МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____ Л.Б. Соколинский

«___»_____ 2024 г.

Разработка веб-приложения для транспортной компании «Луч»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.03.04.2024.308-350.ВКР

Научный руководитель,
доцент кафедры СП, к.т.н.
_____ М.В. Сухов

Автор работы,
студент группы КЭ-403
_____ В.А. Рявкин

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
«___»_____ 2024 г.

Челябинск, 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
29.01.2024 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра

студенту группы КЭ-403
Рявкину Виталию Алексеевичу,
обучающемуся по направлению
09.03.04 «Программная инженерия»

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12)
Разработка веб-приложения для транспортной компании «Луч».
2. **Срок сдачи студентом законченной работы:** 03.06.2024 г.
3. **Исходные данные к работе**
3.1. De Sanctis V. Building Web APIs with ASP.NET Core. //Manning, 2023. – 472 с.
3.2. Entity framework documentation. [Электронный ресурс] URL https://learn.microsoft.com/en-us/ef/core/ (дата обращения: 18.02.2024 г.).
3.3. ASP .NET Core web API documentation. [Электронный ресурс] URL: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/apis?view=aspnetcore-7 (дата обращения: 18.02.2024 г.).
4. **Перечень подлежащих разработке вопросов**
4.1. Провести анализ предметной области.
4.2. Выполнить проектирование API и базы данных.
4.3. Реализовать API.
4.4. Провести тестирование API и сравнить полученные результаты с ожидаемыми.
5. **Дата выдачи задания:** 29.01.2024 г.

Научный руководитель,
доцент кафедры СП, к.т.н. М.В. Сухов

Задание принял к исполнению В.А. Рявкин

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

В современном мире большинство компаний стремятся к автоматизации всех бизнес-процессов, в том числе и в области управления персоналом. В сфере транспортной логистики и складского хозяйства этот вопрос особенно важен, так как необходимо точно и своевременно рассчитывать заработную плату сотрудников складов и иметь постоянный контроль над ситуацией на складах.

Разрабатываемая система позволит автоматизировать процесс расчета заработной платы, уменьшить количество ошибок в работе управляющего персонала, улучшить контроль над исполнителями и повысить эффективность работы сотрудников складов. Это также предоставит возможность руководству быстро получать информацию о заработной плате сотрудников и проводить анализ затрат на персонал.

Постановка задачи

Целью данной работы является разработка веб-приложения для транспортной компании «Луч» по расчету заработной платы сотрудников складов и контроля качества выполненных работ. Для ее достижения необходимо решить следующие задачи:

1) проанализировать существующие аналоги;
2) на основе анализа спроектировать архитектуру системы и выбрать технологии, которые будут применяться для решения поставленной задачи;
3) реализовать приложение согласно спроектированной архитектуре;
4) протестировать приложение.

Структура и содержание работы

Работа состоит из введения, четырех глав, заключения и списка литературы. Объем работы составляет 36 страниц, объем списка литературы – 15 источников.

В первой главе осуществляется анализ предметной области, включая обзор существующих исследований и практических работ, связанных с учетом заработных плат и контролем производительности персонала. Этот анализ помогает более полно понять контекст, в котором функционирует разрабатываемая система, а также выявить основные потребности и требования конечных пользователей.

Во второй главе подробно рассматриваются требования к программной системе. Определяются как основные функциональные, так и нефункциональные требования, учитывая желаемые характеристики и возможности системы. Также в этой главе формируется диаграмма вариантов использования, которая иллюстрирует различные сценарии использования системы, а также составляется спецификация основных прецедентов, описывающих взаимодействие пользователей с системой.

Третья глава посвящена архитектуре системы. Представлено общее описание архитектуры, включая описание компонентов, составляющих систему, их взаимосвязи и взаимодействие. Также в этой главе приводится модель базы данных, определяющая структуру и хранение данных системы, а также описывается процесс работы с системой с точки зрения ее архитектуры.

В четвертой главе представлена реализация ключевых компонентов системы. Здесь подробно описывается процесс разработки и реализации функциональности, включая выбранные технологии и инструменты. Также в этой главе проводится тестирование системы с целью проверки ее корректности работы и соответствия заявленным требованиям, что позволяет убедиться в успешной реализации поставленных задач.

В приложении содержатся скриншоты пользовательского интерфейса.

1. ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ ПРОЕКТА

1.1. Предметная область проекта

Предметная область проекта связана с веб-приложения для транспортной компании «Луч» для расчета заработной платы сотрудников складов и контроля качества выполненных работ.

ТК «Луч» – это логистическая сеть, представленная в 77 городах, с 47 полноценными складами и 30 пунктами выдачи с борта автомобиля. Компания осуществляет ежедневную срочную доставку грузов в 9 регионах Российской Федерации и Республику Казахстан. В главном терминале в Челябинске более 100 специалистов ежедневно поддерживают работу всей транспортной системы. Каждый день в рейс выходит более 100 автомобилей. Партнерами ТК «Луч» являются крупные компании, такие как Экспресс-почта, Ozon, Boxberry, Invitro и другие.

Основной задачей приложения является помощь управляющему персоналу в контроле посещаемости и качества работы подчиненных сотрудников.

1.2. Анализ аналогичных проектов

Существует множество аналогов систем для учета посещаемости сотрудников и просчета заработной платы на основе этой посещаемости и других факторов. В качестве примера можно рассмотреть некоторые из них: Workday, SAP SuccessFactors, Kronos Workforce Management, TSheets.

Workday

Workday [1] – это облачный сервис управления персоналом и финансами для предприятий. Этот сервис позволяет компаниям управлять человеческими ресурсами, управлять зарплатами и выплатами, автоматизировать бухгалтерские процессы и многое другое. Он обеспечивает функции, такие как управление наймом, обучением и развитием сотрудников, управление рабочим временем и отпусками, управление производительностью, управление зарплатой и выплатами, бухгалтерские операции и аналитику.

