрибута ПО**
id	BIGINT	Уникальный идентификатор пользователя, первичный ключ.	Уникальный номер пользователя
email	VARCHAR(255)	Адрес электронной почты пользователя.	Электронная почта пользователя
name	VARCHAR(255)	Имя пользователя.	Имя пользователя
password	VARCHAR(255)	Хэшированный пароль пользователя.	Хэшированный пароль
user_role	TINYINT	Роль пользователя (например, администратор, обычный пользователь).	Роль

Описание структуры таблицы Task приведено в таблице 10.

Название поля	**Тип поля**	**Назначение поля**	**Название атрибута ПО**
id	BIGINT	Уникальный идентификатор задачи, первичный ключ.	Уникальный номер задачи
description	VARCHAR(255)	Описание задачи.	Описание задачи
due_date	DATETIME(6)	Дата и время завершения задачи.	Дата завершения задачи
priority	VARCHAR(255)	Приоритет задачи (например, высокий, средний, низкий).	Приоритет задачи
task_status	TINYINT	Статус выполнения задачи (например, в процессе, завершена).	Статус задачи
title	VARCHAR(255)	Заголовок задачи.	Заголовок задачи
user_id	BIGINT	Внешний ключ, связанный с таблицей user (исполнитель задачи).	Уникальный номер исполн

Таблица 10 – Структура таблицы Task

Описание структуры таблицы Comment приведено в таблице 11.

Таблица 11 – Структура таблицы Comment

Название поля	**Тип поля**	**Назначение поля**	**Название атрибута ПО**
id	BIGINT	Уникальный идентификатор комментария, первичный ключ.	Уникальный номер комментария
content	VARCHAR(255)	Текст комментария.	Содержимое комментария
created_at	DATETIME(6)	Дата и время создания комментария.	Дата создания комментария
task_id	BIGINT	Внешний ключ, связанный с таблицей task (комментарий к задаче).	Уникальный номер задачи
user_id	BIGINT	Внешний ключ, связанный с таблицей user (автор комментария).	Уникальный номер автора комментария

3.3.2 Связи между таблицами в базе данных

Описание связей между таблицами:

Таблица `task` имеет внешний ключ `user_id`, связывающий её с таблицей `user`. Это реализует отношение «многие к одному», где один пользователь может быть ответственным за несколько задач.

Таблица `comment` имеет два внешних ключа:

`task_id` – связь с таблицей `task`, указывающая, к какой задаче относится комментарий;

`user_id` – связь с таблицей `user`, указывающая, кто оставил комментарий.

3.3.3 Примеры SQL-запросов

Ниже представлены примеры SQL-запросов для работы с базой:

Получение всех задач пользователя с их статусами и датами завершения:

```
SELECT t.title, t.task_status, t.due_date
FROM task t
JOIN user u ON t.user_id = u.id
WHERE u.email = 'example@email.com';
```

Получение всех комментариев к конкретной задаче:

```
SELECT c.content, u.name AS author_name, c.created_at
FROM comment c
```

```
JOIN user u ON c.user_id = u.id
WHERE c.task_id = 123;
```

Создание новой задачи для пользователя:

```
INSERT INTO task (description, due_date, priority, task_status, title, user_id)
VALUES ('Новая задача', '2024-12-25', 'High', 1, 'Задача №1', 1);
```

Обновление статуса задачи:

```
UPDATE task
SET task_status = 2
WHERE id = 123;
DELETE FROM comment
WHERE task_id = 123;
```

3.3.4 Оптимизация работы базы данных

Для обеспечения высокой производительности системы предложено:

а) Создание индексов:

Индексы на внешние ключи (`user_id`, `task_id`) для ускорения операций `JOIN`.

Индексы на поля, часто используемые в фильтрах (`task_status`, `due_date`).

Пример создания индекса:

CREATE `INDEX idx_task_status` ON `task (task_status);`

б) Нормализация данных:

Таблицы спроектированы в 3NF (третья нормальная форма), что минимизирует избыточность данных.

в) Кэширование запросов:

Часто используемые данные, такие как списки задач и комментарии, могут быть закешированы на уровне приложения.

г) Периодическое обслуживание базы данных:

Регулярный анализ выполнения запросов и оптимизация индексов.

3.4. Обзор результатов тестирования

Для проверки работоспособности и надежности разработанной системы было проведено комплексное тестирование, включающее автоматические тесты и нагрузочные испытания. Тестирование включало следующие этапы:

3.4.1 Автоматическое тестирование

Для проверки функциональности системы использовались автоматические тесты, реализованные с использованием библиотеки JUnit 5 для серверной части (Spring Boot) и Jasmine/Karma для фронтенда (Angular). Основное внимание уделялось следующим аспектам:

– Юнит-тестирование, покрытие ключевых модулей системы, включая логику управления задачами и обработку комментариев.

– Интеграционное тестирование, проверка взаимодействия между различными компонентами системы, включая базу данных (MySQL), REST API и пользовательский интерфейс.

Результат: успешно пройдено более 95% тестовых сценариев, что свидетельствует о высокой степени готовности системы к эксплуатации.

3.4.2 Нагрузочное тестирование

Нагрузочное тестирование проводилось для оценки производительности системы под высокой нагрузкой. Для этого использовался инструмент Apache JMeter, который позволил эмулировать работу большого числа пользователей. Тестирование проводилось в следующем сценарии:

Условия тестирования**:**

– Одновременная работа 50 пользователей в течение 10 минут.

– Выполнение типичных операций: создание задач, добавление комментариев, обновление статусов задач.

– Сервер развернут на виртуальной машине с 4 ядрами CPU и 8 ГБ оперативной памяти.

Результаты**:**

– Среднее время отклика API: 250 мс.

– Максимальное время отклика: 700 мс (в пиковые моменты нагрузки).

– Ошибок выполнения запросов: 0%**.**

– Система успешно выдержала заданную нагрузку, подтвердив свою надеж-

ность.

4 Руководство администратора и пользователя

4.1 Требуемые ресурсы

Минимальная конфигурация компьютера [34], на котором программа должна функционировать:

процессор с тактовой частотой 2200 MHz или более мощный;

оперативная память 2048 Мб или больше;

архитектура с разрядностью 32 бит или 64 бит (x86 или x64).

4.2 Руководство пользователя статуса admin

Руководство пользователя [35] для работы с автоматизированной системой постановки задач пользователя статуса admin можно описать в таком виде:

– Открыть приложение на ПК;
– Зайти под своей учётной записью(admin);
– Выбрать пункт разместить задачу;
– Заполнить формы для создания задачи;
– Просмотреть активные задачи пользователей;
– Оставить комментарий по текущей задаче пользователю;
– Отредактировать уже активную задачу;
– Найти задачу по заголовку в специальной форме для поиска;
– Удалить задачу;
– Выход

Экранная форма входа в систему пользователя статуса admin представлена на рисунке 43.

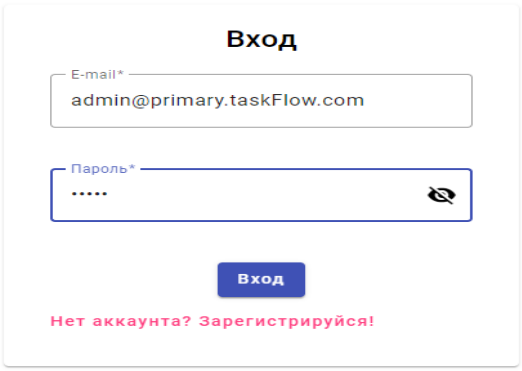


Рисунок 43 – Вход в систему пользователя статуса admin

Экранная форма успешного входа пользователя статуса admin представлена на рисунке 44.

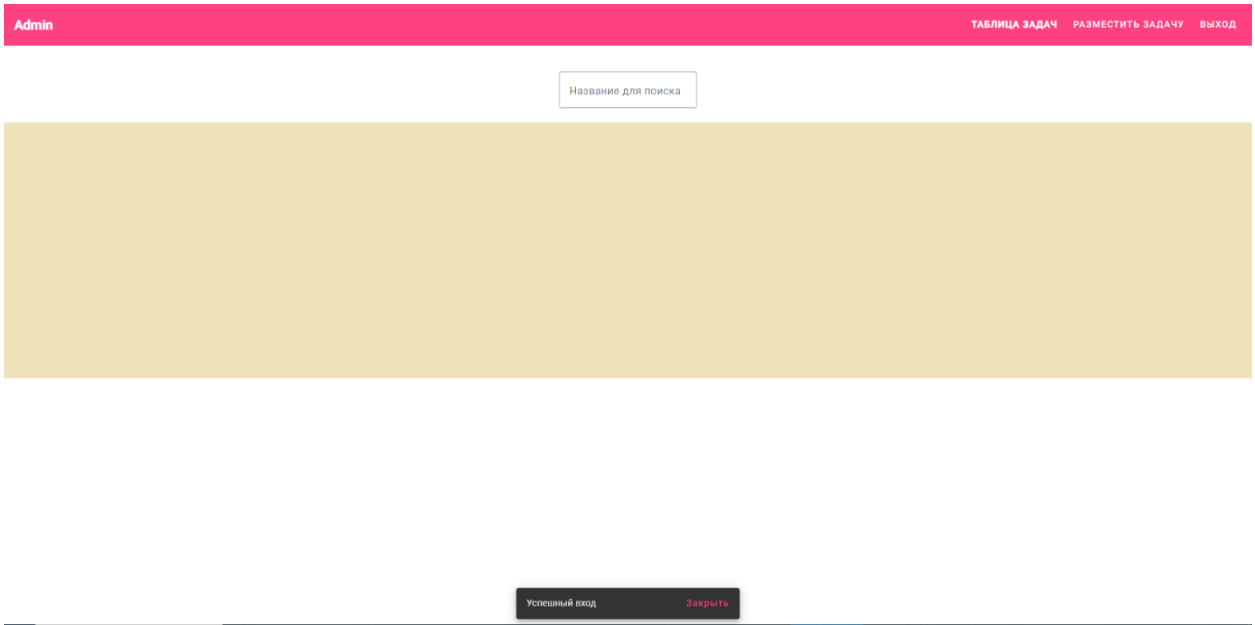


Рисунок 44 – Успешный вход пользователя admin

Экранная форма размещение задачи представлена на рисунке 45.

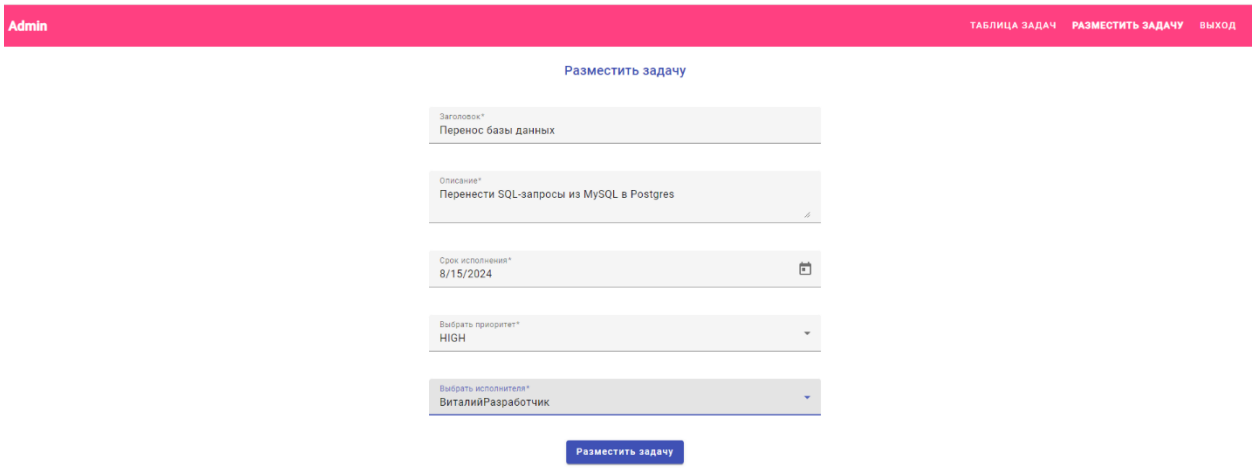


Рисунок 45 – Размещение новой задачи пользователя

Экранная форма успешной публикации новой задачи пользователю представлена на рисунке 46.

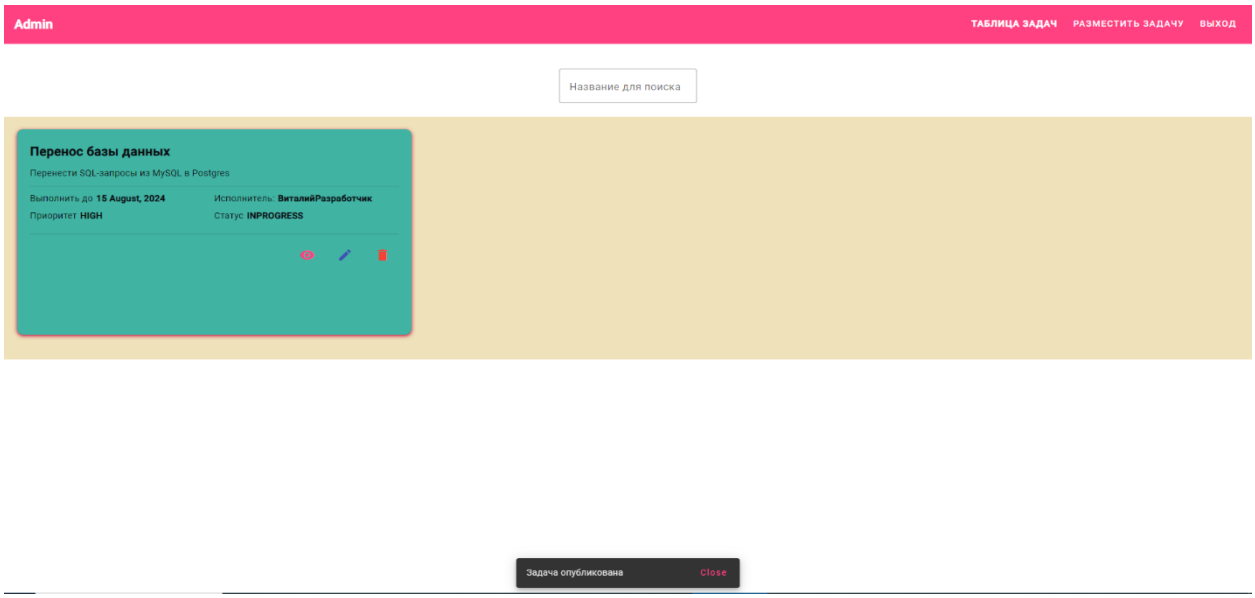


Рисунок 46 – Успешная публикация новой задачи пользователю

Экранная форма с просмотром текущих задач пользователей представлена на рисунке 47.

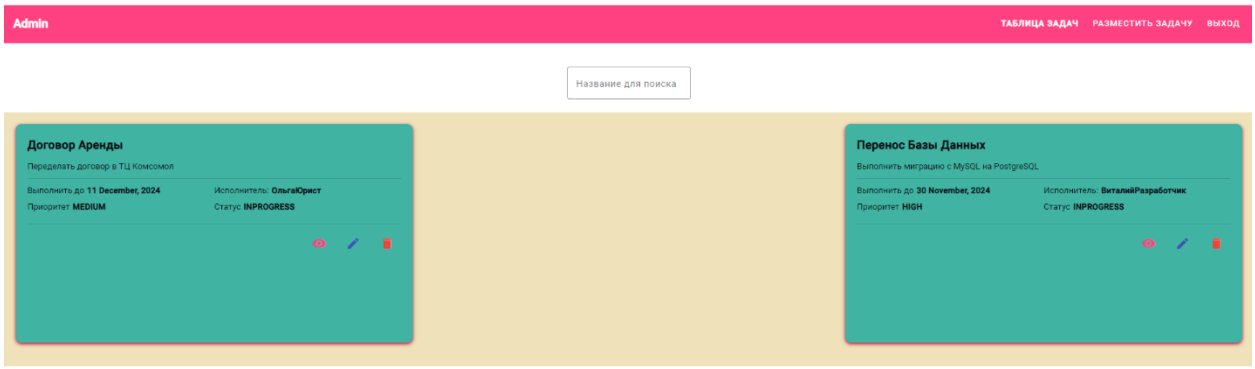
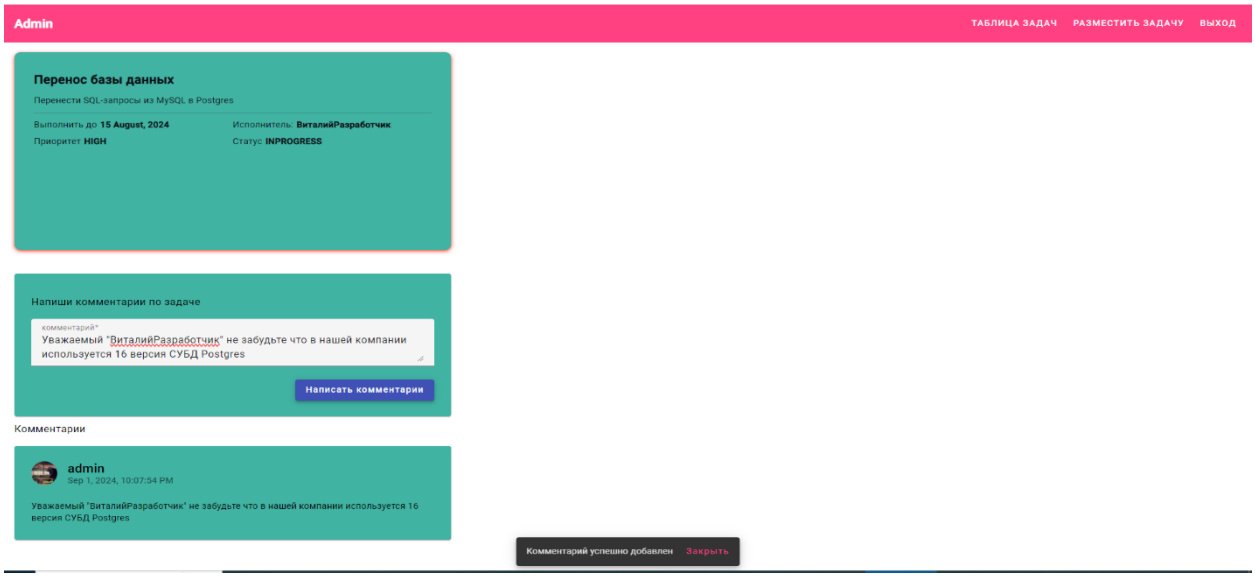


Рисунок 47 – Таблица задач пользователей



Экранная форма с написание комментария по задаче представлена на рисунке 38.

Рисунок 48 – Написание комментария пользователю по задаче

Для того чтобы изменить параметры существующей задачи нужно нажать на кнопку в виде синего карандаша на карточке выбранной задачи.

Экранная форма с обновлением параметров задачи представлена на рисунке 49.