Workday имеет обширную систему безопасности и защиты данных, которая обеспечивает защиту конфиденциальности информации компаний и сотрудников. Сервис также обеспечивает высокую масштабируемость и доступность, что позволяет компаниям масштабировать свой бизнес без проблем.

Технологии:

- используется язык программирования Java и фреймворк Spring;
- в качестве базы данных используется PostgreSQL;
- интерфейс реализован с помощью HTML, CSS и JavaScript.

SAP SuccessFactors

SAP SuccessFactors [2] – это облачный сервис управления персоналом, предназначенный для организации HR-процессов в компаниях любого размера. Он включает в себя различные модули, такие как управление персоналом, найм, обучение и развитие, управление производительностью, компенсации и бенефиты, а также аналитику и отчетность.

Благодаря интеграции с другими системами, SAP SuccessFactors позволяет создавать единую информационную среду для управления персоналом и повышения эффективности бизнеса.

Технологии:

- использует язык программирования Java и фреймворк Spring;
- в качестве базы данных использует SAP HANA;
- интерфейс реализован с помощью HTML, CSS и JavaScript.

Kronos

Kronos [3] – это облачный сервис управления рабочим временем, который предоставляет широкий спектр функций, включая возможность планирования графиков работы, отслеживания рабочего времени, учета отпусков и больничных, а также отчетности о рабочих часах и зарплате сотрудников, который предназначен для упрощения и автоматизации процессов учета рабочего времени в компаниях различных отраслей.

Кроме того, Kronos интегрируется с другими бизнес-приложениями, такими как системы управления персоналом, ERP-системы и системы учета рабочих часов, что позволяет создавать комплексные решения для управления бизнесом.

Сервис Kronos используется в компаниях различного масштаба и отраслей, включая розничную торговлю, здравоохранение, гостинично-ресторанный бизнес, производство и транспортную логистику. Он предоставляет компаниям возможность улучшить управление персоналом, повысить производительность и эффективность работы сотрудников, а также сократить затраты на учет рабочего времени и зарплаты.

Технологии:

- язык программирования Java;
- система управления базами данных Oracle;
- фреймворк Spring Framework.

TSheets

TSheets [4] – это онлайн-сервис для учета рабочего времени, управления задачами и графиками работы сотрудников. С помощью этого сервиса компании могут легко отслеживать время работы своих сотрудников, а также управлять их графиками и задачами.

TSheets позволяет сотрудникам быстро и легко отмечать свое время работы через мобильное приложение, веб-приложение или через специальные устройства для учета времени. Система автоматически собирает данные о времени работы и обрабатывает их для удобного отчета и анализа.

Сервис также позволяет настраивать различные права доступа и роли для пользователей, а также создавать отчеты о времени работы, задачах и графиках работы. Кроме того, TSheets интегрируется с другими приложениями, такими как QuickBooks, Xero и Gusto, что упрощает управление бизнесом и финансами.

Технологии:

- язык программирования JavaScript и фреймворк Node.js для API;

- система управления базами данных MongoDB;
- фреймворк AngularJS для фронтенда.

1.3. Описание средств реализации

В результате анализа подобных решений было принято решение о применении следующих технологий для проектирования приложения: ASP.NET Core Web API-приложения с использованием Entity Framework Core и PostgreSQL на сервере и Angular на клиенте.

ASP.NET Core Web API [5] – это кросплатформенный фреймворк для создания веб-сервисов (API) на языке программирования C# с использованием технологии ASP.NET Core. Он предназначен для разработки API, которые могут быть использованы различными клиентскими приложениями, такими как веб-приложения, мобильные приложения, настольные приложения и другие.

Entity Framework Core – это ORM (Object-Relational Mapping) фреймворк, разработанный Microsoft для работы с базами данных в приложениях .NET Core. Он позволяет разработчикам работать с данными, используя объектно-ориентированный подход, что упрощает работу с базами данных и уменьшает количество кода, необходимого для взаимодействия с ними.

PostgreSQL [6] (или Postgres) – это реляционная объектно-ориентированная система управления базами данных (СУБД), которая использует язык SQL (Structured Query Language) для управления данными. PostgreSQL является свободно распространяемой и открытой системой, которая может работать на различных операционных системах, включая Linux, Windows, macOS и другие.

React [7] – это библиотека для разработки пользовательских интерфейсов на JavaScript. Она предлагает компонентную архитектуру, виртуаль-

ный DOM, односторонний поток данных и JSX. Компоненты позволяют переиспользовать код, виртуальный DOM ускоряет обновления, а JSX упрощает написание интерфейсов.

Вывод по первой главе

В рамках исследования аналогичных решений для управления персоналом, учета рабочего времени и расчета заработной платы были рассмотрены системы Workday, SAP SuccessFactors, Kronos и TSheets. Эти облачные платформы предоставляют комплексные решения для управления персоналом, включая такие функции, как управление наймом, обучение сотрудников, учет рабочего времени, оценка производительности и расчет заработной платы. Каждая из этих систем предлагает богатый набор инструментов, направленных на улучшение управления человеческими ресурсами и оптимизацию рабочих процессов.

На основании проведенного анализа было принято решение использовать ASP.NET Core Web API для серверной части, что обеспечит высокую производительность и возможность масштабирования. Entity Framework Core будет применяться для работы с базами данных, упрощая взаимодействие с данными на более высоком уровне абстракции. Для хранения данных выбрана PostgreSQL, которая благодаря своей высокой производительности, надежности и возможности масштабирования является идеальным выбором для данного проекта. Клиентская часть приложения будет создана с использованием ReactJS, что позволит создавать быстрые и отзывчивые веб-приложения с высоким уровнем интерактивности и удобства использования.

Использование этих технологий обеспечит создание высокопроизводительного, масштабируемого и удобного в использовании приложения, которое поможет управляющему персоналу компании «Луч» эффективно контролировать посещаемость и качество работы сотрудников.

2. АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОЙ СИСТЕМЕ

В результате анализа предметной области и обзора существующих аналогов были сформированы следующие два основных типа требований:

- функциональные требования – определяют, как должна работать система и какие функции она должна выполнять;
- нефункциональные требования – определяют ограничения, свойства и критерии качества системы.

Функциональные требования к проектируемой системе

Функциональные требования определяют основные функции, которые должна выполнять система. Требования предоставлены ниже.

1. Система должна предоставлять возможность изменения информации о личных данных сотрудников.
2. Система должна предоставлять разграничение уровня доступов для сотрудников с разными должностями.
3. Система должна сохранять и отслеживать любые изменения внутри себя.
4. Система должна предоставлять возможность отметки посещения сотрудников, приходящих на смену.
5. Система должна предоставлять возможность открытия и закрытия смены.
6. Система должна предоставлять возможность начальникам смены вносить комментарии о работе сотрудника на их смене.
7. Система должна предоставлять возможность редактирования информации об имеющихся должностях и позволять создавать новые.
8. Система должна предоставлять возможность получения отчета о сменах, прошедших на выбранном складе за выбранный месяц.
9. Система должна предоставлять возможность создания и изменения индивидуального рабочего плана для каждого сотрудника.
10. Система должна вести учет заработной платы сотрудника с учетом следующих факторов: количество отработанных часов из назначенных,

выслуга лет в данной компании, отпускные, штрафы, премии и наставничество.

Нефункциональные требования к проектируемой системе

Нефункциональные требования уточняют ограничения, качественные характеристики и ожидания относительно ее работы. Требования предоставлены ниже.

1. В качестве языка программирования используется C#.
2. В качестве основного фреймворка используется ASP .NET Core Web API.
3. В качестве ORM должен использоваться Entity Framework Core.
4. В качестве СУБД используется PostgreSQL.
5. Приложение должно поддерживать основные браузеры (Chrome, Firefox, Edge, Safari) и быть адаптированным для мобильных устройств.
6. Интерфейс должен быть интуитивно понятным и не требовать длительного обучения пользователей.
7. Система должна вести журналы действий пользователей для возможности аудита и отслеживания изменений данных.

2.1. Диаграмма вариантов использования

На основе требований, предъявляемых к разрабатываемому приложению, были разработаны варианты его использования, что позволило четко определить функциональные сценарии и роли, предусмотренные в системе. Такой подход обеспечивает понимание всех возможных операций, которые могут быть выполнены в приложении, и помогает выявить потенциальные улучшения и оптимизации в функциональности. На рисунке 1 представлены все варианты использования.

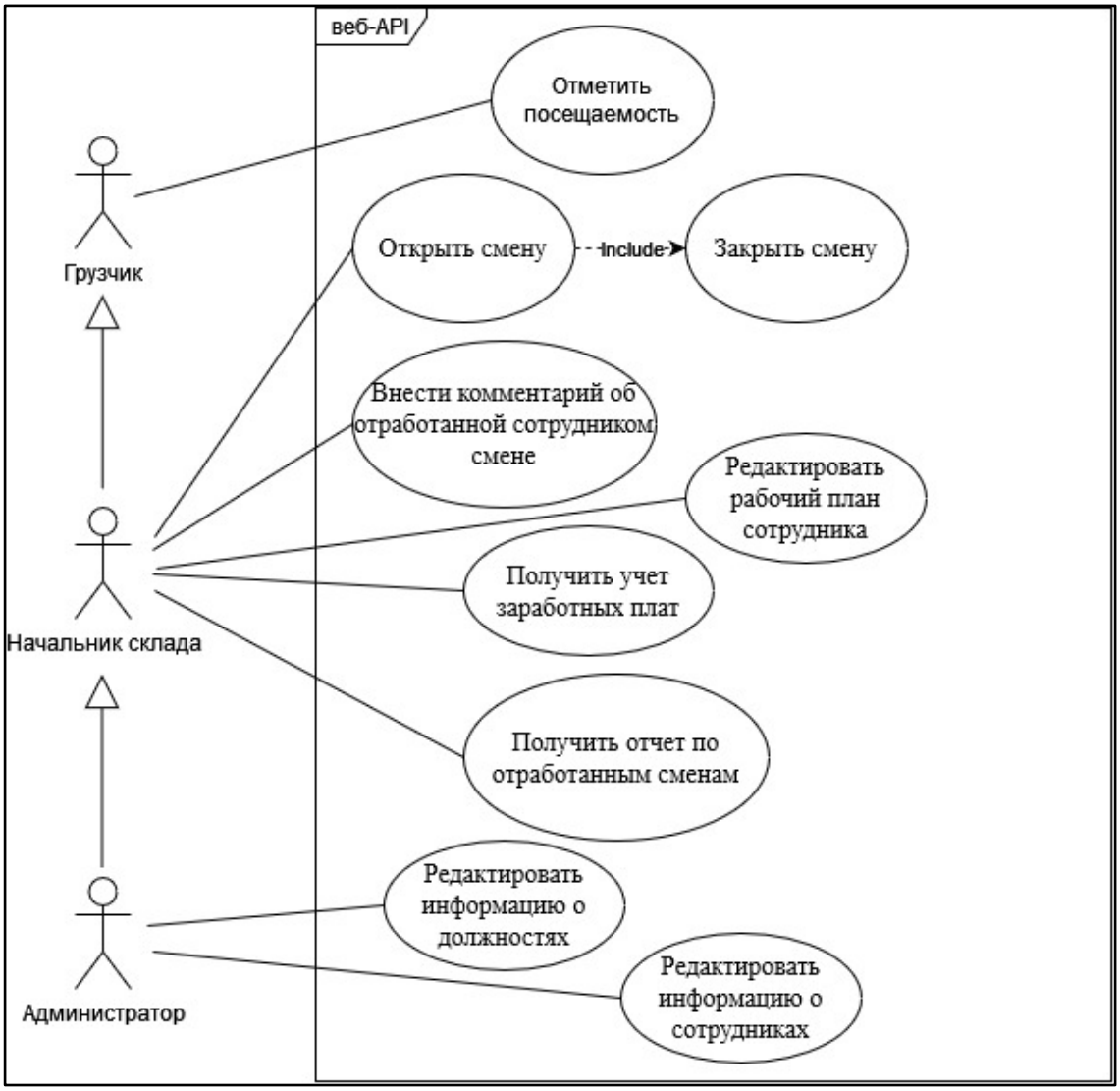


Рисунок 1 – Диаграмма вариантов использования

Список актеров представлен ниже.