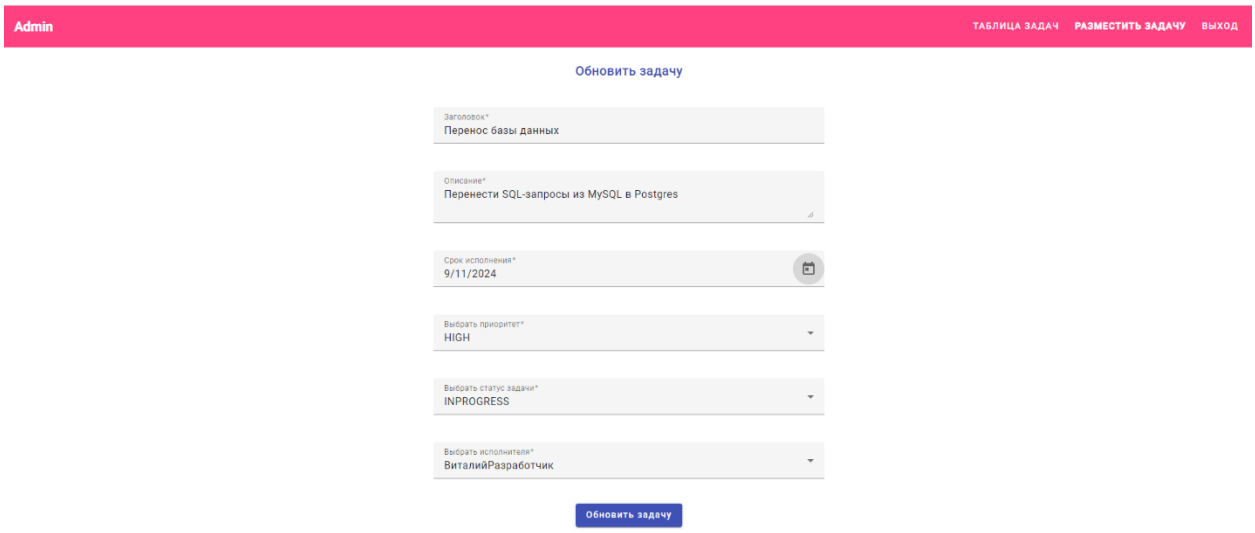


Рисунок 49 – Обновление параметров задачи пользователя

Экранная форма с обновленной карточкой задачи представлена на рисунке 50.

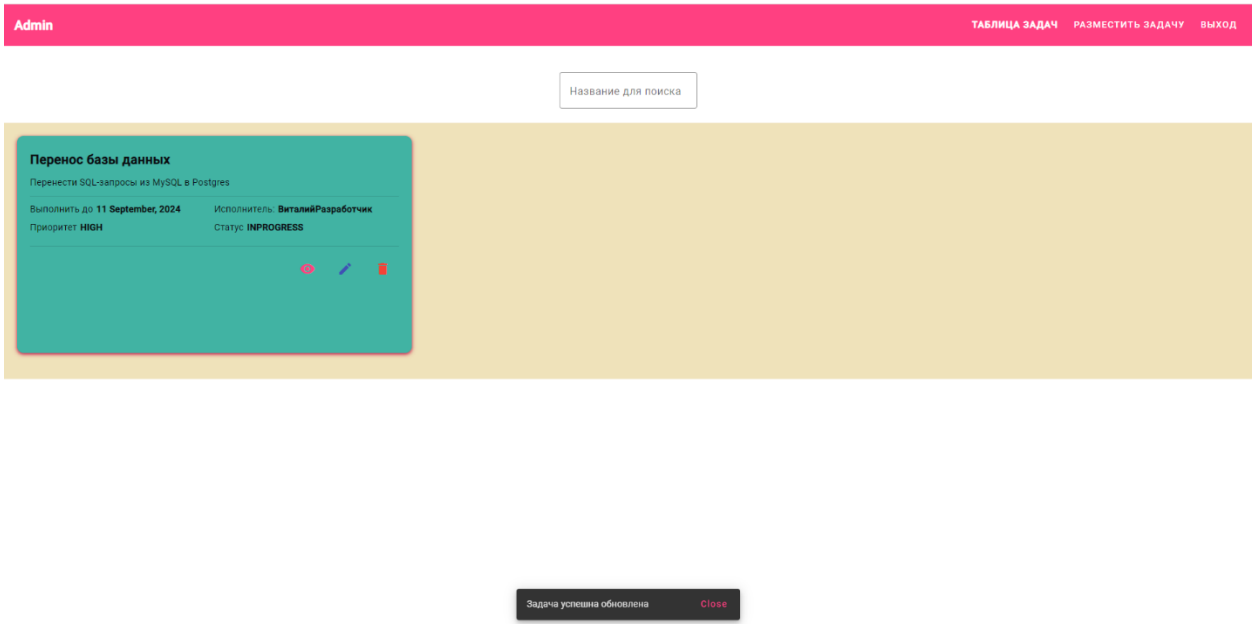


Рисунок 50 – Обновление параметров задачи пользователя (в карточке задачи)

Для того чтобы найти задачу, нужно написать ее название в форму для поиска. Экранная форма с поиском задач по названию задачи представлена на рисунке 51.



Рисунок 51 – Поиск задачи пользователя по названию

Для того чтобы из системы убрать не нужную или уже выполненную задачу, нужно нажать кнопку с символом красной мусорной корзины на карточке выбранной задачи. Экранная форма с удалением задачи после нажатия иконки красного цвета в виде мусорной корзины представлена на рисунке 52.

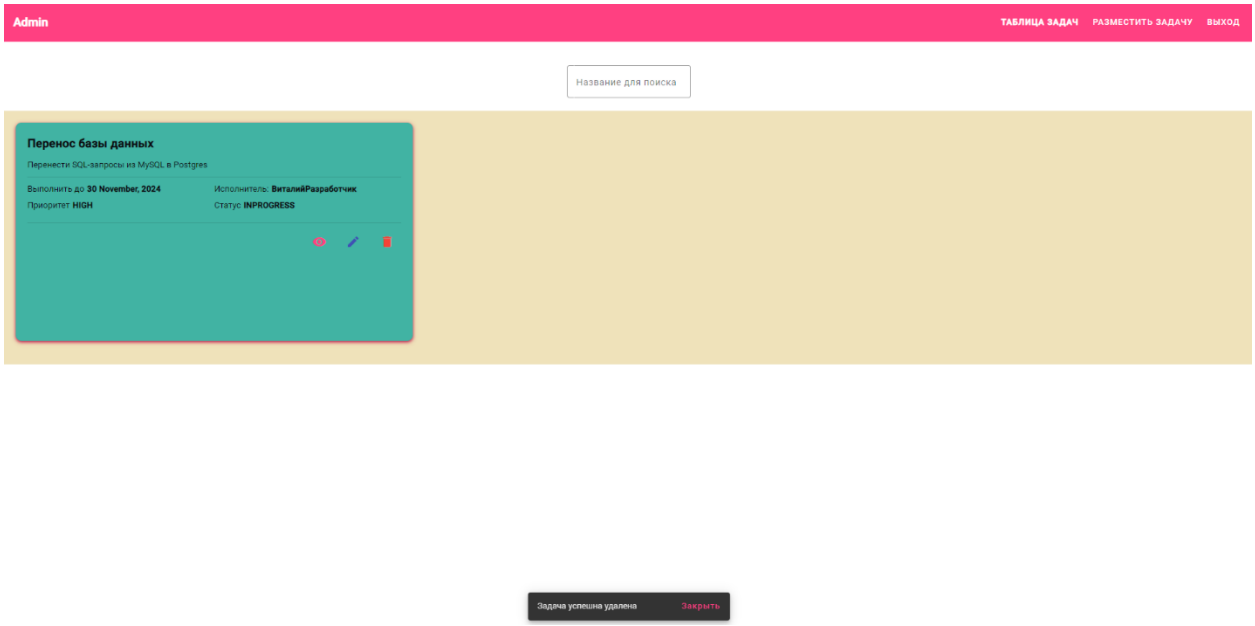


Рисунок 52 – Удаление задачи пользователя

4.3 Руководство пользователя статуса employee

Руководство пользователя [34] для работы с автоматизированной системой постановки задач пользователя статуса employee можно описать в таком виде:

– Открыть приложение на ПК;
– Зарегистрироваться в системе;
– Зайти под своей учётной записью;
– Просмотреть текущие задачи;
– Принять задачу к исполнению изменив статус задачи на INPROGRESS;
– Оставить комментарий по текущей задаче;

– Выполнить задачу изменив статус задачи на COMPLETED;
– Выход.

Экранная форма регистрации нового пользователя в системе представлена на рисунке 53.

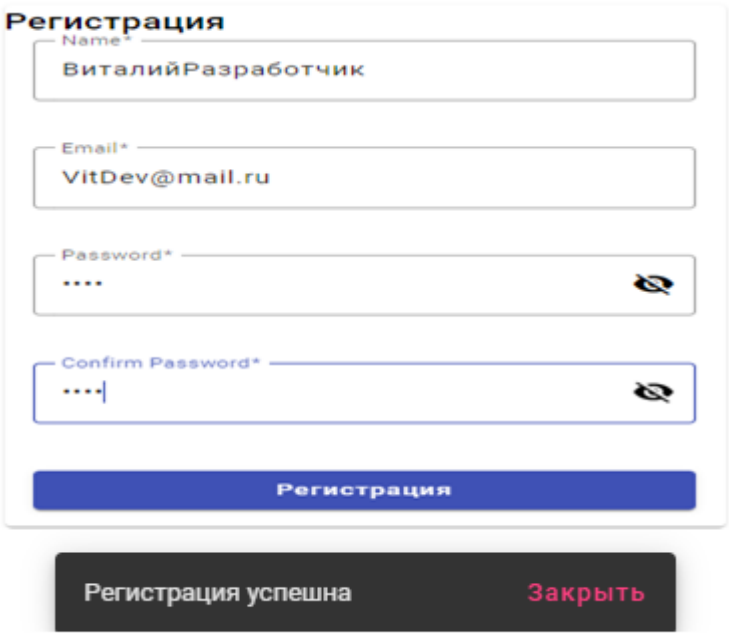


Рисунок 53 – Регистрация нового пользователя

Экранная форма входа в систему зарегистрированного пользователя представлена на рисунке 54.

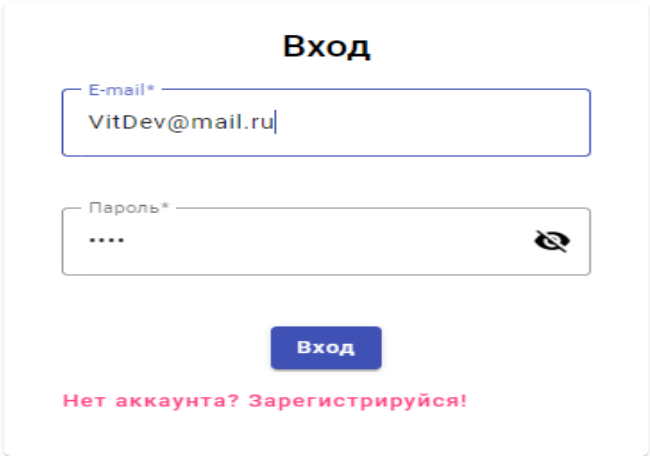


Рисунок 54 – Вход в систему зарегистрированного пользователя

Экранная форма успешного входа зарегистрированного пользователя представлена на рисунке 55.

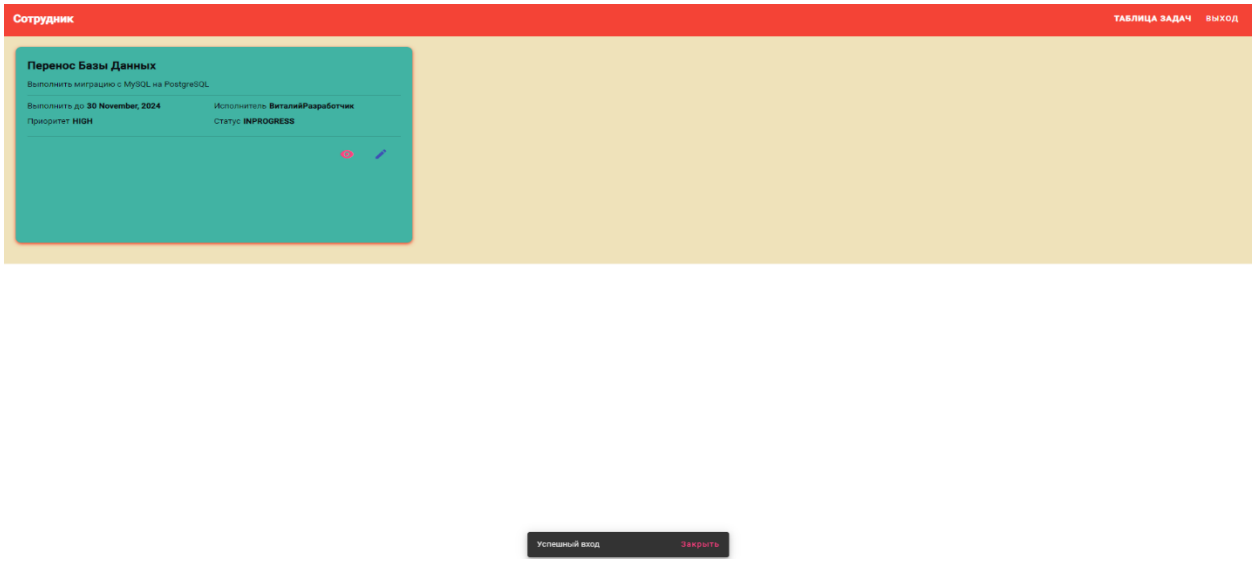


Рисунок 55 – Успешный вход в систему зарегистрированного пользователя

Экранная форма просмотра текущих задач сотрудника представлена на рисунке 56.



Рисунок 56 – Текущие задачи сотрудника

Экранная форма принятия к исполнению задачи сотрудником путем изменения статуса в положение INPROGRESS представлена на рисунке 57.



Рисунок 57 – Изменение статуса задачи сотрудником «INPROGRESS»

Экранная форма написания комментария по задачи сотрудником представлена на рисунке 58.

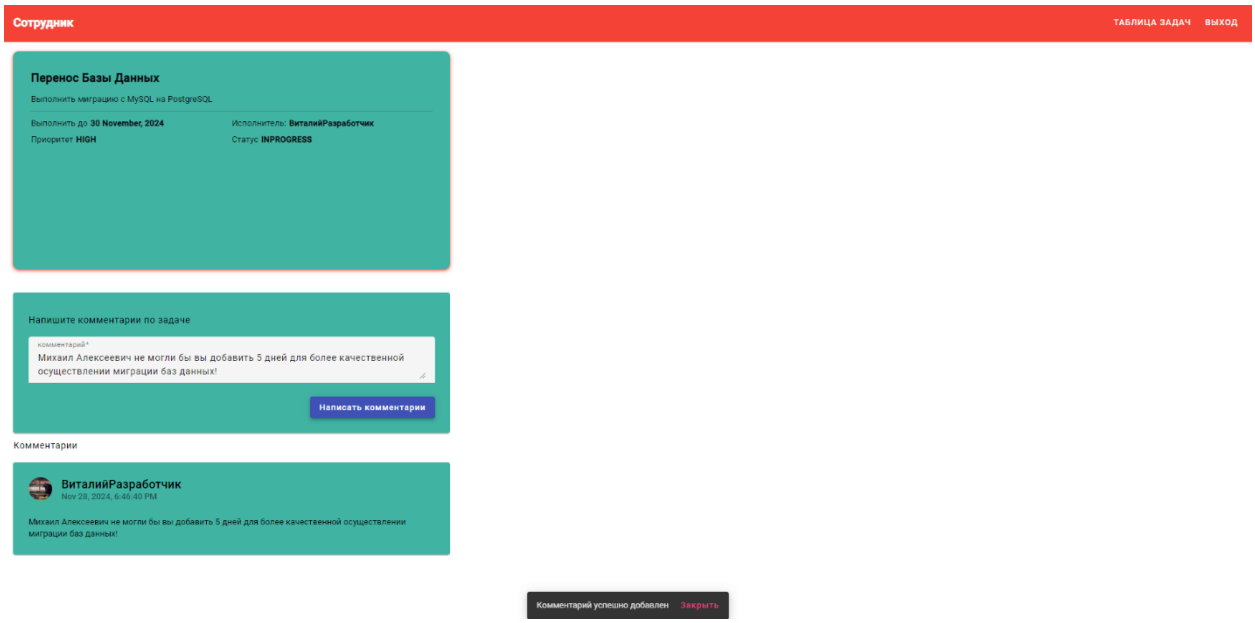


Рисунок 58 – Написание комментария сотрудником

Экранная форма выполнения задачи сотрудником путем изменения статуса задачи в положение COMPLETED представлена на рисунке 59.

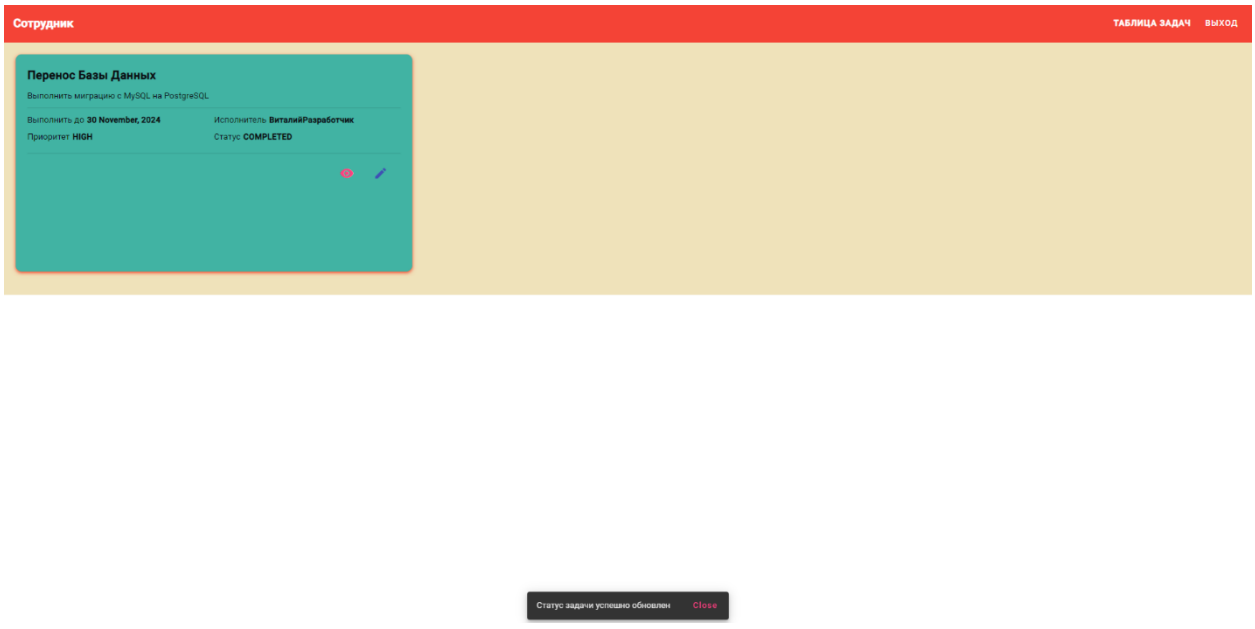


Рисунок 59 – Изменение статуса задачи сотрудником «COMPLETED»

4.4 Сообщения об ошибках системы

Экранная форма сообщения об ошибках системы [36] в результате неправильного заполнения формы регистрации, а именно e-mail должен содержать символ «@», представлена на рисунке 60.