1. Грузчик – сотрудник, выполняющий функции грузчика, являющийся сотрудником склада.

2. Начальник склада – сотрудник склада, отвечающий за открытие и закрытие смен, контроль выполненной работы и поддержку в актуальном состоянии рабочих планов.

3. Администратор – сотрудник, отвечающий за поддержание данных системы в актуальном состоянии.

Ниже приведен список прецедентов.

1. Отметить посещаемость – зафиксировать время своего прибытия на смену.

2. Редактировать информацию о сотрудниках – изменение информации о сотрудниках (ФИО, паспортные данные, дата начала стажа и т.д.).

3. Открыть смену – внести запись о том, что смена начата и утвердить список присутствующих на ней сотрудников.

4. Закрыть смену – внести запись о том, что смена закончена.

5. Внести комментарий об отработанной сотрудником смене – внесение какой-либо информации об отработанной смене.

6. Редактировать информацию о должностях – изменение информации о должностях (название, оклад и т.д.).

7. Редактировать рабочий план сотрудника – внесение изменений о том сколько смен и сколько часов должен отработать сотрудник за месяц.

8. Получить учет заработных плат – получение информации о том сколько сотрудник заработал за выбранный месяц.

9. Получить отчет по отработанным сменам – получение информации о том сколько сотрудник отработал часов и просмотр комментариев к сменам, если таковые имеются.

Вывод по второй главе

Сформулированы функциональные и нефункциональные требования к проекту. Функциональные требования определяют основные функции системы, такие как управление данными сотрудников, отметка посещаемости, формирование отчетов и учет заработной платы. Нефункциональные требования определяют технологии, используемые в разработке (например, C# и ASP.NET Core Web API), а также качественные характеристики системы, такие как поддержка различных браузеров и мобильных устройств, интуитивно понятный интерфейс и журналирование действий пользователей для аудита.

3. АРХИТЕКТУРА СИСТЕМЫ

3.1. Общее описание архитектуры системы и шаблонов экранов

При разработке серверной части данного приложения было принято решение придерживаться подхода Domain-driven design [8]. Domain-driven design (DDD) – это подход к проектированию программного обеспечения, который центрирован вокруг модели предметной области (domain model). Это означает, что в DDD есть стремление создать четкое понимание того, что представляет собой предметная область, которую мы моделируем, и выражаем это понимание в виде явно определенных классов, объектов, связей и операций в коде.

На рисунке 2 предоставлена схема компонентов приложения.

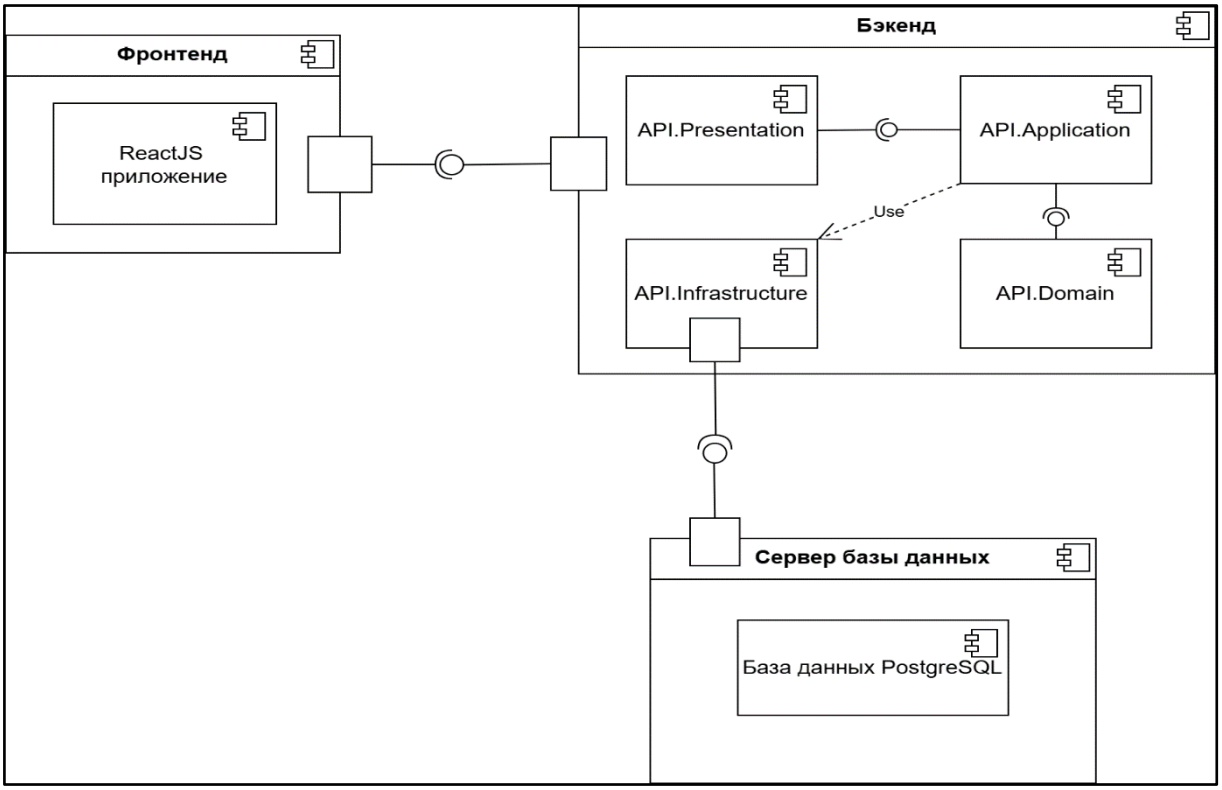


Рисунок 2 – Схема компонентов приложения

В процессе встреч с заказчиком был детально проработан и согласован единый шаблон для всех страниц приложения (рисунок 3). Этот шаблон был разработан с учетом всех пожеланий и требований заказчика, обеспечивая единообразный и интуитивно понятный интерфейс для конечных пользователей. Благодаря этому подходу удалось создать гармоничный и

целостный визуальный стиль, который способствует улучшению пользовательского опыта и облегчает навигацию по приложению.

Рисунок 3 – Шаблон рабочего экрана

3.2. Описание компонентов и сервисов, составляющих систему

Слой Presentation

Слой Presentation разработанного приложения отвечает, за взаимодействие пользователей с системой. Данный слой работает с интерфейсом пользователя, который позволяет взаимодействовать с приложением через различные способы, такие как веб-браузер, мобильное приложение или настольное приложение.

В приложении взаимодействие с системой будет доступно с помощью следующих наборов конечных точек.

1. Набор `Log in`, отвечающий за аутентификацию и авторизацию пользователей.

2. Набор `Employee`, отвечающий за изменение информации о сотрудниках, их увольнение и регистрацию, а также за получение информации о них.

3. Набор `Position`, отвечающий за изменение информации о должностях, их удаление и создание, а также за получение информации о них.

4. Набор `Stock`, отвечающий за получение списка складов с основного API компании и за изменение информации о звеньях складов.

5. Набор `Shift`, отвечающий за открытие и закрытие смен, редактирование и получение информации о них.

6. Набор `Attendance`, отвечающий за получение табеля посещений.

7. Набор `Workplan`, отвечающий за создание и редактирование рабочих планов сотрудников.

8. Набор `Accounting`, отвечающий за получение и редактирование таблицы учета рабочих часов.

Слой Infrastructure

Слой Infrastructure в приложении ответственен за обеспечение надежности, безопасности и производительности взаимодействия приложения с внешними системами и ресурсами. Этот слой играет ключевую роль в поддержании работоспособности всей системы и эффективного управления ее ресурсами.

Одной из основных задач слоя Infrastructure является обеспечение безопасного и эффективного взаимодействия с базой данных проекта. Это включает в себя управление соединениями с базой данных, выполнение запросов и обработку данных, а также обеспечение соответствия требованиям безопасности и целостности данных.

Кроме того, слой Infrastructure заботится о реализации механизмов обработки ошибок и контроля целостности данных, что способствует стабильной и безопасной работе приложения. Также он обеспечивает мониторинг и логирование операций, что позволяет быстро выявлять и решать проблемы производительности, безопасности и надежности системы.

В целом, слой Infrastructure играет важную роль в обеспечении эффективной работы приложения и поддержании его работоспособности в различных условиях эксплуатации.

Слой Application

Слой Application отвечает за реализацию бизнес-логики приложения. Этот слой работает с моделями данных, предоставляет интерфейс для управления бизнес-процессами и обрабатывает запросы от пользователей или других систем.

Основными задачами слоя Application в разработанном приложении являются следующие.

1. Реализация бизнес-логики. слой Application отвечает за обработку бизнес-правил и реализацию логики приложения, которая описывает, как приложение должно работать с данными и какие действия следует предпринимать при определенных событиях или условиях. Этот слой использует модели данных, которые предоставляются слоем Domain.

2. Обеспечение доступа к данным. слой Application обеспечивает доступ к данным, предоставляемым слоем Infrastructure. Он использует слой Infrastructure для работы с базами данных, файловыми системами и другими внешними системами.

3. Обработка запросов. слой Application обрабатывает запросы от пользователей или других систем и предоставляет им необходимую информацию. Он использует слой Presentation для отображения данных пользователю.

Слой Domain

Слой Domain является фундаментальной частью архитектуры приложения, ответственной за представление и управление бизнес-логикой. Этот слой состоит из моделей данных, которые описывают ключевые сущности и объекты бизнес-процессов, используемых в приложении. Модели данных в слое Domain являются абстракциями реальных объектов и процессов, которые присутствуют в предметной области приложения.

В моделях данных содержится не только информация о структуре и связях между объектами, но и бизнес-логика, определяющая правила работы с этими данными. Например, модель данных может определять, какие

операции можно выполнить над объектами, какие условия должны быть выполнены для совершения определенных действий, а также какие данные должны быть сохранены или получены из хранилища данных.

Слой Domain обеспечивает высокий уровень абстракции и независимость от конкретных технологий, что позволяет легко вносить изменения в бизнес-логику приложения без изменения других частей системы. Это делает код приложения более гибким, поддерживаемым и масштабируемым.

Кроме того, слой Domain часто служит важным элементом для обеспечения безопасности данных и правильного взаимодействия между различными компонентами приложения. Благодаря четкому разделению бизнес-логики от других слоев приложения, таких как слой представления или слой доступа к данным, управление и сопровождение приложения становится более простым и прозрачным.

3.3. Модель базы данных

На рисунке 4 представлена диаграмма базы данных проекта, содержащая только ключи.