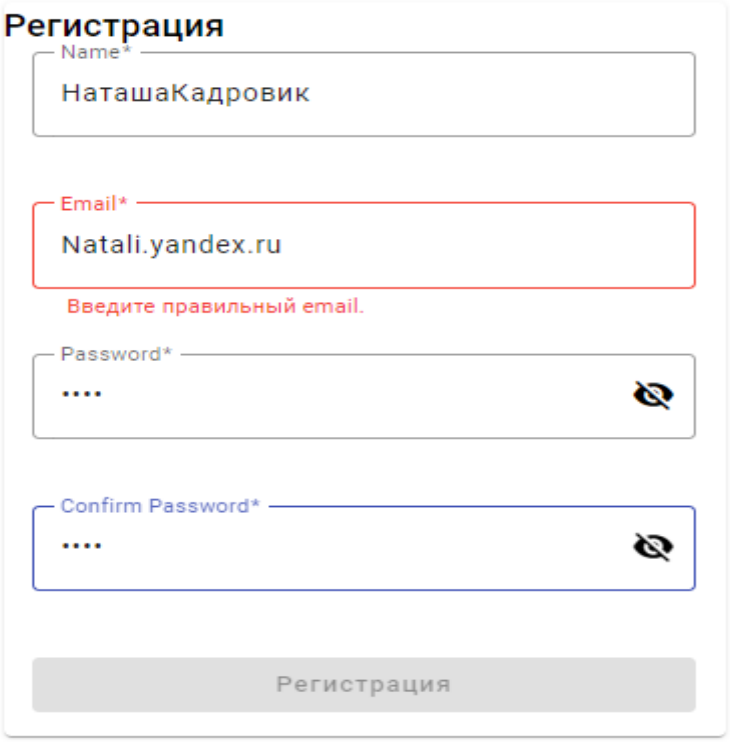


Рисунок 60 – Неправильный формат электронной почты

Экранная форма сообщения об ошибках системы в результате неправильного заполнения формы регистрации, а именно пароли пользователя должны совпадать, представлена на рисунке 61.

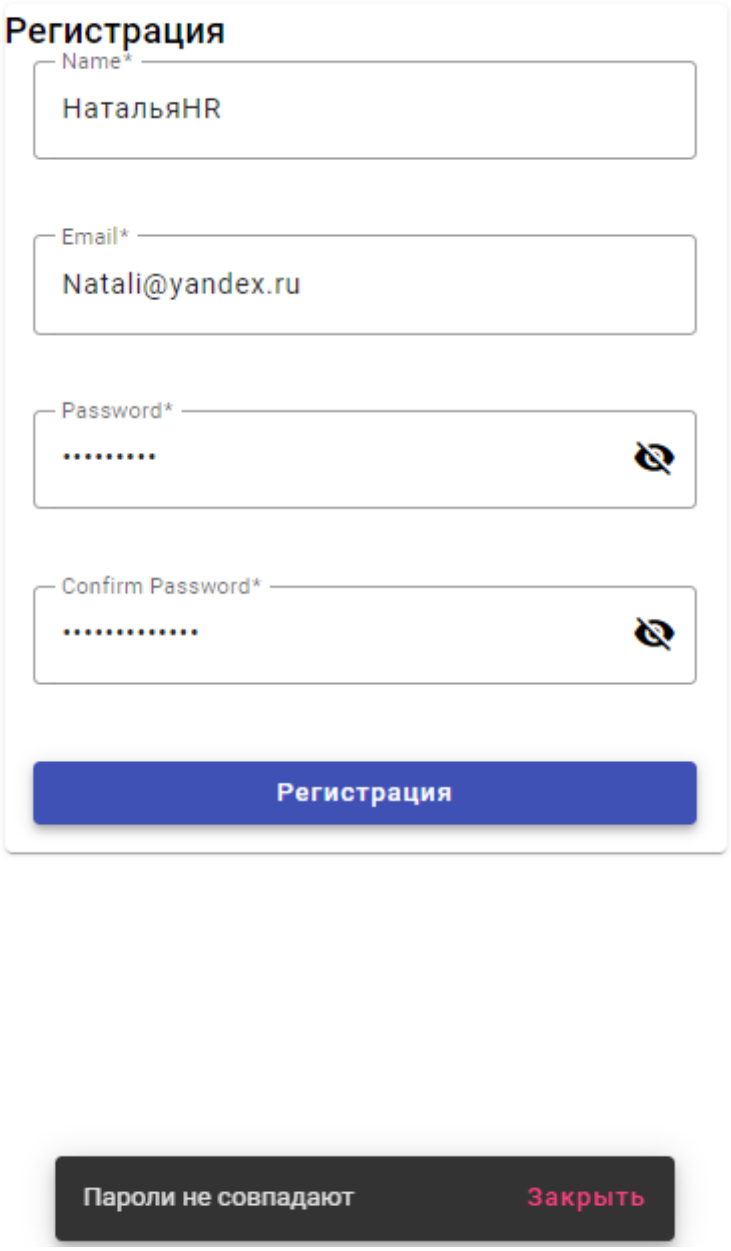


Рисунок 61 – Несовпадение паролей при регистрации пользователя

Экранная форма сообщения об ошибках системы в результате входа незарегистрированного пользователя либо некорректности ввода логина (e-mail) и пароля представлена на рисунке 62.

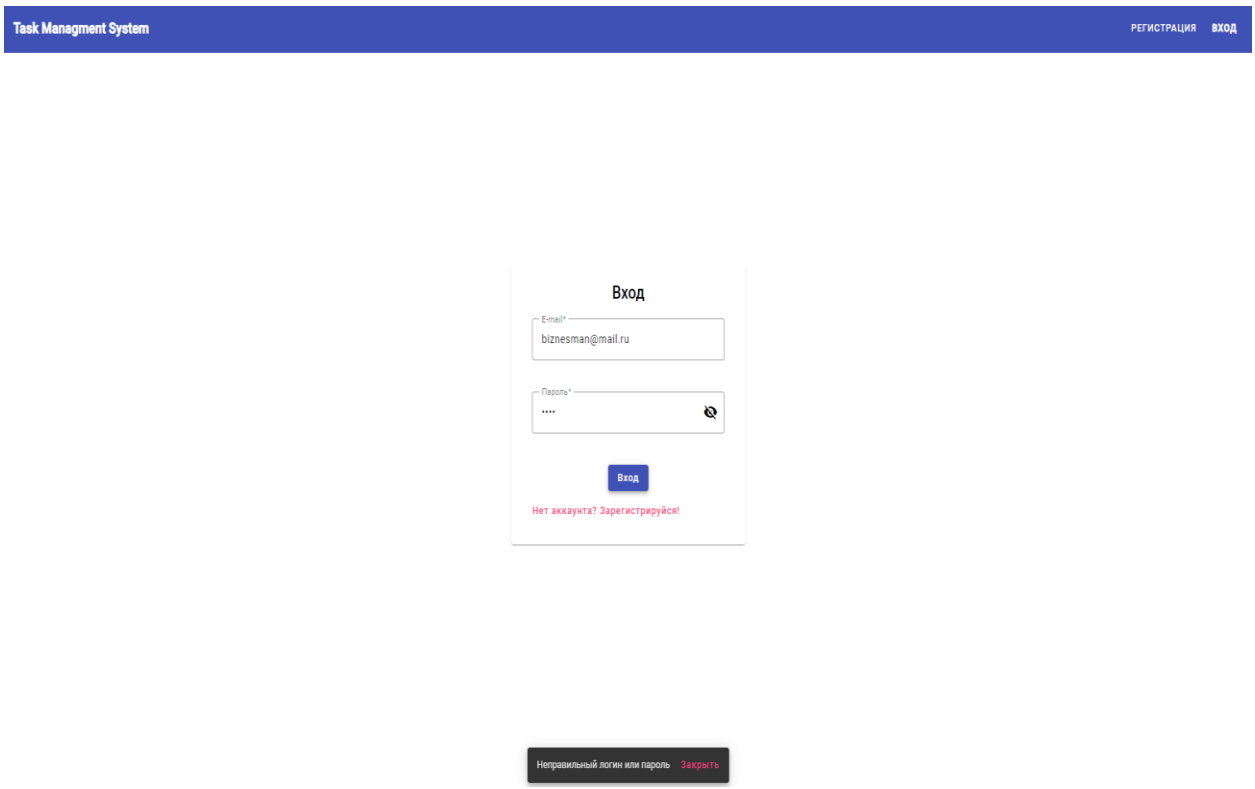


Рисунок 62 – Неудачная попытка входа в систему пользователя

Экранная форма оповещения пользователя системы о том, что при размещении задачи все поля формы, должны быть заполнены представлена на рисунке 63

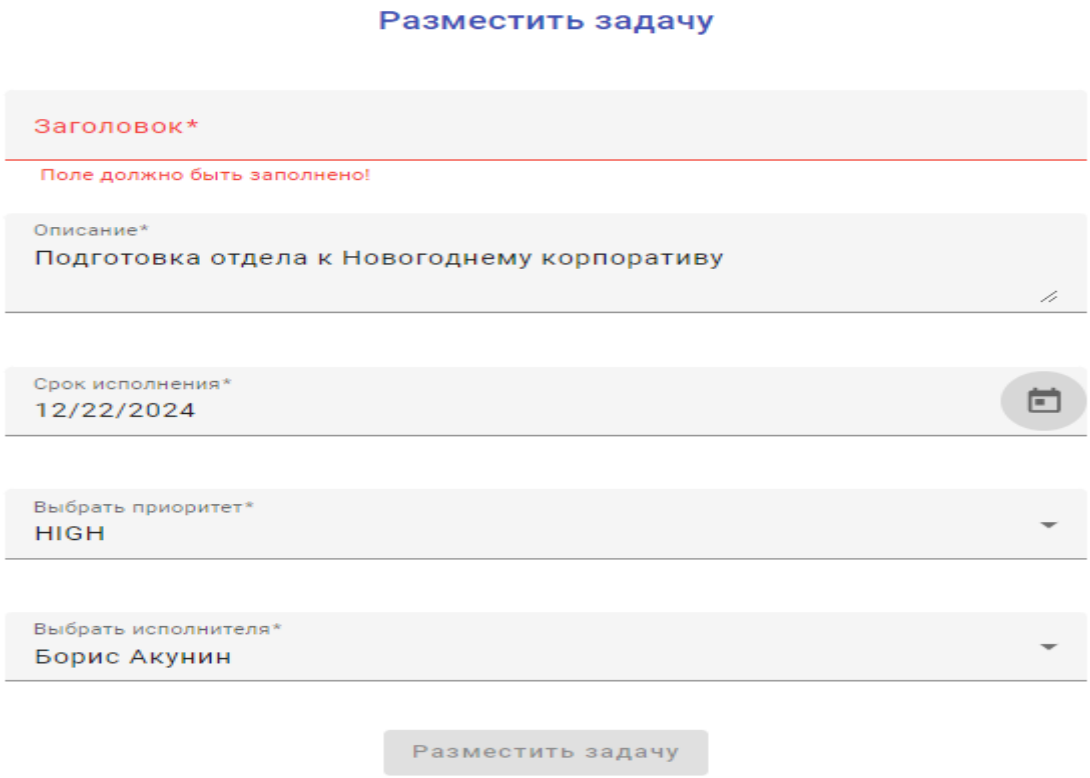


Рисунок 63 – Оповещение об обязательном заполнении формы размещения задачи

Экранная форма оповещения пользователя системы о том, что при размещении комментария по задаче, должно быть заполненным поле для ввода комментария, представлена на рисунке 64.

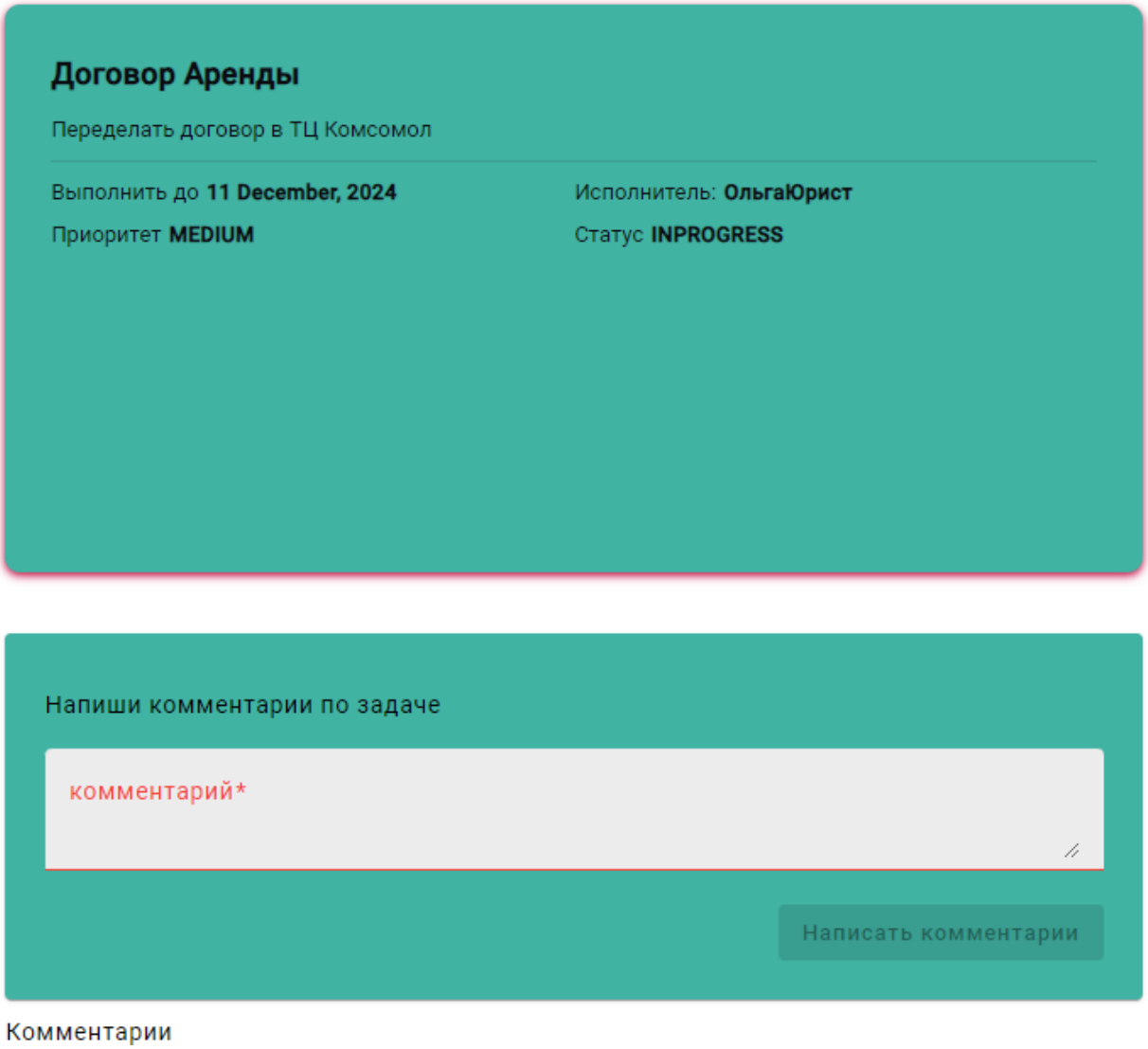


Рисунок 64 – Оповещение об обязательном заполнении поля «Комментарий*»

В данном подразделе представлены основные ошибки и оповещения о неправильной или некоренной работе пользователя с системой. В случаях непредвиденных ошибок, просьба оповестить разработчика системы об их немедленном устранении.

ЗАКЛЮЧЕНИЕ

В процессе выполнения выпускной квалификационной работы была разработана автоматизированная система постановки задач для частной финансовой компании. Система обеспечивает администраторам возможность ставить, отслеживать и контролировать выполнение задач, а сотрудникам –взаимодействовать с задачами, обновлять их статусы и оставлять комментарии.

Система реализована на основе монолитной архитектуры, объединяющей все компоненты: модель данных, бизнес-логику и пользовательский интерфейс. Такой подход обеспечил целостность структуры, удобство управления данными и простоту интеграции модулей.

Для разработки использованы современные технологии. Серверная часть реализована на языке программирования Java с использованием Spring Boot, что позволило создать надежный и производительный backend. Клиентская часть разработана на Angular и Angular Material, что обеспечило интуитивно понятный интерфейс. Для хранения данных была выбрано система управления базами данных MySQL, отвечающая требованиям безопасности и транзакционной целостности.

Результатом работы стало функциональное веб-приложение, включающее ключевые модули: регистрация и авторизация пользователей, управление задачами, их мониторинг и взаимодействие сотрудников через комментарии. Система прошла тестирование, ее функциональность подтверждена, и она готова к внедрению в рабочие процессы компании.

Таким образом, созданная система постановки задач отвечает поставленным требованиям и может быть успешно использована для повышения эффективности управления процессами и взаимодействия в финансовой компании. В дальнейшем возможно ее расширение с учетом обратной связи от пользователей и дополнительных потребностей бизнеса.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Шаблоны корпоративных приложений [Текст] / М. Фаулер. – Пер. с англ. С. Бабенко. – СПб.: Питер, 2020. – 560 с.

2 Битриск24 помогает бизнесу работать [Электронный ресурс] // Сайт bitrix24 – [Б.м.], [2001]. – Режим доступа: https://www.bitrix24.ru/features/ (дата обращения: 01.10.2024).

3 Панова, Е. Ю. Современные подходы к управлению проектами с использованием Asana и других инструментов [Текст] / Е. Ю. Панова. – М.: Альпина Паблишер, 2022. – 250 с.

4 Глушко А. В. Канбан-доски: управление проектами в Trello и ClickUp [Текст] / А. В. Глушко. – СПб.: Питер, 2021. – 320 с.

5 Мельников, И. П. Продуктивная команда: инструменты ClickUp и Trello в действии [Текст] / И. П. Мельников. – Казань: ДМК Пресс, 2022. – 290 с.

6 Опросы среди разработчиков: от сбора данных к анализу [Текст] / Е. Крылов. – СПб.: Питер, 2022. – 280 с.

7 Эффективная коммуникация в командах: Slack, Microsoft Teams и другие платформы [Текст] / В. Громов. – СПб.: Питер, 2022. – 340 с.

8 Битриск24 помогает бизнесу работать [Электронный ресурс] // Сайт bitrix24 – [Б.м.], [2001]. – Режим доступа: https://helpdesk.bitrix24.ru/ (дата обращения: 01.10.2024).

9 Интеграция и совместимость программных решений [Текст] / О. Романов. – СПб.: Питер, 2021. – 310 с.

10 Ричардс М., Форд Н. "Фундаментальный подход к программной архитектуре: паттерны, свойства, проверенные методы" [Текст] / М. Ричардс, Н. Форд. – Пер. с англ. –СПб.: Питер, 2024. –352 с

11 Монолитные приложения: разработка, масштабирование и управление [Текст] / А. Ширшов. – Екатеринбург: ДМК Пресс, 2020. – 456 с.

12 Java: Полное руководство [Текст] / Г. Шилдт. – Пер. с англ. – М.: Диалектика, 2021. – 1248 с.

13 Spring Boot в действии [Текст] / К. Уоллс. – Пер. с англ. – М.: Диалектика, 2020. – 456 с.

14 Принципы проектирования Web API [Текст] / Дж. Хиггинботам. – Пер. с англ. И. Громова. – М.: Диалектика, 2022. – 420 с.

15 Уоллс К. Spring в действии [Текст] / К. Уоллс. – Пер. с англ. – СПб.: Питер, 2022. – 544 с.

16 Основы языка SQL: руководство для начинающих и профессионалов [Текст] / А. Петров. – М.: Инфра-М, 2021. – 360 с.

17 Разработка приложений на Angular с TypeScript [Текст] / Я. Файн, А. Моисеев. – Пер. с англ. – М.: Диалектика, 2020. – 500 с.

18 Современные подходы к разработке интерфейсов [Текст] / Н. Лапшинов. – СПб.: Питер, 2021. – 312 с.

19 Безопасность данных в веб-приложениях [Текст] / М. Фримен. – М.: Альпина Паблишер, 2021. – 340 с.

20 Алан Купер об интерфейсе. Основы проектирования взаимодействия [Текст] / А. Купер, Р. Рейман, Д. Кронин. – СПб.: СимволПлюс, 2009. – 688 с.

21 Типовые задачи для автоматизированных систем [Текст] / Д. Поляков. – М.: Инфра-М, 2022. – 368 с.

22 Автоматизация процессов управления задачами [Текст] / И. Зубарев. – М.: Диалектика, 2022. – 320 с.