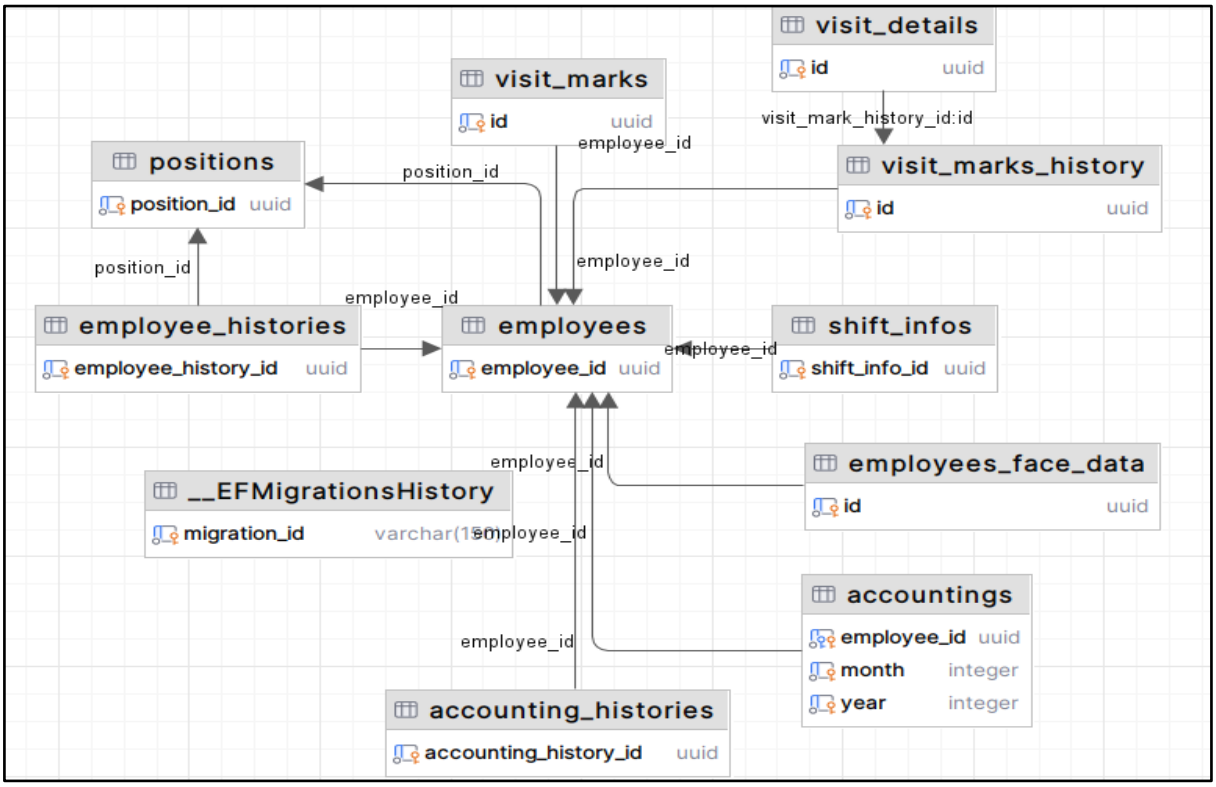


Рисунок 4 – Диаграмма базы данных проекта

Вывод по третьей главе

В процессе работы над проектом была проведена тщательная работа по проектированию и согласованию схемы базы данных с заказчиком. В результате этой работы была определена структура хранения данных, охватывающая различные аспекты управления персоналом. Схема базы данных включает в себя информацию о сотрудниках, должностях, складах, сменах и учете рабочего времени. Этот детальный подход позволил создать основу для надежного и эффективного управления данными, необходимыми для работы системы.

Помимо разработки схемы базы данных, были установлены основные шаблоны интерфейса приложения. Эти шаблоны обеспечивают единый дизайн и консистентность пользовательского опыта, что является важным аспектом для удобства использования системы. Единый дизайн позволяет пользователям быстрее адаптироваться к новому интерфейсу и эффективно использовать его возможности.

На серверной части приложения было принято решение использовать подход Domain-driven design (DDD). Основной целью этого подхода является создание четкой модели предметной области, которая выражается в коде. DDD помогает сосредоточиться на основных бизнес-объектах и процессах, обеспечивая их точное отражение в программной системе. Это позволяет создать более понятное и поддерживаемое приложение, которое отвечает потребностям бизнеса и облегчает дальнейшее развитие и масштабирование системы.

4. РЕАЛИЗАЦИЯ

4.1. Реализация серверной части приложения

Реализация серверной части приложения была начата с создания проектов таким образом, чтобы можно было использовать приведенный выше архитектурный подход.

Следующим этапом было вычленение основных сущностей проекта – тех, что являются непосредственными участниками бизнес-процессов –, чтобы в дальнейшем была возможность смоделировать их поведение в ПО. Примерами таких сущностей могут послужить: смены (`Shift`), сотрудники (`Employee`), склад (`Stock`) и т. д. В качестве примера такой модели может послужить `WorkPlan` (листинг 1).

Листинг 1 – Пример модели `WorkPlan`

```
public class WorkPlan
{
    public Guid WorkPlanId { get; set; }
    public int Month { get; set; }
    public int Year { get; set; }
    public Guid EmployeeId { get; set; }
    public int NumberOfDayShifts { get; set; }
    public int NumberOfHoursPerDayShift { get; set; }
    public int NumberOfNightShifts { get; set; }
    public int NumberOfHoursPerNightShift { get; set; }
    [JsonIgnore]
    public Employee? Employee { get; set; }
    [JsonIgnore]
    public virtual ICollection<Accounting> Accountings { get; } = new
List<Accounting>();
}
```

После того, как были выявлены основные участники бизнес-процессов, была выделена информация, которую эти сущности должны содержать. Так как это относится непосредственно и к тем данным, которые необходимо было хранить, было принято решение приступить к разработке слоя Infrastructure.

В соответствующий проект были установлены все необходимые пакеты для работы с EF Core и PostgreSQL. После данного этапа был описан контекст для того, чтобы появилась возможность создать миграцию.

Миграции – это процесс управления изменениями структуры базы данных в проекте, который использует EF Core. Они позволяют автоматически создавать и применять изменения в схеме базы данных, чтобы она соответствовала моделям данных в коде приложения. Миграции помогают разработчикам отслеживать изменения в структуре базы данных, облегчая процесс обновления и развертывания приложений.