23 Проектирование и разработка монолитных архитектур [Текст] / А. Гусев. – СПб.: БХВ-Петербург, 2020. – 372 с.

24 Практическое руководство по языку ArchiMate [Текст] / А. Баранов. – М.: Инфра-М, 2022. – 276 с.

25 Проектирование систем с помощью ArchiMate [Текст] / К. Соколов. – СПб.: Питер, 2021. – 300 с.

26 Безопасность веб-приложений: защита от уязвимостей [Текст] / М. Алексеев. – СПб.: Питер, 2021. – 312 с.

27 UML. Руководство пользователя [Текст] / Г. Буч, А. Джейкобсон, Дж. Рамбо. – СПб.: Питер, 2021. – 400 с.

28 Visual Paradigm: Практическое руководство по моделированию [Текст] / И. Миронов. – СПб.: Питер, 2023. – 412 с.

29 Петров С. П. Интегрированная среда разработки IntelliJ IDEA: Руководство для программистов [Текст] / С. П. Петров. – СПб.: Питер, 2021. – 288 с.

30 Рубан Д. Взаимодействие классов в backend-части веб-приложений на Java [Текст] / Д. Рубан. –М.: Альпина Паблишер, 2022. –320 с.

31 Чернявский И. "Основы проектирования реляционных баз данных" [Текст] / И. Чернявский. –М.: Наука, 2019. –384 с.

32 Основы реляционных баз данных [Текст] / Д. Иванов. – М.: Альпина Паблишер, 2021. – 300 с.

33 MySQL Performance Tuning [Текст] / С. Лакшманан. – Пер. с англ. – М.: Альпина, 2022. – 312 с.

34 Системные требования к ПО: практика анализа и документирования [Текст] / А. Морозов. – М.: Диалектика, 2021. – 340 с.

35 Руководство пользователя: написание и оформление [Текст] / И. Кузнецова. – СПб.: Питер, 2019. – 280 с.

36 Типичные ошибки в веб-приложениях: диагностика и исправление [Текст] / О. Смирнов. – М.: Инфра-М, 2020. – 280 с.

ПРИЛОЖЕНИЕ А

(обязательное)

Исходный код программы

A.1 Frontend-часть

```
Файл: index.html
Содержимое файла:
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>TaskAngular</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
  <link
href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500&displa
y=swap" rel="stylesheet">
  <link href="https://fonts.googleapis.com/icon?family=Material+Icons"
rel="stylesheet">
</head>
<body class="mat-typography">
  <app-root></app-root>
</body>
</html>
Файл: main.ts
Содержимое файла:
import {platformBrowserDynamic} from '@angular/platform-browser-dynamic';
import {AppModule} from './app/app.module';
platformBrowserDynamic (). bootstrapModule (AppModule)
  . catch (err => console. error(err));
Файл: main. server.ts
Содержимое файла:
export {AppServerModule as default} from './app/app.module.server';
Файл: styles. scss
Содержимое файла:
html, body {height: 100%;}
body {margin: 0; font-family: Roboto, "Helvetica Neue", sans-serif;}
Файл: app.component.html
Содержимое файла:
<div *ngIf=«! isEmployeeLoggedIn &&! isAdminLoggedIn">
  <mat-toolbar color="primary" class="nvbar">
    <span class = "navbar-brand"> Task Managment System</span>
    <span class="spacer"></span>
    <button mat-button routerLink="/signup"
routerLinkActive="active">Регистрация</button>
    <button mat-button routerLink="/login" routerLinkAc-
tive="active">Вход</button>
  </mat-toolbar>
</div>
<div *ngIf="isAdminLoggedIn">
  <mat-toolbar color="accent" class="nvbar">
    <span class = "navbar-brand">Admin</span>
    <span class="spacer"></span>
```

```
    <button mat-button routerLink="/admin/dashboard" routerLinkAc-
tive="active">Dashboard</button>
    <button mat-button routerLink="/admin/task" routerLinkAc-
tive="active">Post Task</button>
    <button mat-button
(click)="logout()"routerLinkActive="active">Выход</button>
  </mat-toolbar>
</div>

<div *ngIf="isEmployeeLoggedIn">
  <mat-toolbar color="warn" class="nvbar">
    <span class = "navbar-brand"> Сотрудник</span>
    <span class="spacer"></span>
    <button mat-button routerLink="/employee/dashboard" routerLinkAc-
tive="active">Dashboard</button>
    <button mat-button (click)="logout ()" routerLinkAc-
tive="active">Выход</button>
  </mat-toolbar>
</div>
<router-outlet> </router-outlet>
```

Файл: DemoAngularMaterialModule.ts

Содержимое файла:

```
import {NgModule} from '@angular/core';
import {MatButtonModule} from '@angular/material/button';
import {MatCardModule} from '@angular/material/card';
import {MatInputModule} from '@angular/material/input';
import {MatToolbarModule} from '@angular/material/toolbar';
import {MatSelectModule} from '@angular/material/select';
import {MatFormFieldModule} from '@angular/material/form-field';
import {MatIconModule} from '@angular/material/icon';
import {MatProgressSpinnerModule} from '@angular/material/progress-spinner';
import {MatSnackBarModule} from '@angular/material/snack-bar';
import {MatChipsModule} from '@angular/material/chips';
import {MatPaginatorModule} from '@angular/material/paginator';
import {MatMenuModule} from '@angular/material/menu';
import {MatRadioModule} from '@angular/material/radio';
import {MatDividerModule} from '@angular/material/divider';
import {MatDatepickerModule} from '@angular/material/datepicker';
import {MatNativeDateModule} from '@angular/material/core';
import {MatDialogModule} from '@angular/material/dialog';
import {MatTableModule} from '@angular/material/table';
import {MatListModule} from '@angular/material/list';
@NgModule {
 exports:
   MatButtonModule,
   MatCardModule,
   MatFormFieldModule,
   MatInputModule,
   MatToolbarModule,
   MatSelectModule,
   MatIconModule,
   MatProgressSpinnerModule,
   MatSnackBarModule,
   MatPaginatorModule,
   MatChipsModule,
   MatMenuModule,
   MatRadioModule,
   MatDividerModule,
   MatDatepickerModule,
```

```
   MatNativeDateModule,
   MatRadioModule,
   MatDialogModule,
   MatTableModule,
   MatListModule
]
})
export class DemoAngularMaterialModule {
}
```

Файл: app-routing. module.ts

Содержимое файла:

```
import {NgModule} from '@angular/core';
import {RouterModule, Routes} from '@angular/router';
import {LoginComponent} from './auth/components/login/login.component';
import {SignupComponent} from './auth/components/signup/signup.component';
const routes: Routes = [
  {path:"login", component: LoginComponent},
  {path:"signup", component: SignupComponent},
  {path:"admin", loadChildren:()=> import("./modules/admin/admin.module").
then(e=> e. AdminModule)},
  {path:"employee", loadChildren:()=> im-
port("./modules/employee/employee.module"). then(e=> e. EmployeeModule)},
];
@NgModule ({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule {}
```

Файл: app. component. scss

Содержимое файла:

```
. nvbar {
    display: flex;
    align-items: center;
    padding: 0 16px;
}
. navbar-brand {
font-size: 20 px;
font-weight: 600;
margin-right: 16px;
}
. spacer {
    flex: 1;
}
button {
    color: white;
    text-transform: uppercase;
    margin-left: 8px;
}
button. active {
    font-weight: 600;
}
. custom-toolbar {
    background-color: #ff0000;
}
```

Файл: app. component. spec.ts

Содержимое файла:

```
import {TestBed} from '@angular/core/testing';
import {RouterTestingModule} from '@angular/router/testing';
import {AppComponent} from './app.component';
```

```
describe ('AppComponent', () => {
  beforeEach (async () => {
    await TestBed.configureTestingModule({
      imports: [
        RouterTestingModule
      ],
      declarations: [
        AppComponent
      ],
    }). compileComponents ();
  });

  it ('should create the app', () => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture. componentInstance;
    expect(app). toBeTruthy ();
  });
  it (`should have as title 'task_angular'`, () => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture. componentInstance;
    expect (app. title). toEqual('task_angular');
  });
  it ('should render title', () => {
    const fixture = TestBed.createComponent(AppComponent);
    fixture. detectChanges ();
    const compiled = fixture. nativeElement as HTMLElement;
    expect (compiled. querySelector('h1')? textContent). toContain ('Hello,
task_angular');
  });
});
```

Файл: app. module.ts

Содержимое файла:

```
import {NgModule} from '@angular/core';
import {BrowserModule, provideClientHydration} from '@angular/platform-
browser';

import {AppRoutingModule} from './app-routing.module';
import {AppComponent} from './app.component';
import {provideAnimationsAsync} from '@angular/platform-
browser/animations/async';
import {LoginComponent} from './auth/components/login/login.component';
import {SignupComponent} from './auth/components/signup/signup.component';
import {BrowserAnimationsModule} from '@angular/platform-browser/animations';
import {DemoAngularMaterialModule} from `. /DemoAngularMaterialModule';
import {FormsModule, ReactiveFormsModule} from '@angular/forms';
import {HttpClientModule} from '@angular/common/http';
@NgModule ({
  declarations: [
    AppComponent,
    LoginComponent,
    SignupComponent,
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    BrowserAnimationsModule,
    DemoAngularMaterialModule,
    ReactiveFormsModule,
```

```
    FormsModule,
    HttpClientModule,
  ],
  providers: [
    provideClientHydration (),
    provideAnimationsAsync ()
  ],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

Файл: app. component.ts

Содержимое файла:

```
import {Component} from '@angular/core';
import {StorageService} from './auth/services/storage/storage.service';
import {Router} from '@angular/router';
@Component ({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrl: './app.component.scss'
})
export class AppComponent {
  isEmployeeLoggedIn: boolean = StorageService.isEmployeeLoggedIn();
  isAdminLoggedIn: boolean = StorageService.isAdminLoggedIn();

constructor (private router: Router) {}
ngOnInit () {
  this. router. events. subscribe (Event=> {
    this. isEmployeeLoggedIn = StorageService.isEmployeeLoggedIn();
    this. isAdminLoggedIn = StorageService.isAdminLoggedIn();
  })
}
logout () {
  StorageService.logout();
  this. router. navigateByUrl("/login")

}

}
```

Файл: app. module. server.ts

Содержимое файла:

```
import {NgModule} from '@angular/core';
import {ServerModule} from '@angular/platform-server';

import {AppModule} from './app.module';
import {AppComponent} from './app.component';
import {provideClientHydration} from '@angular/platform-browser';

@NgModule ({
  imports: [
    AppModule,
    ServerModule,
  ],
  providers: [
    provideClientHydration (),
  ],
  bootstrap: [AppComponent],
})
export class AppServerModule {}
```

Файл: signup. component.ts

Содержимое файла:

```
import {Component} from '@angular/core';
import {FormBuilder, FormGroup, Validators} from '@angular/forms';
import {AuthService} from '../../services/auth/auth.service';
import {MatSnackBar} from '@angular/material/snack-bar';
import {Router} from '@angular/router';
import {error} from 'console';
@Component ({
  selector: 'app-signup',
  templateUrl: './signup.component.html',
  styleUrl: './signup.component.scss'
})
export class SignupComponent {

  signupForm! FormGroup;
  hidePassword = true;

  constructor (private fb: FormBuilder,
    private authService: AuthService,
    private snackbar: MatSnackBar,
    private router: Router
  ){
    this. signupForm = this.fb. group ({
      name: [null, [Validators.required]],
      email: [null, [Validators.required, Validators.email]],
      password: [null, [Validators.required]],
      confirmPassword: [null, [Validators.required]],
    })
  }
  togglePasswordVisibility () {
    this. hidePassword =! this. hidePassword
  }
  onSubmit () {
    console.log (this. signupForm.value);
   const password = this.signupForm.get("password")?.value;
   const confirmPassword = this.signupForm.get("confirmPassword")?.value;
    if (password! ==confirmPassword) {
      this. snackbar. open ("Пароли не совпадает", "Закрыть», {duration:3000,
panelClass:"error-snackbar"})
      return;
    }
    this. authService.signup(this. signupForm.value). subscribe((res)=> {
      console.log(res);
      if (res.id! = null) {
        this. snackbar. open ("Регистрация успешна", "Закрыть", {dura-
tion:3000});
        this. router. navigateByUrl("/login");
      } else {
        this. snackbar. open ("Регистрация не прошла. Попробуй еще раз", "За-
крыть», {duration:3000, panelClass:"error-snackbar"})
      }
    })

  }

}
```

Файл: signup. component. spec.ts

Содержимое файла:

```
import {ComponentFixture, TestBed} from '@angular/core/testing';
```

```
import {SignupComponent} from './signup.component';
describe ('SignupComponent', () => {
  let component: SignupComponent;
  let fixture: ComponentFixture<SignupComponent>;
  beforeEach (async () => {
    await TestBed.configureTestingModule({
      declarations: [SignupComponent]
    })
    . compileComponents ();
        fixture = TestBed.createComponent(SignupComponent);
    component = fixture. componentInstance;
    fixture. detectChanges ();
  });
  it ('should create', () => {
    expect(component). toBeTruthy ();
  });
});
```

Файл: signup.component.html
Содержимое файла:

```
<div class = "container">
    <mat-card>
        <mat-card-title>Регистрация</mat-card-title>

        <mat-card-content>
            <form [formGroup]="signupForm">

              <mat-form-field appearance="outline">
                <mat-label>Name</mat-label>
                <input matInput formControlName="name" placeholder="Имя" re-
quired>
                <mat-error *ngIf="signupForm.get('name').invalid && signup-
Form.get('name').touched">
                 Введите имя.
                </mat-error>
            </mat-form-field>
            <mat-form-field appearance="outline">
                <mat-label>Email</mat-label>
                <input matInput formControlName="email" placeholder="email"
required>
                <mat-error *ngIf="signupForm.get('email').invalid && signup-
Form.get('email').touched">
                Введите правильный email.
                </mat-error>
                </mat-form-field>
                <mat-form-field appearance="outline">
                    <mat-label>Password</mat-label>
                    <input matInput type="{{ hidePassword ? 'password':
'text'}}" formControlName="password"
                    placeholder="Пароль" required>

                     <button mat-icon-button matSuffix
(click)="togglePasswordVisibility()">
                        <mat-icon>{{hidePassword ? 'visibility_off' :
'visibility'}} </mat-icon>
                     </button>
                    <mat-error *ngIf="signupForm.get('password').invalid &&
signupForm.get('password').touched">
                        Введите пароль.
```

```
                    </mat-error>
                </mat-form-field>

                <mat-form-field appearance="outline">
                    <mat-label>Confirm Password</mat-label>
                <input matInput type ="{{ hidePassword ? 'password' :
'text'}}" formControlName="confirmPassword"
                    placeholder="пароль" required>

                <button mat-icon-button matSuffix
(click)="togglePasswordVisibility()">
                    <mat-icon>{{hidePassword ? 'visibility_off':
'visibility'}}</mat-icon>
                </button>
                <mat-error
                    *ngIf="signupForm.get('confirmPassword').invalid && sig-
nupForm.get('confirmPassword').touched">
                    Повторите пароль для регистрации.
                </mat-error>
                </mat-form-field>

                <button mat-raised-button  color="primary" type="submit"
[disabled]="signupForm.invalid"
                (click)="onSubmit()">Регистрация</button>
                 </form>
        </mat-card-content>
    </mat-card>
</div>
```

Файл: signup.component.scss

Содержимое файла:

```
.container{
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
}
mat-card{
    max-width: 400px;
    width: 100%;
    padding: 20 px;

    @media screen and (max-width: 500px){
        max-width: 90%;
    }

    mat-cart-title{
        text-align: center;
        font-size: 24px;
        margin-bottom: 20px;
    }

    form{
        display: flex;
        flex-direction: column;
        align-items: center;
    }
    mat-form-field {
        width: 100%;
        margin-bottom: 20px;
```

```
    }
    button{
        width: 100%;
    }
    @media screen and (max-width:500px){
        mat-card-title{
            font-size:20px;
        }
        mat-form-filed {
            margin-bottom:10px;
        }

    }
.error-snackbar{
background-color: #f44336 !important;
}
}
```

Файл: login.component.spec.ts

Содержимое файла:

```
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { LoginComponent } from './login.component';
describe('LoginComponent', () => {
  let component: LoginComponent;
  let fixture: ComponentFixture<LoginComponent>;

  beforeEach(async () => {
    await TestBed.configureTestingModule({
      declarations: [LoginComponent]
    })
    .compileComponents();

    fixture = TestBed.createComponent(LoginComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });
});
```

Файл: login.component.ts

Содержимое файла:

```
import { Component } from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { AuthService } from '../../services/auth/auth.service';
import { MatSnackBar } from '@angular/material/snack-bar';
import { StorageService } from '../../services/storage/storage.service';
import { Router } from '@angular/router';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrl: './login.component.scss'
})
export class LoginComponent {
  loginForm!: FormGroup;
  hidePassword = true;
  constructor(private fb: FormBuilder,
    private authService:AuthService,
```

```
    private snackbar:MatSnackBar,
    private router:Router
  ){
    this.loginForm = this.fb.group({
      email:[null,[Validators.required, Validators.email]],
      password:[null,[Validators.required]],
    })
  }
  togglePasswordVisibility(){
    this.hidePassword = !this.hidePassword
  }
  onSubmit() {
    console.log(this.loginForm.value);
    this.authService.login(this.loginForm.value).subscribe(
      (res) => {
        console.log(res);
        if (res.userId != null) {
          const user = {
            id: res.userId,
            role: res.userRole,
          };
          StorageService.saveUser(user);
          StorageService.saveToken(res.jwt);
          if (StorageService.isAdminLoggedIn()) {
            this.router.navigateByUrl("/admin/dashboard");
          } else if (StorageService.isEmployeeLoggedIn()) {
            this.router.navigateByUrl("/employee/dashboard");
          }
          this.snackbar.open("Успешный вход", "Закрыть", { duration: 3000 });
        } else {
          this.snackbar.open("Некорректный вход. Попробуй еще раз", "За-
крыть", {
            duration: 3000,
            panelClass: "error-snackbar",
          });
        }
      },
      (error) => {
        if (error.status === 403) {
          this.snackbar.open("Неправильный логин или пароль", "Закрыть", {
            duration: 3000,
            panelClass: "error-snackbar",
          });
        } else {
          this.snackbar.open("Произошла ошибка. Попробуйте снова позже", "За-
крыть", {
            duration: 3000,
            panelClass: "error-snackbar",
          });
        }
      }
    );
  }

  }
```

Файл: login.component.scss

Содержимое файла:

```
.container{
    display: flex;
```

```
    justify-content: center;
    align-items: center;
    height: 90vh;
}
mat-card {
    max-width: 400px;
    width: 100%;
    padding: 20px;
    @media screen and (max-width: 500px){
        max-width: 90%;
    }
}
mat-card-title{
    text-align: center;
    font-size: 24px;
    margin-bottom: 20px;
}
form{
    display: flex;
    flex-direction: column;
    align-items: center;
}
mat-form-field{
    width: 100%;
    margin-bottom: 20px;
}
@media screen and (max-width: 500px){
    mat-card title{
        font-size: 20 px;
    }
    mat-form-field{
        margin-bottom: 10 px;
    }
}

.error-snackbar{
    background-color: #f44336 !important;
}
```

Файл: login.component.html
Содержимое файла:

```
<div class="container">
    <mat-card>
        <mat-card-title>Вход</mat-card-title>
        <mat-card-content>
            <form [formGroup]="loginForm">

        <mat-form-field appearance="outline">
                <mat-label>E-mail</mat-label>
                <input matInput formControlName="email" placeholder="Email"
required>
                <mat-error *ngIf="loginForm.get('email')?.invalid && login-
Form.get('email')?.touched">
                    Введите свой e-mail адрес
                </mat-error>
        </mat-form-field>
                <mat-form-field appearance="outline">
            <mat-label>Пароль</mat-label>
```

```
            <input matInput type ="{{ hidePassword ? 'password' : 'text'}}"
formControlName="password"
            placeholder="Пароль" required>
            <button mat-icon-button matSuffix
(click)="togglePasswordVisibility()">
                <mat-icon> {{ hidePassword ? 'visibility_off':
'visibility'}}</mat-icon>
            </button>
            <mat-error *ngIf="loginForm.get('password')?.invalid && login-
Form.get('password')?.touched">
                Введите пароль
            </mat-error>
        </mat-form-field>
        <button mat-raised-button color="primary" type="submit" [dis-
abled]="loginForm.invalid"
        (click)="onSubmit()">Вход</button>
    </form>
 </mat-card-content>
 <mat-card-actions>
    <button mat-button color="accent" routerLink="/signup">Нет аккаунта? За-
регистрируйся!</button>
 </mat-card-actions>
</mat-card>
</div>
```

Файл: auth.service.ts

Содержимое файла:

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
const BASE_URL = "http://localhost:8080/";
@Injectable({
  providedIn: 'root'
})
export class AuthService {
  constructor(private http: HttpClient) { }
  signup(signupRequest: any): Observable<any> {
   return this.http.post(BASE_URL+"api/auth/signup", signupRequest)

  }

login(loginRequest:any):Observable<any>{
 return this.http.post(BASE_URL+"api/auth/login",loginRequest)
}

}
```

Файл: auth.service.spec.ts

Содержимое файла:

```
import { TestBed } from '@angular/core/testing';
import { AuthService } from './auth.service';
describe('AuthService', () => {
  let service: AuthService;
  beforeEach(() => {
    TestBed.configureTestingModule({});
    service = TestBed.inject(AuthService);
  });
  it('should be created', () => {
    expect(service).toBeTruthy();
  });
});
```

Файл: storage.service.spec.ts
Содержимое файла:

```
import { TestBed } from '@angular/core/testing';
import { StorageService } from './storage.service';
describe('StorageService', () => {
  let service: StorageService;
  beforeEach(() => {
    TestBed.configureTestingModule({});
    service = TestBed.inject(StorageService);
  });
  it('should be created', () => {
    expect(service).toBeTruthy();
  });
});
```

Файл: storage.service.ts
Содержимое файла:

```
import { Injectable } from '@angular/core';

const TOKEN = "token";
const USER = "user";

@Injectable({
  providedIn: 'root'
})
export class StorageService {
  constructor() { }
  static saveToken(token:string):void{
    window.localStorage.removeItem(TOKEN);
    window.localStorage.setItem(TOKEN, token);
  }
    static saveUser(user:any):void{
    window.localStorage.removeItem(USER);
    window.localStorage.setItem(USER, JSON.stringify(user));
  }

  static getToken(): string{
    return localStorage.getItem(TOKEN)
  } // его метод который ругается

  static getUser(): any{
    return JSON.parse(localStorage.getItem(USER));
  }// его метод который ругается

  static getUserRole():string{
  const user = this.getUser();
  if (user == null)
    return '';
   return user.role;
  }
    static isAdminLoggedIn(): boolean{
    if(this.getToken() === null)
      return false;
    const role: string = this.getUserRole();
    return role == "ADMIN";
  }
    static isEmployeeLoggedIn(): boolean{
    if(this.getToken() === null)
      return false;
    const role: string = this.getUserRole();
```

```
    return role == "EMPLOYEE";
  }
    static getUserId(): string{
    const user = this.getUser();
    if (user == null)
      return "";
    return user.id;
  }
    static logout():void{
    window.localStorage.removeItem(TOKEN);
    window.localStorage.removeItem(USER);
  }

}
```
Файл: admin.module.ts
Содержимое файла:
```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { AdminRoutingModule } from './admin-routing.module';
import { DashboardComponent } from
'./components/dashboard/dashboard.component';
import { PostTaskComponent } from './components/post-task/post-
task.component';
import { FormsModule, ReactiveFormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/common/http';
import { DemoAngularMaterialModule } from '../../DemoAngularMaterialModule';
import { UpdateTaskComponent } from './components/update-task/update-
task.component';
import { ViewTaskDetailsComponent } from './components/view-task-
details/view-task-details.component';
@NgModule({
  declarations: [
    DashboardComponent,
    PostTaskComponent,
    UpdateTaskComponent,
    ViewTaskDetailsComponent,
  ],
  imports: [
    CommonModule,
    AdminRoutingModule,
    FormsModule,
    ReactiveFormsModule, HttpClientModule,
    DemoAngularMaterialModule
      ]
})
export class AdminModule { }
```
Файл: admin-routing.module.ts
Содержимое файла:
```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { DashboardComponent } from
'./components/dashboard/dashboard.component';
import { PostTaskComponent } from './components/post-task/post-
task.component';
import { UpdateTaskComponent } from './components/update-task/update-
task.component';
import { ViewTaskDetailsComponent } from './components/view-task-
details/view-task-details.component';
```

```
const routes: Routes = [
  {path:"dashboard",component:DashboardComponent },
  {path:"task",component:PostTaskComponent },
  {path:"task/:id/edit",component:UpdateTaskComponent },
  {path:"task-details/:id",component:ViewTaskDetailsComponent },
];

@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})
export class AdminRoutingModule { }
```
Файл: dashboard.component.spec.ts
Содержимое файла:
```
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { DashboardComponent } from './dashboard.component';
describe('DashboardComponent', () => {
  let component: DashboardComponent;
  let fixture: ComponentFixture<DashboardComponent>;
  beforeEach(async () => {
    await TestBed.configureTestingModule({
      declarations: [DashboardComponent]
    })
    .compileComponents();
        fixture = TestBed.createComponent(DashboardComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });
});
```
Файл: dashboard.component.html
Содержимое файла:
```
<form class="search-form" [formGroup]="searchForm">
    <mat-form-field appearance="outline" class="search-form">
        <mat-label>Название для поиска</mat-label>
        <input matInput formControlName="title" (input)="searchTask()">
    </mat-form-field>
</form>


<div class = "task-list">
    <mat-card class="task-card" *ngFor="let task of listOfTasks">
        <div class="task-details">
            <h2 class="task-name">{{ task.title }}</h2>
            <p class="task-description">{{ task.description }}</p>

            <mat-divider></mat-divider>
            <div class="task-info">
                <div class="info-row">
                    <span class="info-label">Выполнить до </span>
                    <span class="info-value">{{ task.dueDate | date:'dd MMMM,
yyyy'}}</span>
                </div>
                <div class="info-row">
                    <span class="info-label">Исполнитель:</span>
                    <span class="info-value">{{ task.employeeName }}</span>
```

```
                </div>
                <div class="info-row">
                    <span class="info-label">Приоритет</span>
                    <span class="info-value">{{ task.priority }}</span>
                </div>
                <div class="info-row">
                    <span class="info-label">Статус</span>
                    <span class="info-value">{{ task.taskStatus }}</span>
                </div>
            </div>
            <mat-divider></mat-divider>
            <div class="actions">
                <button mat-icon-button color="accent" class="view-button"
routerLink="/admin/task-details/{{task.id}}">
                    <mat-icon>visibility</mat-icon>
                </button>
                <a mat-icon-button color="primary" class="update-button"
                [routerLink]="'/admin/task/' + task.id + '/edit'">
                <mat-icon>edit</mat-icon>
                </a>
                <button mat-icon-button color="warn" class="delete-
button"(click)="deleteTask(task.id)">
                    <mat-icon>delete</mat-icon>
                </button>
            </div>
        </div>
    </mat-card>
</div>
```

Файл: dashboard.component.scss
Содержимое файла:

```
.search-form {
    display: flex;
    align-items: center;
    justify-content: center;
    margin-top: 20px;

    .search-input{
        width: 300px;
        margin-right: 15px;

        input{
            padding-right: 40px;
        }
    }
}
.task-list {
    display: flex;
    flex-wrap: wrap;
    justify-content: space-between;
    background-color: #EFE2BA;
    min-height: 37vh;
    padding: 20px;

    .task-card {
        width: calc(33.33% - 20px);
        margin-bottom: 20px;
        border-radius: 10px;
```

```
box-shadow: 0 2px 6px #C3073F;
background-color: #41B3A3;

.task-details {
    padding: 20px;

    .task-name {
        font-size: 20px;
        font-weight: bold;
        color: #0B0c10;
        margin-bottom: 10px;
    }

    .task-description {
        margin-bottom: 10px;
        color: #0B0c10;
    }

    .task-info {
        display: flex;
        flex-wrap: wrap;
        margin-bottom: 10px;
        margin-top: 10px;
        color: #0B0c10;

        .info-row {
            flex: 0 0 50%;
            display: flex;
            align-items: center;
            margin-bottom: 8px;

            .info-label {
                font-weight: 10px;
                margin-right: 5px;
            }

            .info-value {
                font-weight: bold;
            }
        }
    }

    .actions {
        display: flex;
        justify-content: flex-end;
        margin-top: 10px;

        .view-button {
            margin-right: 10px;
        }

        .update-button {
            margin-right: 10px;
        }

        .view-button {
            margin-right: 10px;
        }
    }
```

```
        }
    }
}
```

Файл: dashboard.component.ts
Содержимое файла:

```
import { Component } from '@angular/core';
import { AdminService } from '../../services/admin.service';
import { MatSnackBar } from '@angular/material/snack-bar';
import { FormBuilder, FormGroup } from '@angular/forms';

@Component({
  selector: 'app-dashboard',
  templateUrl: './dashboard.component.html',
  styleUrl: './dashboard.component.scss'
})
export class DashboardComponent {

  listOfTasks : any = [];
  searchForm!: FormGroup

  constructor(private service:AdminService,
    private snackbar:MatSnackBar,
    private fb:FormBuilder
  ) {
    this.getTasks();
    this.searchForm = this.fb.group({
      title: [null]
    })
  }

  getTasks(){
    this.service.getAllTasks().subscribe((res)=>{
      this.listOfTasks = res;
    })
  }

  deleteTask(id:number){
    this.service.deleteTask(id).subscribe((res)=>{
      this.snackbar.open("Задача успешна удалена", "Закрыть", {dura-
tion:3000});
      this.getTasks();
    })
  }
  searchTask(){
    this.listOfTasks = [];
    const title = this.searchForm.get('title')!.value;
    console.log(title);
    this.service.searchTask(title).subscribe((res)=>{
      console.log(res);
      this.listOfTasks = res;
    })
  }
}
```

Файл: view-task-details.component.scss
Содержимое файла:

```
.task-list {
    display: flex;
    flex-wrap: wrap;
```

```
justify-content: space-between;
// background-color: #EFE2BA;
min-height: 37vh;

.task-card {
  width: calc(33.33% - 20px);
  margin-bottom: 20px;
  width: 100%;
  border-radius: 10px;
  box-shadow: 0 2px 6px #C3073F;
  background-color: #41B3A3;

  .task-details {
    padding: 20px;

    .task-name {
      font-size: 20px;
      font-weight: bold;
      color: #0B0c10;
      margin-bottom: 10px;
    }

    .task-description {
      color: #0B0c10;
      margin-bottom: 10px;
    }

    .task-info {
      color: #0B0c10;
      display: flex;
      flex-wrap: wrap;
      margin-bottom: 10px;
      margin-top: 10px;

      .info-row {
        flex: 0 0 50%;
        display: flex;
        align-items: center;
        margin-bottom: 8px;

        .info-label {
          font-weight: 10px;
          margin-right: 5px;
        }
          .info-value {
          font-weight: bold;
        }
      }
    }
      .actions {
      display: flex;
      justify-content: flex-end;
      margin-top: 10px;

      .update-button {
        margin-right: 10px;
      }
        .view-button {
        margin-right: 10px;
```

```
          }
        }
      }
    }
  }
    .card-container {
    margin: 16px;
    max-width: 35%;
    padding-bottom: 30px;
  }

  mat-card {
    background-color: #41B3A3;
    padding: 10px;
  }

  .example-header-image {
    background-image: url('/assets/img/admin.jpg');
    background-size: cover;
  }
    mat-card-content {
    margin-top: 10px;
  }

  mat-card-footer {
    margin-left: 20px;
  }
    .img {

    object-fit: cover;
    }
    .comment-card {
    margin-top: 20px;
  }

  mat-form-field {
    width: 100%;
  }
```

Файл: view-task-details.component.html
Содержимое файла:

```
<div *ngIf="taskData" class="card-container">

<div class = "task-list" *ngIf="taskData">
    <mat-card class="task-card">
        <div class="task-details">
            <h2 class="task-name">{{ taskData.title }}</h2>
            <p class="task-description">{{ taskData.description }}</p>

            <mat-divider></mat-divider>
            <div class="task-info">
                <div class="info-row">
                    <span class="info-label">Выполнить до </span>
                    <span class="info-value">{{ taskData.dueDate | date:'dd
MMMM, yyyy'}}</span>
                </div>
                <div class="info-row">
                    <span class="info-label">Исполнитель:</span>
```

```
                    <span class="info-value">{{ taskData.employeeName
}}</span>
                </div>
                <div class="info-row">
                    <span class="info-label">Приоритет</span>
                    <span class="info-value">{{ taskData.priority }}</span>
                </div>
                <div class="info-row">
                    <span class="info-label">Статус</span>
                    <span class="info-value">{{ taskData.taskStatus }}</span>
                </div>
            </div>
        </div>
    </mat-card>
</div>
<mat-card class="comment-card">
    <mat-card-content>
        <h4>Напиши комментарии по задаче</h4>
        <div>
            <form [formGroup]="commentForm">
                <mat-form-field appearance="fill">
                    <mat-label>комментарий</mat-label>
                    <textarea matInput formControlName="content"></textarea>
                </mat-form-field>
                <button mat-raised-button color="primary" style="float:
right;" [disabled]="!commentForm.valid"
                    (click)="publishComment()">Написать комментарии</button>
            </form>
        </div>
    </mat-card-content>
</mat-card>

<div style="margin-top: 10px;">
    <h4>Комментарии</h4>
    <mat-card *ngFor="let comment of comments" style="margin-top: 10px;">
        <mat-card-header>
            <div mat-card-avatar class="example-header-image"></div>
            <mat-card-title>{{ comment.postedBy }}</mat-card-title>
            <mat-card-subtitle>{{ comment.createdAt | date:'medium' }}</mat-
card-subtitle>
        </mat-card-header>
        <mat-card-content>
            {{ comment.content }}
        </mat-card-content>
    </mat-card>
</div>

</div>
Файл: view-task-details.component.ts
Содержимое файла:
import { Component } from '@angular/core';
import { AdminService } from '../../services/admin.service';
import { ActivatedRoute } from '@angular/router';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { MatSnackBar } from '@angular/material/snack-bar';

@Component({
  selector: 'app-view-task-details',
  templateUrl: './view-task-details.component.html',
```

```
  styleUrl: './view-task-details.component.scss'
})
export class ViewTaskDetailsComponent {

  taskId:number= this.activatedRoute.snapshot.params["id"];
  taskData:any;
  comments:any;
  commentForm!: FormGroup;

  constructor(private service:AdminService,
    private activatedRoute: ActivatedRoute,
    private fb: FormBuilder,
    private snackbar:MatSnackBar
  ){ }

  ngOnInit(){
    this.getTaskById();
    this.getComments();
    this.commentForm = this.fb.group({
      content:[null, Validators.required]
    })
  }
  getTaskById(){
    this.service.getTaskById(this.taskId).subscribe((res)=>{
      this.taskData = res;
    })
  }

  getComments(){
    this.service.getCommentsByTask(this.taskId).subscribe((res)=>{
      this.comments = res;
    })

  }

  publishComment(){
    this.service.createComment(this.taskId,
this.commentForm.get("content")?.value).subscribe((res)=>{
      if(res.id !=null){
          this.snackbar.open("Комментарий успешно добавлен","Закрыть", {dura-
tion:3000});
          this.getComments();
      } else{
        this.snackbar.open("Комментарий не был добавлен, Ошибка!","Закрыть",
{duration:3000});
      }
    })
  }
}
```

Файл: view-task-details.component.spec.ts

Содержимое файла:

```
import { ComponentFixture, TestBed } from '@angular/core/testing';

import { ViewTaskDetailsComponent } from './view-task-details.component';

describe('ViewTaskDetailsComponent', () => {
  let component: ViewTaskDetailsComponent;
  let fixture: ComponentFixture<ViewTaskDetailsComponent>;
```

```
  beforeEach(async () => {
    await TestBed.configureTestingModule({
      declarations: [ViewTaskDetailsComponent]
    })
    .compileComponents();

    fixture = TestBed.createComponent(ViewTaskDetailsComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });
});
```

Файл: post-task.component.html

Содержимое файла:

```
<div class="container">
    <h2>Разместить задачу</h2>

    <form [formGroup]="taskForm">

        <mat-form-field apperance="outline" class="form-field">
            <mat-label>Заголовок</mat-label>
            <input matInput type="text" placeholder="Заголовок" formControl-
Name="title"/>
            <mat-error *ngIf="taskForm.get('title')?.hasError('required')">
                Поле должно быть заполнено!</mat-error>
        </mat-form-field>
        <mat-form-field apperance="outline" class="form-field">
            <mat-label>Описание</mat-label>
            <textarea matInput placeholder="Описание" formControl-
Name="description"></textarea>
            <mat-error
*ngIf="taskForm.get('description')?.hasError('required')">
                Поле должно быть заполнено!</mat-error>
        </mat-form-field>

        <mat-form-field apperance="outline" class="form-field">
            <mat-label>Срок исполнения</mat-label>
            <input matInput [matDatepicker]="dueDatePicker"
placeholder="Выберите срок исполнения" formControlName="dueDate">
            <mat-datepicker-toggle matSuffix [for]="dueDatePicker"></mat-
datepicker-toggle>
            <mat-datepicker #dueDatePicker></mat-datepicker>
            <mat-error *ngIf="taskForm.get('dueDate')?.hasError('required')">
                This Input Is Required!</mat-error>
        </mat-form-field>

        <mat-form-field apperance="outline" class="form-field">
            <mat-label>Выбрать приоритет</mat-label>
            <mat-select formControlName="priority" name="Priority">
                <mat-option *ngFor="let priority of listOfPriorities" [val-
ue]="priority">
                    {{priority}}
                </mat-option>
            </mat-select>
            <mat-error
*ngIf="taskForm.get('priority')?.hasError('required')">
```

```
                Поле должно быть заполнено!</mat-error>
        </mat-form-field>

        <mat-form-field apperance="outline" class="form-field">
            <mat-label>Выбрать исполнителя</mat-label>
            <mat-select formControlName="employeeId" name="employeeId">
                <mat-option *ngFor="let employee of listOfEmployees" [val-
ue]="employee.id">
                    {{employee.name}}
                </mat-option>
            </mat-select>
            <mat-error
*ngIf="taskForm.get('employeeId')?.hasError('required')">
                Поле должно быть заполнено!</mat-error>
        </mat-form-field>
        <button mat-raised-button color="primary" type="submit" [dis-
abled]="taskForm.invalid"
            (click)="postTask()">Разместить задачу</button>
    </form>
</div>
```

Файл: post-task.component.scss

Содержимое файла:

```
.container {
    max-width: 600px;
    margin: 0 auto;
    padding: 20px;

    h2 {
        text-align: center;
        margin-bottom: 20px;
        color: #3f51b5;
    }

    form {
        display: flex;
        flex-direction: column;

        mat-form-field {
            margin-top: 20px;
        }

        button {
            align-self: center;
            margin-top: 16px;
        }
    }
}

@media (max-width: 600px) {
    .container {
        padding: 10px;

        form {
            mat-form-field {
                margin-bottom: 10px;
            }
        }
    }
}
```