В листинге 2 предоставлен фрагмент кода с описание таблиц миграции.

Листинг 2 – Описание создаваемых таблиц

```
public virtual DbSet<Employee> Employees { get; set; } = null!;
    public virtual DbSet<EmployeeHistory> EmployeeHistories { get; set; } =
null!;
    public virtual DbSet<Position> Positions { get; set; } = null!;
    public virtual DbSet<Stock> Stocks { get; set; } = null!;
    public virtual DbSet<Shift> Shifts { get; set; } = null!;
    public virtual DbSet<ShiftInfo> ShiftInfos { get; set; } = null!;
    public virtual DbSet<ShiftHistory> ShiftHistories { get; set; } =
null!;
    public virtual DbSet<Mark> Marks { get; set; } = null!;
    public virtual DbSet<WorkPlan> WorkPlans { get; set; } = null!;
    public virtual DbSet<Accounting> Accountings { get; set; } = null!;
    public virtual DbSet<AccountingHistory> AccountingHistories { get; set;
} = null!;
    public virtual DbSet<CalculationFormulas> CalculationFormulas { get;
set; } = null!;

    public virtual DbSet<VisitMark> VisitMarks { get; set; } = null!;
    public virtual DbSet<VisitMarkHistory> VisitMarksHistory { get; set; }
= null!;
    public virtual DbSet<VisitDetails> VisitDetails { get; set; } = null!;
```

Следующим этапом реализации стало описание конечных точек для API приложения. был создан новый проект ASP.NET Core Web API. В процессе настройки проекта в файле `Startup.cs` были добавлены необходимые сервисы, такие как контроллеры и Swagger [9] для документирования API. Конфигурация также включала настройку middleware-компонентов для обработки запросов, маршрутизации и авторизации. Пример контроллера предоставлен в листинге 3.

Листинг 3 – `AccountingController`

```
[ApiController]
[Route("api/[controller]")]
[Authorize]
public class AccountingController : ControllerBase
```

```
{
    // Регистрация зависимостей
    private readonly IAccountingService _accountingService;

    // Внедрение зависимостей
    public AccountingController(IAccountingService accountingService)

    // Получение учета заработных плат
    [HttpGet]
    public async Task<IActionResult> GetAccounting([FromQuery] int year,
[FromQuery] int month, [FromQuery] int stockId)

    // Обновление учета заработных плат
    [HttpPatch]
    public async Task<IActionResult> UpdateAccountings([FromBody] List<Up-
dateAccountingRequest> request)
}
```

Также была настроена поддержка Swagger, что позволило автоматически генерировать документацию для API. Это облегчило тестирование и интеграцию, предоставляя удобный интерфейс для взаимодействия с конечными точками.

Завершающим этапом разработки серверной части стала реализация слоя application, который отвечает непосредственно за моделирование бизнес-процессов. Была создана библиотека классов, в которую был установлен пакет AutoMapper [10]. Это инструмент, который автоматизирует процесс сопоставления (маппинга) данных между объектами различных типов. Он особенно полезен при работаете с различными уровнями приложения, такими как слой представления, слой бизнес-логики и слой доступа к данным AutoMapper позволяет легко сопоставить объекты, передаваемые между этими уровнями, без необходимости писать большое количество boilerplate-кода.

4.2. Реализация клиентской части приложения

Первым действием после создания проекта стало конфигурирование маршрутизации, которая управляет переходами между различными страницами сайта. Для начала были импортированы необходимые модули и компоненты: `Routes` и `Route` из `react-router-dom`, а также ряд других компонентов, таких как `RequireAuth`, `Layout`, и страницы, такие как

AccountingPage, AttendancePage, EmployeesPage, ErrorPage, HomePage, LoginPage, PositionsPage и ShiftsPage.

Для защиты маршрутов был использован компонент RequireAuth (листинг 4), который проверяет права доступа пользователя. В зависимости от уровня доступа пользователя отображаются или скрываются определенные страницы. Например, главная страница (HomePage) и другие функциональные страницы, такие как EmployeesPage, PositionsPage, ShiftsPage, AttendancePage и AccountingPage, защищены и доступны только авторизованным пользователям.

Листинг 4 – Hook RequireAuth

```
const RequireAuth = ({ allowedPages }) => {
    const { userData } = useAuth();
    const location = useLocation();

    if (userData) {
        return Object.keys(userData).some(role => (allowedPages.in-
cludes(role) && userData[role])) ? <Outlet/> : <Navigate to='/login' state={{
from: location }}/>;
    }

    return <Navigate to='/login' state={{ from: location }}/>;
}
```

Следующим этапом была реализация авторизации с помощью JWT [11]. Для начала пользователя необходимо аутентифицировать. Для этого фронтенд отсылает данные, введенные пользователем на сервер, и ждет ответа – валидны они или нет. При успешной аутентификации сервер возвращает access и refresh токены. Access токен используется для доступа к защищенным ресурсам, а refresh токен – для обновления access токена. Оба токена сохраняются в браузере.

При каждом запросе к защищенным ресурсам фронтенд включает access токен в заголовок Authorization. Если access токен истек, фронтенд отправляет запрос на сервер, используя refresh токен, для получения нового access токена. Сервер возвращает новый access токен, и цикл продолжается. Реализация представлена в листинге 5.