Файл: post-task.component.ts
Содержимое файла:

```
import { Component } from '@angular/core';
import { AdminService } from '../../services/admin.service';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { MatSnackBar } from '@angular/material/snack-bar';
import { Router } from '@angular/router';
@Component({
  selector: 'app-post-task',
  templateUrl: './post-task.component.html',
  styleUrl: './post-task.component.scss'
})
export class PostTaskComponent {
  taskForm!: FormGroup;
  listOfEmployees: any=[];
  listOfPriorities: any=["LOW","MEDIUM", "HIGH"];
  constructor(private adminService: AdminService,
    private fb: FormBuilder,
    private snackBar:MatSnackBar,
    private router: Router
  ){
    this.getUsers();
    this.taskForm = this.fb.group({
      employeeId:[null,[Validators.required]],
      title:[null,[Validators.required]],
      description:[null,[Validators.required]],
      dueDate:[null,[Validators.required]],
      priority:[null,[Validators.required]],

    })
  }
  getUsers(){
    this.adminService.getUsers().subscribe((res)=>{
      this.listOfEmployees = res;
      console.log(res);
    })
  }
  postTask(){
    this.adminService.postTask(this.taskForm.value).subscribe((res)=>{
      if(res.id !=null){
          this.snackBar.open("Задача опубликована","Close",{duration:3000});
          this.router.navigateByUrl("/admin/dashboard");
      } else {
        this.snackBar.open("Что-то пошло не так","Ошибка",{duration:3000});
      }
    })
  }
}
```

Файл: post-task.component.spec.ts
Содержимое файла:

```
import { ComponentFixture, TestBed } from '@angular/core/testing';

import { PostTaskComponent } from './post-task.component';

describe('PostTaskComponent', () => {
  let component: PostTaskComponent;
  let fixture: ComponentFixture<PostTaskComponent>;
  beforeEach(async () => {
    await TestBed.configureTestingModule({
```

```
      declarations: [PostTaskComponent]
    })
    .compileComponents();
        fixture = TestBed.createComponent(PostTaskComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });
});
```

Файл: update-task.component.spec.ts

Содержимое файла:

```
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { UpdateTaskComponent } from './update-task.component';
describe('UpdateTaskComponent', () => {
  let component: UpdateTaskComponent;
  let fixture: ComponentFixture<UpdateTaskComponent>;
  beforeEach(async () => {
    await TestBed.configureTestingModule({
      declarations: [UpdateTaskComponent]
    })
    .compileComponents();
        fixture = TestBed.createComponent(UpdateTaskComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });
  it('should create', () => {
    expect(component).toBeTruthy();
  });
});
```

Файл: update-task.component.scss

Содержимое файла:

```
.container {
    max-width: 600px;
    margin: 0 auto;
    padding: 20px;

    h2 {
        text-align: center;
        margin-bottom: 20px;
        color: #3f51b5;
    }

    form {
        display: flex;
        flex-direction: column;

        mat-form-field {
            margin-top: 20px;
        }
        button {
            align-self: center;
            margin-top: 16px;
        }
    }
}
@media (max-width: 600px) {
```

```
    .container {
        padding: 10px;

        form {
            mat-form-field {
                margin-bottom: 10px;
            }
        }
    }
}
```

Файл: update-task.component.ts

Содержимое файла:

```
import { Component } from '@angular/core';
import { AdminService } from '../../services/admin.service';
import { ActivatedRoute, Router } from '@angular/router';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { MatSnackBar } from '@angular/material/snack-bar';

@Component({
  selector: 'app-update-task',
  templateUrl: './update-task.component.html',
  styleUrl: './update-task.component.scss'
})
export class UpdateTaskComponent {

  id:number = this.route.snapshot.params["id"];
  updateTaskForm!: FormGroup;
  listOfEmployees: any=[];
  listOfPriorities: any=["LOW","MEDIUM", "HIGH"];
  listOfTaskStatus:
any=["PENDING","INPROGRESS","COMPLETED","DEFERRED","CANCELLED"];
  constructor(private service:AdminService,
    private route: ActivatedRoute,
    private fb: FormBuilder,
    private adminService: AdminService,
    private snackBar: MatSnackBar,
    private router: Router
  ){
    this.getTaskById();
    this.getUsers();
    this.updateTaskForm = this.fb.group({
      employeeId:[null,[Validators.required]],
      title:[null,[Validators.required]],
      description:[null,[Validators.required]],
      dueDate:[null,[Validators.required]],
      priority:[null,[Validators.required]],
      taskStatus:[null,[Validators.required]],
    })
  }
  getTaskById(){
    this.service.getTaskById(this.id).subscribe((res)=>{
      this.updateTaskForm.patchValue(res);
      console.log(res);
    })
  }
  getUsers(){
    this.adminService.getUsers().subscribe((res)=>{
      this.listOfEmployees = res;
      console.log(res);
```

```
    })
  }
  updateTask(){
    this.adminService.updateTask(this.id,
this.updateTaskForm.value).subscribe((res)=>{
      if(res.id !=null){
          this.snackBar.open("Задача успешна
обновлена","Close",{duration:3000});
          this.router.navigateByUrl("/admin/dashboard");
      } else {
        this.snackBar.open("Что-то пошло не так","Ошибка",{duration:3000});
      }
    })
  }

}
```

Файл: update-task.component.html

Содержимое файла:

```
<div class="container">
    <h2>Обновить задачу</h2>

    <form [formGroup]="updateTaskForm">

        <mat-form-field apperance="outline" class="form-field">
            <mat-label>Заголовок</mat-label>
            <input matInput type="text" placeholder="Заголовок" formControl-
Name="title"/>
            <mat-error
*ngIf="updateTaskForm.get('title')?.hasError('required')">
                Поле должно быть заполнено!</mat-error>
        </mat-form-field>
        <mat-form-field apperance="outline" class="form-field">
            <mat-label>Описание</mat-label>
            <textarea matInput placeholder="Описание" formControl-
Name="description"></textarea>
            <mat-error
*ngIf="updateTaskForm.get('description')?.hasError('required')">
                Поле должно быть заполнено!</mat-error>
        </mat-form-field>

        <mat-form-field apperance="outline" class="form-field">
            <mat-label>Срок исполнения</mat-label>
            <input matInput [matDatepicker]="dueDatePicker"
placeholder="Выберите срок исполнения" formControlName="dueDate">
            <mat-datepicker-toggle matSuffix [for]="dueDatePicker"></mat-
datepicker-toggle>
            <mat-datepicker #dueDatePicker></mat-datepicker>
            <mat-error
*ngIf="updateTaskForm.get('dueDate')?.hasError('required')">
                This Input Is Required!</mat-error>
        </mat-form-field>

        <mat-form-field apperance="outline" class="form-field">
            <mat-label>Выбрать приоритет</mat-label>
            <mat-select formControlName="priority" name="Priority">
                <mat-option *ngFor="let priority of listOfPriorities" [val-
ue]="priority">
                    {{priority}}
                </mat-option>
```

```
            </mat-select>
            <mat-error
*ngIf="updateTaskForm.get('priority')?.hasError('required')">
                Поле должно быть заполнено!</mat-error>
        </mat-form-field>

        <mat-form-field apperance="outline" class="form-field">
            <mat-label>Выбрать статус задачи</mat-label>
            <mat-select formControlName="taskStatus" name="taskStatus">
                <mat-option *ngFor="let status of listOfTaskStatus" [val-
ue]="status">
                    {{status}}
                </mat-option>
            </mat-select>
            <mat-error
*ngIf="updateTaskForm.get('taskStatus')?.hasError('required')">
                Поле должно быть заполнено!</mat-error>
        </mat-form-field>

        <mat-form-field apperance="outline" class="form-field">
            <mat-label>Выбрать исполнителя</mat-label>
            <mat-select formControlName="employeeId" name="employeeId">
                <mat-option *ngFor="let employee of listOfEmployees" [val-
ue]="employee.id">
                    {{employee.name}}
                </mat-option>
            </mat-select>
            <mat-error
*ngIf="updateTaskForm.get('employeeId')?.hasError('required')">
                Поле должно быть заполнено!</mat-error>
        </mat-form-field>

        <button mat-raised-button color="primary" type="submit" [dis-
abled]="updateTaskForm.invalid"
            (click)="updateTask()">Обновить задачу</button>
    </form>
</div>
```

Файл: admin.service.spec.ts
Содержимое файла:

```
import { TestBed } from '@angular/core/testing';

import { AdminService } from './admin.service';

describe('AdminService', () => {
  let service: AdminService;

  beforeEach(() => {
    TestBed.configureTestingModule({});
    service = TestBed.inject(AdminService);
  });

  it('should be created', () => {
    expect(service).toBeTruthy();
  });
});
```

Файл: admin.service.ts
Содержимое файла:

```
import { HttpClient, HttpHeaders } from '@angular/common/http';
```

```
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { StorageService } from
'../../../auth/services/storage/storage.service';
const BASIC_URL = "http://localhost:8080/";
@Injectable({
  providedIn: 'root'
})
export class AdminService {

  constructor(private http:HttpClient) { }

  getUsers():Observable<any>{
   return this.http.get(BASIC_URL + "api/admin/users",{
      headers:this.createAuthorizationHeader()
    })
  }
  postTask(TaskDto: any):Observable<any>{
    return this.http.post(BASIC_URL + "api/admin/task", TaskDto, {
       headers:this.createAuthorizationHeader()

     })
   }
   deleteTask(id: number):Observable<any>{
    return this.http.delete(BASIC_URL + "api/admin/task/" + id, {
       headers:this.createAuthorizationHeader()
      })
   }
   getTaskById(id: number):Observable<any>{
    return this.http.get(BASIC_URL + "api/admin/task/" + id, {
       headers:this.createAuthorizationHeader()

     })
   }
   getAllTasks():Observable<any>{
    return this.http.get(BASIC_URL + "api/admin/tasks",{
       headers:this.createAuthorizationHeader()

     })
   }
   searchTask(title: string): Observable<any>{
    return this.http.get(BASIC_URL + `api/admin/tasks/search/${title}`, {
       headers:this.createAuthorizationHeader()
      })
   }
   updateTask(id: number, TaskDto: any): Observable<any>{
    return this.http.put(BASIC_URL + `api/admin/task/${id}`, TaskDto, {
       headers:this.createAuthorizationHeader()
      })
   }
   createComment(id: number, content: string):Observable<any>{
    const params = {
      content: content
    }
    return this.http.post(BASIC_URL + "api/admin/task/comment/" + id, null,{
      params: params,
      headers:this.createAuthorizationHeader()
      })
   }
```

```
   getCommentsByTask(id: number):Observable<any>{
    return this.http.get(BASIC_URL + "api/admin/comments/" + id, {
       headers:this.createAuthorizationHeader()

     })
   }

  private createAuthorizationHeader(): HttpHeaders {
    return new HttpHeaders().set(
      'Authorization', 'Bearer ' +  StorageService.getToken()
    )
  }
  }
```

Файл: employee-routing.module.ts
Содержимое файла:

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { DashboardComponent } from
'./components/dashboard/dashboard.component';
import { ViewTaskDetailsComponent } from './components/view-task-
details/view-task-details.component';

const routes: Routes = [
  {path:"dashboard",component:DashboardComponent},
  {path:"task-details/:id",component:ViewTaskDetailsComponent},
];

@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})
export class EmployeeRoutingModule { }
```

Файл: employee.module.ts
Содержимое файла:

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';

import { EmployeeRoutingModule } from './employee-routing.module';
import { DashboardComponent } from
'./components/dashboard/dashboard.component';
import { HttpClientModule } from '@angular/common/http';
import { FormsModule, ReactiveFormsModule } from '@angular/forms';
import { DemoAngularMaterialModule } from '../../DemoAngularMaterialModule';
import { ViewTaskDetailsComponent } from './components/view-task-
details/view-task-details.component';


@NgModule({
  declarations: [
    DashboardComponent,
    ViewTaskDetailsComponent
  ],
  imports: [
    CommonModule,
    EmployeeRoutingModule,
    HttpClientModule,
    ReactiveFormsModule,
```

```
    FormsModule,
    DemoAngularMaterialModule


  ]
})
export class EmployeeModule { }
```

Файл: dashboard.component.spec.ts
Содержимое файла:

```
import { ComponentFixture, TestBed } from '@angular/core/testing';

import { DashboardComponent } from './dashboard.component';

describe('DashboardComponent', () => {
  let component: DashboardComponent;
  let fixture: ComponentFixture<DashboardComponent>;

  beforeEach(async () => {
    await TestBed.configureTestingModule({
      declarations: [DashboardComponent]
    })
    .compileComponents();

    fixture = TestBed.createComponent(DashboardComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });
});
```

Файл: dashboard.component.html
Содержимое файла:

```
<div class="task-list">
    <mat-card class="task-card" *ngFor="let task of listOfTasks">
        <div class="task-details">
            <h2 class="task-name">{{ task.title }}</h2>
            <p class="task-description">{{ task.description }}</p>
            <mat-divider></mat-divider>
            <div class="task-info">
                <div class="info-row">
                    <span class="info-label">Выполнить до</span>
                    <span class="info-value">{{task.dueDate | date: 'dd MMMM,
yyyy'}}</span>
                </div>
                <div class="info-row">
                    <span class="info-label">Исполнитель</span>
                    <span class="info-value">{{task.employeeName}}</span>
                </div>
                <div class="info-row">
                    <span class="info-label">Приоритет</span>
                    <span class="info-value">{{task.priority}}</span>
                </div>
                <div class="info-row">
                    <span class="info-label">Статус</span>
                    <span class="info-value">{{task.taskStatus}}</span>
```

```
                </div>
            </div>
            <mat-divider></mat-divider>
            <div class="actions">
                <button mat-icon-button color="accent" class="view-button"
routerLink="/employee/task-details/{{task.id}}">
                    <mat-icon>visibility</mat-icon>
                </button>
                <button mat-icon-button color="primary" class="update-button"
[matMenuTriggerFor]="statusMenu">
                    <mat-icon>edit</mat-icon>
                </button>
                <mat-menu #statusMenu="matMenu">
                    <button mat-menu-item (click)="updateStatus(task.id,
'INPROGRESS')">INPROGRESS</button>
                    <button mat-menu-item (click)="updateStatus(task.id,
'COMPLETED')">COMPLETED</button>
                </mat-menu>
            </div>
        </div>
    </mat-card>
</div>
```

Файл: dashboard.component.scss
Содержимое файла:

```
.task-list {
    display: flex;
    flex-wrap: wrap;
    justify-content: space-between;
    background-color: #EFE2BA;
    min-height: 37vh;
    padding: 20px;

    .task-card {
        width: calc(33.33% - 20px);
        margin-bottom: 20px;
        border-radius: 10px;
        box-shadow: 0 2px 6px #F13C20;
        background-color: #41B3A3;

        .task-details {
            padding: 20px;

            .task-name {
                font-size: 20px;
                font-weight: bold;
                color: #0B0c10;
                margin-bottom: 10px;
            }

            .task-description {
                margin-bottom: 10px;
                color: #0B0c10;
            }

            .task-info {
                display: flex;
                flex-wrap: wrap;
                margin-bottom: 10px;
```

```
                margin-top: 10px;
                color: #0B0c10;

                .info-row {
                    flex: 0 0 50%;
                    display: flex;
                    align-items: center;
                    margin-bottom: 8px;

                    .info-label {
                        font-weight: 10px;
                        margin-right: 5px;
                    }

                    .info-value {
                        font-weight: bold;
                    }
                }
            }
            .actions {
                display: flex;
                justify-content: flex-end;
                margin-top: 10px;

                .update-button {
                    margin-right: 10px;
                }

                .view-button {
                    margin-right: 10px;
                }
            }

        }
    }
}
```

Файл: dashboard.component.ts
Содержимое файла:

```
import { Component } from '@angular/core';
import { EmployeeService } from '../../services/employee.service';
import { MatSnackBar } from '@angular/material/snack-bar';

@Component({
  selector: 'app-dashboard',
  templateUrl: './dashboard.component.html',
  styleUrl: './dashboard.component.scss'
})
export class DashboardComponent {

  listOfTasks: any = [];

  constructor(private service: EmployeeService,
    private snackbar: MatSnackBar
   ) {
    this.getTasks();
  }

  getTasks(){
```

```
    this.service.getEmployeeTasksById().subscribe((res)=>{
      console.log(res);
      this.listOfTasks = res;
    })
  }

  updateStatus(id: number, status: string){
    this.service.updateStatus(id,status).subscribe((res)=>{
      if (res.id!=null){
        this.snackbar.open("Статус задачи успешно обновлен",
"Close",{duration:3000})
        this.getTasks();
      }else{
        this.snackbar.open("Произошла ошибка при обновлении задачи",
"Close",{duration:3000})
      }
    })
  }

}
```

Файл: view-task-details.component.scss
Содержимое файла:

```
.task-list {
    display: flex;
    flex-wrap: wrap;
    justify-content: space-between;
    // background-color: #EFE2BA;
    min-height: 37vh;

    .task-card {
      width: calc(33.33% - 20px);
      margin-bottom: 20px;
      width: 100%;
      border-radius: 10px;
      box-shadow: 0 2px 6px #F13C20;
      background-color: #41B3A3;

      .task-details {
        padding: 20px;

        .task-name {
          font-size: 20px;
          font-weight: bold;
          color: #0B0c10;
          margin-bottom: 10px;
        }

        .task-description {
          color: #0B0c10;
          margin-bottom: 10px;
        }

        .task-info {
          color: #0B0c10;
          display: flex;
          flex-wrap: wrap;
          margin-bottom: 10px;
```

```
        margin-top: 10px;

        .info-row {
          flex: 0 0 50%;
          display: flex;
          align-items: center;
          margin-bottom: 8px;

          .info-label {
            font-weight: 10px;
            margin-right: 5px;
          }

          .info-value {
            font-weight: bold;
          }
        }
      }

      .actions {
        display: flex;
        justify-content: flex-end;
        margin-top: 10px;

        .update-button {
          margin-right: 10px;
        }

        .view-button {
          margin-right: 10px;
        }
      }
    }
  }
}


.card-container {
  margin: 16px;
  max-width: 35%;
  padding-bottom: 30px;

}

mat-card {
  background-color: #41B3A3;
  padding: 10px;
}

.example-header-image {
  background-image: url('/assets/img/employee.jpg');
  background-size: cover;
}

mat-card-content {

  margin-top: 10px;
}
```

```
  mat-card-footer {
    margin-left: 20px;
  }

  .img {
    object-fit: cover;

  }

  .comment-card {
    margin-top: 20px;
  }

  mat-form-field {
    width: 100%;
  }
```

Файл: view-task-details.component.html
Содержимое файла:

```
<div *ngIf="taskData" class="card-container">

    <div class = "task-list" *ngIf="taskData">
        <mat-card class="task-card">
            <div class="task-details">
                <h2 class="task-name">{{ taskData.title }}</h2>
                <p class="task-description">{{ taskData.description }}</p>

                <mat-divider></mat-divider>
                <div class="task-info">
                    <div class="info-row">
                        <span class="info-label">Выполнить до </span>
                        <span class="info-value">{{ taskData.dueDate |
date:'dd MMMM, yyyy'}}</span>
                    </div>
                    <div class="info-row">
                        <span class="info-label">Исполнитель:</span>
                        <span class="info-value">{{ taskData.employeeName
}}</span>
                    </div>
                    <div class="info-row">
                        <span class="info-label">Приоритет</span>
                        <span class="info-value">{{ taskData.priority
}}</span>
                    </div>
                    <div class="info-row">
                        <span class="info-label">Статус</span>
                        <span class="info-value">{{ taskData.taskStatus
}}</span>
                    </div>
                </div>
            </div>
        </mat-card>
    </div>

    <mat-card class="comment-card">
        <mat-card-content>
            <h4>Напишите комментарии по задаче</h4>
            <div>
                <form [formGroup]="commentForm">
```

```
                    <mat-form-field appearance="fill">
                        <mat-label>комментарий</mat-label>
                        <textarea matInput formControl-
Name="content"></textarea>
                    </mat-form-field>
                    <button mat-raised-button color="primary" style="float:
right;" [disabled]="!commentForm.valid"
                    (click)="publishComment()">Написать комментарии</button>
                </form>
            </div>
        </mat-card-content>
    </mat-card>

    <div style="margin-top: 10px;">
        <h4>Комментарии</h4>
        <mat-card *ngFor="let comment of comments" style="margin-top: 10px;">
            <mat-card-header>
                <div mat-card-avatar class="example-header-image"></div>
                <mat-card-title>{{ comment.postedBy }}</mat-card-title>
                <mat-card-subtitle>{{ comment.createdAt | date:'medium'
}}</mat-card-subtitle>
            </mat-card-header>
            <mat-card-content>
                {{ comment.content }}
            </mat-card-content>
        </mat-card>
    </div>

    </div>
```

Файл: view-task-details.component.ts
Содержимое файла:

```
import { Component } from '@angular/core';
import { EmployeeService } from '../../services/employee.service';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { MatSnackBar } from '@angular/material/snack-bar';
import { ActivatedRoute } from '@angular/router';

@Component({
  selector: 'app-view-task-details',
  templateUrl: './view-task-details.component.html',
  styleUrl: './view-task-details.component.scss'
})
export class ViewTaskDetailsComponent {
  taskId:number= this.activatedRoute.snapshot.params["id"];
  taskData:any;
  comments:any;
  commentForm!: FormGroup;

  constructor(private service: EmployeeService,
    private activatedRoute: ActivatedRoute,
    private fb: FormBuilder,
    private snackbar:MatSnackBar
  ){ }

  ngOnInit(){
    this.getTaskById();
    this.getComments();
    this.commentForm = this.fb.group({
```

```
      content:[null, Validators.required]
    })
  }

  getTaskById(){
    this.service.getTaskById(this.taskId).subscribe((res)=>{
      this.taskData = res;
    })
  }

  getComments(){
    this.service.getCommentsByTask(this.taskId).subscribe((res)=>{
      this.comments = res;
    })
  }
  publishComment(){
    this.service.createComment(this.taskId,
this.commentForm.get("content")?.value).subscribe((res)=>{
      if(res.id !=null){
          this.snackbar.open("Комментарий успешно добавлен","Закрыть", {dura-
tion:3000});
          this.getComments();
      } else{
        this.snackbar.open("Комментарий не был добавлен, Ошибка!","Закрыть",
{duration:3000});
      }
    })
  }
}
```

Файл: view-task-details.component.spec.ts

Содержимое файла:

```
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { ViewTaskDetailsComponent } from './view-task-details.component';
describe('ViewTaskDetailsComponent', () => {
  let component: ViewTaskDetailsComponent;
  let fixture: ComponentFixture<ViewTaskDetailsComponent>;
  beforeEach(async () => {
    await TestBed.configureTestingModule({
      declarations: [ViewTaskDetailsComponent]
    })
    .compileComponents();

    fixture = TestBed.createComponent(ViewTaskDetailsComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });
});
```

Файл: employee.service.spec.ts

Содержимое файла:

```
import { TestBed } from '@angular/core/testing';
import { EmployeeService } from './employee.service';
describe('EmployeeService', () => {
  let service: EmployeeService;
  beforeEach(() => {
    TestBed.configureTestingModule({});
```

```
    service = TestBed.inject(EmployeeService);
  });
  it('should be created', () => {
    expect(service).toBeTruthy();
  });
});
```

Файл: employee.service.ts

Содержимое файла:

```
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { StorageService } from
'../../../auth/services/storage/storage.service';
const BASIC_URL = "http://localhost:8080/";
@Injectable({
  providedIn: 'root'
})
export class EmployeeService {
  constructor(private http: HttpClient) { }
  getEmployeeTasksById():Observable<any>{
    return this.http.get(BASIC_URL + "api/employee/tasks",{
       headers:this.createAuthorizationHeader()
      })
   }
   updateStatus(id:number, status: string):Observable<any>{
    return this.http.get(BASIC_URL + `api/employee/task/${id}/${status}`,{
       headers:this.createAuthorizationHeader()

     })
   }
   getTaskById(id: number):Observable<any>{
    return this.http.get(BASIC_URL + "api/employee/task/" + id, {
       headers:this.createAuthorizationHeader()
      })
   }
   createComment(id: number, content: string):Observable<any>{
    const params = {
      content: content
    }
    return this.http.post(BASIC_URL + "api/employee/task/comment/" + id,
null,{
      params: params,
      headers:this.createAuthorizationHeader()
      })
   }
   getCommentsByTask(id: number):Observable<any>{
    return this.http.get(BASIC_URL + "api/employee/comments/" + id, {
       headers:this.createAuthorizationHeader()

     })
   }
    private createAuthorizationHeader(): HttpHeaders {
    return new HttpHeaders().set(
      'Authorization', 'Bearer ' +  StorageService.getToken()
    )
  }
}
```

A.2 Backend-часть

Файл: TaskFlowApplicationTests.java
Содержимое файла:

```
package com.primary.taskFlow;
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;
@SpringBootTest
class TaskFlowApplicationTests {
	@Test
	void contextLoads() {
	}
}
```

Файл: application.properties
Содержимое файла:

```
#server.port=8585
spring.application.name=taskFlow
spring.datasource.url=jdbc:mysql://localhost:3306/taskFlowProject
spring.datasource.username=root
spring.datasource.password=qwerty12345
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
```

Файл: TaskFlowApplication.java
Содержимое файла:

```
package com.primary.taskFlow;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class TaskFlowApplication {
	public static void main(String[] args) {
		SpringApplication.run(TaskFlowApplication.class, args);
	}

}
```

Файл: AuthenticationResponse.java
Содержимое файла:

```
package com.primary.taskFlow.dto;
import com.primary.taskFlow.enums.UserRole;
import lombok.Data;
@Data
public class AuthenticationResponse {
    private String jwt;
    private Long userId;
    private UserRole userRole;
}
```

Файл: SignupRequest.java
Содержимое файла:

```
package com.primary.taskFlow.dto;
import lombok.Data;
@Data
public class SignupRequest {
    private String name;
    private String email;
    private String password;
}
```

Файл: TaskDto.java
Содержимое файла:

```
package com.primary.taskFlow.dto;
import com.primary.taskFlow.enums.TaskStatus;
import lombok.Data;
import java.util.Date;
@Data
public class TaskDto {
    private Long id;
    private String title; //заголовок
    private String description; //описание
    private Date dueDate; //срок выполнения
    private String priority;// приоритет задачи
    private TaskStatus taskStatus; //статус задачи
    private Long employeeId; // идентификатор сотрудника
    private String employeeName; // имя сотрудника

}
```

Файл: AuthenticationRequest.java
Содержимое файла:

```
package com.primary.taskFlow.dto;
import lombok.Data;
@Data
public class AuthenticationRequest {
    private String email;
    private String password;

}
```

Файл: UserDto.java
Содержимое файла:

```
package com.primary.taskFlow.dto;
import com.primary.taskFlow.enums.UserRole;
import lombok.Data;
@Data
public class UserDto {
    private long id;
    private String name;
    private String email;
    private String password;
    private UserRole userRole;
}
```

Файл: CommentDto.java
Содержимое файла:

```
package com.primary.taskFlow.dto;
import lombok.Data;
import java.util.Date;
@Data
public class CommentDto {
    private Long id;
    private String content;
    private Date createdAt;
    private Long taskId;
    private Long userId;
    private String postedBy;
}
```

Файл: JwtAuthenticationFilter.java
Содержимое файла:

```
package com.primary.taskFlow.config;
import com.primary.taskFlow.services.jwt.UserService;
import com.primary.taskFlow.utils.JwtUtil;
import lombok.RequiredArgsConstructor;
```

```
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.apache.commons.lang3.StringUtils;
import org.springframework.lang.NonNull;
import
org.springframework.security.authentication.UsernamePasswordAuthenticationTok
en;
import org.springframework.security.core.context.SecurityContext;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import
org.springframework.security.web.authentication.WebAuthenticationDetailsSourc
e;
import java.io.IOException;
@Component
@RequiredArgsConstructor
public class JwtAuthenticationFilter extends OncePerRequestFilter {
    private final JwtUtil jwtUtil;
    private final UserService userService;
    @Override
    protected void doFilterInternal(@NonNull HttpServletRequest request,
                                    @NonNull HttpServletResponse response,
                                    @NonNull FilterChain filterChain) throws
ServletException, IOException {
        final String authHeader = request.getHeader("Authorization");
        final String jwt;
        final String userEmail;
        if (StringUtils.isEmpty(authHeader) || !StringU-
tils.startsWith(authHeader, "Bearer")) {
            filterChain.doFilter(request, response);
            return;
        }
        jwt = authHeader.substring(7);
        userEmail = jwtUtil.extractUserName(jwt);
        if (StringUtils.isNotEmpty(userEmail) && SecurityContextHold-
er.getContext().getAuthentication() == null) {
            UserDetails userDetails = userSer-
vice.userDetailService().loadUserByUsername(userEmail);
            if (jwtUtil.isTokenValid(jwt, userDetails)) {
                SecurityContext context = SecurityContextHold-
er.createEmptyContext();
                UsernamePasswordAuthenticationToken authToken = new Username-
PasswordAuthenticationToken(
                        userDetails, null, userDetails.getAuthorities());
                authToken.setDetails(new WebAuthenticationDetails-
Source().buildDetails(request));
                context.setAuthentication(authToken);
                SecurityContextHolder.setContext(context);
            }
        }
        filterChain.doFilter(request, response);
    }
}
```

Файл: WebSecurityConfiguration.java

Содержимое файла:

```
package com.primary.taskFlow.config;
import com.primary.taskFlow.enums.UserRole;
import com.primary.taskFlow.services.jwt.UserService;
import lombok.RequiredArgsConstructor;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.method.configuration.EnableMet
hodSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSec
urity;
import org.springframework.context.annotation.Bean;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.AuthenticationProvider;
import
org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import
org.springframework.security.config.annotation.authentication.configuration.A
uthenticationConfiguration;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configurers.AbstractHttpCo
nfigurer;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import
org.springframework.security.web.authentication.UsernamePasswordAuthenticatio
nFilter;
import static
org.springframework.security.config.http.SessionCreationPolicy.STATELESS;
@Configuration
@EnableWebSecurity
@EnableMethodSecurity
@RequiredArgsConstructor
public class WebSecurityConfiguration {
    private final JwtAuthenticationFilter jwtAuthenticationFilter;
    private final UserService userService;
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
Exception {
        http.csrf(AbstractHttpConfigurer::disable)
                        .authorizeHttpRequests(request-
>request.requestMatchers("/api/auth/**").permitAll()

.requestMatchers("/api/admin/**").hasAnyAuthority(UserRole.ADMIN.name())

.requestMatchers("/api/employee/**").hasAnyAuthority(UserRole.EMPLOYEE.name()
)

.anyRequest().authenticated()).sessionManagement(manager -> manag-
er.sessionCreationPolicy(STATELESS))

.authenticationProvider(authenticationProvider()).addFilterBefore(jwtAuthenti
cationFilter, UsernamePasswordAuthenticationFilter.class);
                return http.build();
    }
    @Bean
```

```
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
    @Bean
    public AuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider authProvider = new DaoAuthenticationProvid-
er();
        authProvider.setUserDetailsService(userService.userDetailService());
        authProvider.setPasswordEncoder(passwordEncoder());
        return authProvider;

    }
    @Bean
    public AuthenticationManager authenticationManag-
er(AuthenticationConfiguration config) throws Exception {
        return config.getAuthenticationManager();
    }
}
```

Файл: UserRole.java
Содержимое файла:

```
package com.primary.taskFlow.enums;
public enum UserRole {
    ADMIN, EMPLOYEE
}
```

Файл: TaskStatus.java
Содержимое файла:

```
package com.primary.taskFlow.enums;
public enum TaskStatus {
    PENDING, // в ожидании
    INPROGRESS,  // в процессе выполнения
    COMPLETED, // выполнена
    DEFERRED, //отложена
    CANCELLED, //отменена

}
```

Файл: JwtUtil.java
Содержимое файла:

```
package com.primary.taskFlow.utils;
import com.primary.taskFlow.entities.User;
import com.primary.taskFlow.repositories.UserRepository;
import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import io.jsonwebtoken.io.Decoders;
import io.jsonwebtoken.security.Keys;
import lombok.RequiredArgsConstructor;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Component;
import java.security.Key;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.Optional;
import java.util.function.Function;
@Component
```

```
@RequiredArgsConstructor
public class JwtUtil {
    private final UserRepository userRepository;
    public String generateToken(UserDetails userDetails) {
        return generateToken(new HashMap<>(), userDetails);
    }
    private String generateToken(Map<String, Object> extraClaims, UserDetails
userDetails) {
        return
Jwts.builder().setClaims(extraClaims).setSubject(userDetails.getUsername())
                .setIssuedAt(new Date(System.currentTimeMillis()))
                .setExpiration(new Date(System.currentTimeMillis() + 1000 *
60 * 60 * 24*10))//хранение токена
                .signWith(getSignKey(), SignatureAlgorithm.HS256).compact();
    }
    private Key getSignKey() {
        byte[] keyBytes = Decod-
ers.BASE64.decode("a1b2c3d4e5f6a1b2c3d4e5f6a1b2c3d4e5f6a1b2c3d4");
        return Keys.hmacShaKeyFor(keyBytes);
    }
    public boolean isTokenValid(String token, UserDetails userDetails) {
        final String userName = extractUserName(token);
        return (userName.equals(userDetails.getUsername())) && !isTokenEx-
pired(token);
    }
    public String extractUserName(String token) {
        return extractClaim(token, Claims::getSubject);
    }
    private boolean isTokenExpired(String token) {
        return extractExpiration(token).before(new Date());
    }
    private Date extractExpiration(String token) {
        return extractClaim(token, Claims::getExpiration);
    }
    private <T> T extractClaim(String token, Function<Claims, T> claimsRe-
solver) {
        final Claims claims = extractAllClaims(token);
        return claimsResolver.apply(claims);
    }
    private Claims extractAllClaims(String token) {
        return Jwts.parserBuilder().setSigningKey(getSignKey())
                .build()
                .parseClaimsJws(token)
                .getBody();
    }

   public User getLoggedInUser(){
        Authentication authentication = SecurityContextHold-
er.getContext().getAuthentication();
        if (authentication !=null && authentication.isAuthenticated()){
            User user = (User) authentication.getPrincipal();
            Optional<User> optionalUser = userReposito-
ry.findById(user.getId());
            return optionalUser.orElse(null);
        }
        return null;
    }

}
```

Файл: CommentRepository.java
Содержимое файла:
```
package com.primary.taskFlow.repositories;
import com.primary.taskFlow.dto.CommentDto;
import com.primary.taskFlow.entities.Comment;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import java.util.List;
@Repository
public interface CommentRepository extends JpaRepository<Comment, Long> {
    List<Comment> findAllByTaskId(Long taskId);
}
```
Файл: TaskRepository.java
Содержимое файла:
```
package com.primary.taskFlow.repositories;
import com.primary.taskFlow.dto.TaskDto;
import com.primary.taskFlow.entities.Task;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import java.util.List;
@Repository
public interface TaskRepository extends JpaRepository<Task, Long> {
    List<Task> findAllByTitleContaining(String title);

    List<Task> findAllByUserId(Long id);
}
```
Файл: UserRepository.java
Содержимое файла:
```
package com.primary.taskFlow.repositories;
import com.primary.taskFlow.entities.User;
import com.primary.taskFlow.enums.UserRole;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import java.util.Optional;
@Repository
public interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findFirstByEmail(String username);

    Optional<User> findByUserRole(UserRole userRole);
}
```
Файл: AuthController.java
Содержимое файла:
```
package com.primary.taskFlow.controller.auth;
import com.primary.taskFlow.dto.AuthenticationRequest;
import com.primary.taskFlow.dto.AuthenticationResponse;
import com.primary.taskFlow.dto.SignupRequest;
import com.primary.taskFlow.dto.UserDto;
import com.primary.taskFlow.entities.User;
import com.primary.taskFlow.repositories.UserRepository;
import com.primary.taskFlow.services.jwt.UserService;
import com.primary.taskFlow.services.auth.AuthService;
import com.primary.taskFlow.utils.JwtUtil;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
importrg.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.BadCredentialsException;
```

```
import
org.springframework.security.authentication.UsernamePasswordAuthenticationTok
en;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.web.bind.annotation.*;
import java.util.Optional;
@RestController
@RequestMapping("/api/auth")
@RequiredArgsConstructor
@CrossOrigin("*")
public class AuthController {
    private final AuthService authService;
    private final UserRepository userRepository;
    private final JwtUtil jwtUtil;
    private final UserService userService;
    private final AuthenticationManager authenticationManager;
    @PostMapping("/signup")
    public ResponseEntity<?> signupUser(@RequestBody SignupRequest signupRe-
quest) {
        if (authService.hasUserWithEmail(signupRequest.getEmail()))
            return
ResponseEntity.status(HttpStatus.NOT_ACCEPTABLE).body("Пользователь с таким
E-mail, уже существует");
        UserDto createUserDto = authService.signupUser(signupRequest);
        if (createUserDto == null)
            return
ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Пользователь не создан");
        return ResponseEntity.status(HttpStatus.CREATED).body(createUserDto);
    }
    @PostMapping("/login")
    public AuthenticationResponse login(@RequestBody AuthenticationRequest
authenticationRequest) {
        try {
            authenticationManager.authenticate(new UsernamePasswordAuthenti-
cationToken(
                    authenticationRequest.getEmail(),
                    authenticationRequest.getPassword()));
        } catch (BadCredentialsException e) {
            throw new BadCredentialsException("Неправильное имя пользователя
или пароль");
        }
        final UserDetails userDetails = userSer-
vice.userDetailService().loadUserByUsername(
                authenticationRequest.getEmail());
        Optional<User> optionalUser = userReposito-
ry.findFirstByEmail(authenticationRequest.getEmail());
        final String jwtToken = jwtUtil.generateToken(userDetails);
        AuthenticationResponse authenticationResponse = new Authenticatio-
nResponse();
        if (optionalUser.isPresent()) {
            authenticationResponse.setJwt(jwtToken);
            authenticationResponse.setUserId(optionalUser.get().getId());
            authenticationRes-
ponse.setUserRole(optionalUser.get().getUserRole());
        }
        return authenticationResponse;
    }
}
```

Файл: AdminController.java

Содержимое файла:

```
package com.primary.taskFlow.controller.admin;
import com.primary.taskFlow.dto.CommentDto;
import com.primary.taskFlow.dto.TaskDto;
import com.primary.taskFlow.services.admin.AdminService;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import java.util.List;
@RestController
@RequiredArgsConstructor
@RequestMapping("api/admin")
@CrossOrigin("*")
public class AdminController {
    private final AdminService adminService;
    @GetMapping("/users")
    public ResponseEntity<?> getUsers() {
        return ResponseEntity.ok(adminService.getUsers());
    }

    @PostMapping("/task")
    public ResponseEntity<TaskDto> createTask(@RequestBody TaskDto taskDto) {
        TaskDto createTaskDto = adminService.createTask(taskDto);
        if (createTaskDto == null) return ResponseEnti-
ty.status(HttpStatus.BAD_REQUEST).build();
        return ResponseEntity.status(HttpStatus.CREATED).body(createTaskDto);
    }
    @GetMapping("/tasks")
    public ResponseEntity<?> getAllTasks() {
        return ResponseEntity.ok(adminService.getAllTasks());
    }
    @DeleteMapping("/task/{id}")
    public ResponseEntity<Void> deleteTask(@PathVariable Long id) {
        adminService.deleteTask(id);
        return ResponseEntity.ok(null);
    }
    @GetMapping("/task/{id}")
    public ResponseEntity<TaskDto> getTaskById(@PathVariable Long id) {
        return ResponseEntity.ok(adminService.getTaskById(id));
    }
    @PutMapping("/task/{id}")
    public ResponseEntity<?> updateTask(@PathVariable Long id, @RequestBody
TaskDto taskDto) {
       TaskDto updateTask = adminService.updateTask(id, taskDto);
       if (updateTask == null) return ResponseEntity.notFound().build();
       return ResponseEntity.ok(updateTask);
    }
    @GetMapping("/tasks/search/{title}")
    public ResponseEntity<List<TaskDto>> searchTask(@PathVariable String
title) {
        return ResponseEntity.ok(adminService.searchTaskByTitle(title));
    }
    @PostMapping("/task/comment/{taskId}")
    public ResponseEntity<CommentDto> createComment(@PathVariable Long task-
Id, @RequestParam String content) {
        CommentDto createCommentDto = adminService.createComment(taskId, con-
tent);
```

```
        if (createCommentDto == null) return ResponseEnti-
ty.status(HttpStatus.BAD_REQUEST).build();
        return ResponseEnti-
ty.status(HttpStatus.CREATED).body(createCommentDto);
    }
    @GetMapping("/comments/{taskId}")
    public ResponseEntity<List<CommentDto>> getCommentsByTaskId(@PathVariable
Long taskId) {
        return ResponseEntity.ok(adminService.getCommentsByTaskId(taskId));
    }
}
```

Файл: EmployeeController.java

Содержимое файла:

```
package com.primary.taskFlow.controller.employee;
import com.primary.taskFlow.dto.CommentDto;
import com.primary.taskFlow.dto.TaskDto;
import com.primary.taskFlow.services.employee.EmployeeService;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import java.util.List;
@RestController
@RequestMapping("/api/employee")
@RequiredArgsConstructor
@CrossOrigin("*")
public class EmployeeController {
    private final EmployeeService employeeService;
    @GetMapping("/tasks")
    public ResponseEntity<List<TaskDto>> getTasksByUserId() {
        return ResponseEntity.ok(employeeService.getTasksByUserId());
    }
    @GetMapping("/task/{id}/{status}")
    public ResponseEntity<TaskDto> updateTask(@PathVariable Long id,
@PathVariable String status) {
        TaskDto updatedTaskDto = employeeService.updateTask(id, status);
        if (updatedTaskDto == null)
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).build();
        return ResponseEntity.ok(updatedTaskDto);
    }
    @GetMapping("/task/{id}")
    public ResponseEntity<TaskDto> getTaskById(@PathVariable Long id) {
        return ResponseEntity.ok(employeeService.getTaskById(id));
    }
    @PostMapping("/task/comment/{taskId}")
    public ResponseEntity<CommentDto> createComment(@PathVariable Long task-
Id, @RequestParam String content) {
        CommentDto createCommentDto = employeeService.createComment(taskId,
content);
        if (createCommentDto == null) return ResponseEnti-
ty.status(HttpStatus.BAD_REQUEST).build();
        return ResponseEnti-
ty.status(HttpStatus.CREATED).body(createCommentDto);
    }
    @GetMapping("/comments/{taskId}")
    public ResponseEntity<List<CommentDto>> getCommentsByTaskId(@PathVariable
Long taskId) {
        return ResponseEnti-
ty.ok(employeeService.getCommentsByTaskId(taskId));
```

```
    }
}
```
Файл: UserService.java
Содержимое файла:
```
package com.primary.taskFlow.services.jwt;
import org.springframework.security.core.userdetails.UserDetailsService;
public interface UserService {
   UserDetailsService userDetailService();
}
```
Файл: UserServiceImpl.java
Содержимое файла:
```
package com.primary.taskFlow.services.jwt;
import com.primary.taskFlow.repositories.UserRepository;
import lombok.RequiredArgsConstructor;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import
org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
@Service
@RequiredArgsConstructor
public class UserServiceImpl implements UserService {
    private final UserRepository userRepository;
    @Override
    public UserDetailsService userDetailService() {
        return new UserDetailsService() {
            @Override
            public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
                return  userReposito-
ry.findFirstByEmail(username).orElseThrow(()->new
UsernameNotFoundException("Пользователь не найден"));
            }
        };
    }
}
```
Файл: AuthService.java
Содержимое файла:
```
package com.primary.taskFlow.services.auth;
import com.primary.taskFlow.dto.SignupRequest;
import com.primary.taskFlow.dto.UserDto;
public interface AuthService {
   UserDto signupUser(SignupRequest signupRequest);
   boolean hasUserWithEmail(String email);
}
```
Файл: AuthServiceImpl.java
Содержимое файла:
```
package com.primary.taskFlow.services.auth;
import com.primary.taskFlow.dto.SignupRequest;
import com.primary.taskFlow.dto.UserDto;
import com.primary.taskFlow.entities.User;
import com.primary.taskFlow.enums.UserRole;
import com.primary.taskFlow.repositories.UserRepository;
import jakarta.annotation.PostConstruct;
import lombok.RequiredArgsConstructor;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service;
```

```
import java.util.Optional;
@Service
@RequiredArgsConstructor
public class AuthServiceImpl implements AuthService {
    private final UserRepository userRepository;
    @PostConstruct
    public void createAnAdminAccount() {
        Optional<User> optionalUser = userReposito-
ry.findByUserRole(UserRole.ADMIN);
        if (optionalUser.isEmpty()) {
            User user = new User();
            user.setEmail("admin@primary.taskFlow.com");
            user.setName("admin");
            user.setPassword(new BCryptPasswordEncoder().encode("admin"));
            user.setUserRole(UserRole.ADMIN);
            userRepository.save(user);
            System.out.println("Администратор создан");
        } else {
            System.out.println("Администратор уже существует");
        }
    }
    @Override
    public UserDto signupUser(SignupRequest signupRequest) {
        User user = new User();
        user.setEmail(signupRequest.getEmail());
        user.setName(signupRequest.getName());
        user.setPassword(new BCryptPasswordEncod-
er().encode(signupRequest.getPassword()));
        user.setUserRole(UserRole.EMPLOYEE);
        User createdUser = userRepository.save(user);
        return createdUser.getUserDto();
    }
    @Override
    public boolean hasUserWithEmail(String email) {
        return userRepository.findFirstByEmail(email).isPresent();
    }
}
```

Файл: AdminService.java

Содержимое файла:

```
package com.primary.taskFlow.services.admin;
import com.primary.taskFlow.dto.CommentDto;
import com.primary.taskFlow.dto.TaskDto;
import com.primary.taskFlow.dto.UserDto;
import java.util.List;
public interface AdminService {
    List<UserDto> getUsers();
    TaskDto createTask(TaskDto taskDto);
    List<TaskDto> getAllTasks();
    void deleteTask(Long id);
    TaskDto getTaskById(Long id);
    TaskDto updateTask(Long id, TaskDto taskDto);
    List<TaskDto> searchTaskByTitle(String title);
    CommentDto createComment(Long taskId, String content);
    List<CommentDto> getCommentsByTaskId(Long taskId);
}
```

Файл: AdminServiceImpl.java

Содержимое файла:

```
package com.primary.taskFlow.services.admin;
import com.primary.taskFlow.dto.CommentDto;
```

```
import com.primary.taskFlow.dto.TaskDto;
import com.primary.taskFlow.dto.UserDto;
import com.primary.taskFlow.entities.Comment;
import com.primary.taskFlow.entities.Task;
import com.primary.taskFlow.entities.User;
import com.primary.taskFlow.enums.TaskStatus;
import com.primary.taskFlow.enums.UserRole;
import com.primary.taskFlow.repositories.CommentRepository;
import com.primary.taskFlow.repositories.TaskRepository;
import com.primary.taskFlow.repositories.UserRepository;
import com.primary.taskFlow.utils.JwtUtil;
import jakarta.persistence.EntityNotFoundException;
import lombok.RequiredArgsConstructor;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Comparator;
import java.util.Date;
import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;
@Service
@RequiredArgsConstructor
          public class AdminServiceImpl implements AdminService {
    private final UserRepository userRepository;
    private final TaskRepository taskRepository;
    private final JwtUtil jwtUtil;
    private final CommentRepository commentRepository;
    @Override
    public List<UserDto> getUsers() {
        return userRepository.findAll().stream().filter(user -> us-
er.getUserRole() == UserRole.EMPLOYEE)
                .map(User::getUserDto).collect(Collectors.toList());
    }
    @Override
    public TaskDto createTask(TaskDto taskDto) {
        Optional<User> optionalUser = userReposito-
ry.findById(taskDto.getEmployeeId());
        if (optionalUser.isPresent()) {
            Task task = new Task();
            task.setTitle(taskDto.getTitle());
            task.setDescription(taskDto.getDescription());
            task.setPriority(taskDto.getPriority());
            task.setDueDate(taskDto.getDueDate());
            task.setTaskStatus(TaskStatus.INPROGRESS);
            task.setUser(optionalUser.get());
            return taskRepository.save(task).getTaskDto();
        }
        return null;
    }
    @Override
    public List<TaskDto> getAllTasks() { //компоратор задач по дате
        return taskRepository.findAll().stream().sorted(Comparator
                        .comparing(Task::getDueDate)
                        .reversed()).map(Task::getTaskDto)
                .collect(Collectors.toList());
    }
    @Override
```

```
    public void deleteTask(Long id) {
        taskRepository.deleteById(id);
    }
    @Override
    public TaskDto getTaskById(Long id) {
        Optional<Task> optionalTask = taskRepository.findById(id);
        return optionalTask.map(Task::getTaskDto).orElse(null);
    }
    @Override
    public TaskDto updateTask(Long id, TaskDto taskDto) {
        Optional<Task> optionalTask = taskRepository.findById(id);
        Optional<User> optionalUser = userReposito-
ry.findById(taskDto.getEmployeeId());
        if (optionalTask.isPresent() && optionalUser.isPresent()) {
            Task existingTask = optionalTask.get();
            existingTask.setTitle(taskDto.getTitle());
            existingTask.setDescription(taskDto.getDescription());
            existingTask.setDueDate(taskDto.getDueDate());
            existingTask.setPriority(taskDto.getPriority());
            existing-
Task.setTaskStatus(mapStringToTaskStatus(String.valueOf(taskDto.getTaskStatus
()));
            existingTask.setUser(optionalUser.get());
            return taskRepository.save(existingTask).getTaskDto();
        }
        return null;
    }
    @Override
    public List<TaskDto> searchTaskByTitle(String title) {
        return taskRepository.findAllByTitleContaining(title)
                .stream()
                .sorted(Comparator
                        .comparing(Task::getDueDate)
                        .reversed()).map(Task::getTaskDto)
                .collect(Collectors.toList());
    }
    @Override
    public CommentDto createComment(Long taskId, String content) {
        Optional<Task> optionalTask = taskRepository.findById(taskId);
        User user = jwtUtil.getLoggedInUser();
        if ((optionalTask.isPresent()) && user != null) {
            Comment comment = new Comment();
            comment.setCreatedAt(new Date());
            comment.setContent(content);
            comment.setTask(optionalTask.get());
            comment.setUser(user);
            return commentRepository.save(comment).getCommentDto();
        }
        throw new EntityNotFoundException("Пользователь либо задача не найде-
ны");
    }
    @Override
    public List<CommentDto> getCommentsByTaskId(Long taskId) {
        return commentRepository.findAllByTaskId(taskId)
                .stream().map(Comment::getCommentDto)
                .collect(Collectors.toList());
    }
    private TaskStatus mapStringToTaskStatus(String status) {
        return switch (status) {
```

```
            case "PENDING" -> TaskStatus.PENDING;
            case "INPROGRESS" -> TaskStatus.INPROGRESS;
            case "COMPLETED" -> TaskStatus.COMPLETED;
            case "DEFERRED" -> TaskStatus.DEFERRED;
            default -> TaskStatus.CANCELLED;
        };
    }
}
```

Файл: EmployeeServiceImpl.java

Содержимое файла:

```
package com.primary.taskFlow.services.employee;
import com.primary.taskFlow.dto.CommentDto;
import com.primary.taskFlow.dto.TaskDto;
import com.primary.taskFlow.entities.Comment;
import com.primary.taskFlow.entities.Task;
import com.primary.taskFlow.entities.User;
import com.primary.taskFlow.enums.TaskStatus;
import com.primary.taskFlow.repositories.CommentRepository;
import com.primary.taskFlow.repositories.TaskRepository;
import com.primary.taskFlow.utils.JwtUtil;
import jakarta.persistence.EntityNotFoundException;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import java.util.Comparator;
import java.util.Date;
import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;
@Service
@RequiredArgsConstructor
public class EmployeeServiceImpl implements EmployeeService {
    private final TaskRepository taskRepository;
    private final JwtUtil jwtUtil;
    private final CommentRepository commentRepository;
    @Override
    public List<TaskDto> getTasksByUserId() {
        User user = jwtUtil.getLoggedInUser();
        if (user != null) {
            return taskRepository.findAllByUserId(user.getId())
                    .stream()
                    .sorted(Comparator.comparing(Task::getDueDate)
                            .reversed())
                    .map(Task::getTaskDto)
                    .collect(Collectors.toList());
        }
        throw new EntityNotFoundException("Пользователь не найден");
    }
    @Override
    public TaskDto updateTask(Long id, String status) {
        Optional<Task> optionalTask = taskRepository.findById(id);
        if (optionalTask.isPresent()) {
            Task existingTask = optionalTask.get();
            existingTask.setTaskStatus(mapStringToTaskStatus(status));
            return taskRepository.save(existingTask).getTaskDto();
        }
        throw new EntityNotFoundException("Задача не найдена");
    }
    @Override
    public TaskDto getTaskById(Long id) {
```

```
        Optional<Task> optionalTask = taskRepository.findById(id);
        return optionalTask.map(Task::getTaskDto).orElse(null);
    }
    @Override
    public CommentDto createComment(Long taskId, String content) {
        Optional<Task> optionalTask = taskRepository.findById(taskId);
        User user = jwtUtil.getLoggedInUser();
        if ((optionalTask.isPresent()) && user != null) {
            Comment comment = new Comment();
            comment.setCreatedAt(new Date());
            comment.setContent(content);
            comment.setTask(optionalTask.get());
            comment.setUser(user);
            return commentRepository.save(comment).getCommentDto();
        }
        throw new EntityNotFoundException("Пользователь либо задача не найде-
ны");
    }
    @Override
    public List<CommentDto> getCommentsByTaskId(Long taskId) {
        return commentRepository.findAllByTaskId(taskId)
                .stream().map(Comment::getCommentDto)
                .collect(Collectors.toList());
    }
    private TaskStatus mapStringToTaskStatus(String status) {
        return switch (status) {
            case "PENDING" -> TaskStatus.PENDING;
            case "INPROGRESS" -> TaskStatus.INPROGRESS;
            case "COMPLETED" -> TaskStatus.COMPLETED;
            case "DEFERRED" -> TaskStatus.DEFERRED;
            default -> TaskStatus.CANCELLED;
        };
    }
}
```

Файл: EmployeeService.java

Содержимое файла:

```
package com.primary.taskFlow.services.employee;
import com.primary.taskFlow.dto.CommentDto;
import com.primary.taskFlow.dto.TaskDto;
import java.util.List;
public interface EmployeeService {
    List<TaskDto> getTasksByUserId();
    TaskDto updateTask(Long id, String status);
    TaskDto getTaskById(Long id);
    CommentDto createComment(Long taskId, String content);
    List<CommentDto> getCommentsByTaskId(Long taskId);
}
```

Файл: User.java

Содержимое файла:

```
package com.primary.taskFlow.entities;
import com.primary.taskFlow.dto.UserDto;
import com.primary.taskFlow.enums.UserRole;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import lombok.Data;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
```

```
import org.springframework.security.core.userdetails.UserDetails;
import java.util.Collection;
import java.util.List;
@Data
@Entity
public class User implements UserDetails {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String email;
    private String password;
    private UserRole userRole;
    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return List.of(new SimpleGrantedAuthority(userRole.name()));
    }
    @Override
    public String getUsername() {
        return email;
    }
    @Override
    public boolean isAccountNonExpired() {
        return true;
    }
    @Override
    public boolean isAccountNonLocked() {
        return true;
    }
    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }
    @Override
    public boolean isEnabled() {
        return true;
    }
    public UserDto getUserDto() {
        UserDto userDto= new UserDto();
        userDto.setId(id);
        userDto.setName(name);
        userDto.setEmail(email);
        userDto.setUserRole(userRole);
        return userDto;
    }
}
```

Файл: Comment.java

Содержимое файла:

```
package com.primary.taskFlow.entities;
import com.primary.taskFlow.dto.CommentDto;
import com.fasterxml.jackson.annotation.JsonIgnore;
import jakarta.persistence.*;
import lombok.Data;
import org.hibernate.annotations.OnDelete;
import org.hibernate.annotations.OnDeleteAction;
import java.util.Date;
@Entity
@Data
public class Comment {
```

```
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String content;
    private Date createdAt;
    @ManyToOne(fetch = FetchType.LAZY, optional = false) //многие к одному
    @JoinColumn(name = "user_id", nullable = false)
    @OnDelete(action = OnDeleteAction.CASCADE) //ложная авторизация
    @JsonIgnore
    private User user;
    @ManyToOne(fetch = FetchType.LAZY, optional = false) //многие к одному
    @JoinColumn(name = "task_id", nullable = false)
    @OnDelete(action = OnDeleteAction.CASCADE) //ложная авторизация
    @JsonIgnore
    private Task task;
    public CommentDto getCommentDto() {
        CommentDto commentDto = new CommentDto();
        commentDto.setId(id);
        commentDto.setContent(content);
        commentDto.setCreatedAt(createdAt);
        commentDto.setTaskId(task.getId());
        commentDto.setPostedBy(user.getName());
        return commentDto;
    }
}
```

Файл: Task.java

Содержимое файла:

```
package com.primary.taskFlow.entities;
import com.primary.taskFlow.dto.TaskDto;
import com.primary.taskFlow.enums.TaskStatus;
import com.fasterxml.jackson.annotation.JsonIgnore;
import jakarta.persistence.*;
import lombok.Data;
import org.hibernate.annotations.OnDelete;
import org.hibernate.annotations.OnDeleteAction;
import java.text.SimpleDateFormat;
import java.util.Date;
@Data
@Entity
public class Task {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String title; //заголовок
    private String description; //описание
    private Date dueDate; //срок выполнения
    private String priority;// приоритет задачи
    private TaskStatus taskStatus; //статус задачи
    @ManyToOne(fetch = FetchType.LAZY, optional = false) //многие к одному
    @JoinColumn(name = "user_id", nullable = false)
    @OnDelete(action = OnDeleteAction.CASCADE) //ложная авторизация
    @JsonIgnore
    private User user;
    public TaskDto getTaskDto() {
        TaskDto taskDto = new TaskDto();
        taskDto.setId(id);
        taskDto.setTitle(title);
        taskDto.setDescription(description);
        taskDto.setEmployeeName(user.getName());
```

```
            taskDto.setEmployeeId(user.getId());
            taskDto.setTaskStatus(taskStatus);
            taskDto.setDueDate(dueDate);
            taskDto.setPriority(priority);
            return taskDto;
        }
    }
```