Листинг 5 – Объявление перехватчика

```
AxiosInstance.interceptors.response.use(config => {
    return config;
}, async error => {
    const initialRequest = {...error.config};
    initialRequest._isRetry = true;

    if (error.response.status === 401 && error.config && !error.config._is-
Retry) {
        if (error.response.data === 'Invalid refresh token') {
            localStorage.clear();

            return window.location = '/login';
        }
        else {
            try {
                let oldToken = localStorage.getItem('jwtToken');

                const response = await AxiosInstance.post('/LogIn/refresh-
token', oldToken, { withCredentials: true })

                if (response.status === 200) {
                    localStorage.setItem('jwtToken', response.data);
                    return AxiosInstance.request(initialRequest);
                }
                else {
                    console.log('Неудачное обновление токена', response)
                }
            }
            catch (error) {
                if (!error?.response) {
                    console.log('Сервер не отвечает.');
                }
                else {
                    console.log('Запрос был прерван:', error.message);
                }
            }
        }
    }

    throw error;
})
```

Завершающим этапом стала верстка страниц и компонентов. Для упрощения и ускорения процесса верстки было принято решение использовать библиотеку Tailwind [12]. Эта утилитарная CSS-библиотека позволила быстро применять стили к элементам, используя заранее определенные классы. Благодаря Tailwind удалось добиться консистентности в дизайне и гибкости в настройке стилей, что значительно сократило время разработки. Кроме того, библиотека уменьшила объем собственного CSS-кода, облегчая поддержку и масштабирование проекта. Tailwind также позволил легко ин-

тегрировать стилизацию с существующими компонентами, что способствовало созданию отзывчивого и привлекательного интерфейса. Скриншоты предоставлены в приложении.

4.3. Тестирование приложения

В ходе разработки API были разработаны и проведены комплексные тесты для проверки функциональности и стабильности работы системы. Проведенные тесты относятся к виду функционального тестирования.

В качестве основного инструментом тестирования было выбрано приложение Postman [13].

Тестирование проводилось в соответствии с предложенной методологией, при которой каждый из тестовых сценариев включал в себя конкретные шаги для воспроизведения действий пользователя, а также определенный ожидаемый результат. В процессе тестирования внимание было уделено как отдельным функциям, так и взаимодействию между различными компонентами системы.

Процесс тестирования осуществлялся с использованием различных входных данных, включая граничные значения и невалидные данные, чтобы оценить устойчивость и надежность системы в различных условиях, а также чтобы выявить и устранить возможные недостатки. Примеры тестов приведены в таблице 1.

Таблица 1 – Тестирование системы

№	Название теста	Шаги	Ожидаемый результат	Тест пройден?
1	Авторизация	Отправить запрос с методом POST на uri /api/login с паролем сотрудника в теле	Пользователю должен вернуться JWT токен, а в базу данных и cookie браузера занестись refresh-токен и время его истечения.	Да
2	Добавление сотрудника в систему	Отправить запрос с методом POST на uri /api/Employee со всеми необходимыми данными о сотруднике в теле	Клиенту возвращается ответ со статусом 200, в теле которого содержится информация о сотруднике, которую только что внесли в систему. В базе данных появился новый сотрудник	Да

Продолжение таблицы 1

№	Название теста	Шаги	Ожидаемый результат	Тест пройден?
3	Получение информации о всех сотрудниках, зарегистрированных в системе	Отправить запрос с методом GET на uri `/api/Employee`	Клиенту возвращается ответ со статусом 200, в теле которого содержится краткая информация о всех сотрудниках, зарегистрированных в системе.	Да
4	Некорректная попытка обновления информации о сотруднике	Отправить запрос с методом PUT на uri /api/Employee/{employeeId} со всеми необходимыми данными о сотруднике в теле и несуществующим id сотрудника	Клиенту возвращается ответ со статусом 404 и сообщением о том, что заданного сотрудника не существует.	Да
5	Создание рабочего плана для сотрудника	Отправить запрос с методом POST на uri /api/workplan с необходимыми параметрами и данными о сменах и количестве часов в них в теле.	Клиенту возвращается ответ со статусом 200. В базе данных появляется запись о новом рабочем плане за заданный месяц.	Да
6	Открытие смены	Отправить запрос с методом POST на uri /api/shift/open с данными о том на каком складе открыта смен, дневная она или ночная и списком сотрудников, которые должны заступить на смену.	Клиенту возвращается ответ со статусом 200 и id открытой смены. В базе данных появляется запись о новой открытой смене.	Да
7	Закрытие смены	Отправить запрос с методом POST на uri /api/shift/close с данными о том сколько часов отработали сотрудники на смене.	Клиенту возвращается ответ со статусом 200. Смена закрывается и сохраняется в таблицу с историями.	Да

Окончание таблицы 1

№	Название теста	Шаги	Ожидаемый результат	Тест пройден?
8	Закрытие смены с неотмеченным сотрудником	Отправить запрос с методом POST на uri /api/shift/close с данными о том сколько часов отработали сотрудники на смене, но не отмечать присутствие сотрудника перед этим.	Клиенту возвращается ответ со статусом 500 и сообщением, что сотрудник из списка не был отмечен.	Да
9	Получение посещаемости склада за выбранный месяц	Отправить запрос с методом GET на uri /api/attendance с данными о том за какой месяц, и на каком складу мы хотим узнать посещаемость.	Клиенту возвращается ответ со статусом 200 и тело, в котором содержится краткая информация о сотруднике и отработанные им смены.	Да
10	Удобство работы с экраном формата large (>992px)	Зайти в инструменты разработчика в браузере и выставить соответствующие параметры отображения	Пользовательский интерфейс остается удобным для использования	Да

4.4. Подготовка к развертыванию приложения

Для более удобного развертывания приложения использовались Docker [14] (листинги 7 и 8) и Docker Compose [15] (листинг 6).

Docker предоставляет возможность контейнеризации, что позволяет упаковать приложение и все его зависимости в единый контейнер. Это гарантирует, что приложение будет работать одинаково в любой среде, будь то локальная машина разработчика, тестовый сервер или продакшн. Контейнеризация с Docker упрощает управление зависимостями и изолирует приложение от специфики операционной системы, что способствует более стабильной и предсказуемой работе.

Docker Compose дополнительно упрощает процесс развертывания, позволяя описывать многоконтейнерные приложения с помощью простого YAML-файла. В этом файле можно указать все сервисы, которые должны

быть запущены, их зависимости, сеть и объемы. Это значительно облегчает настройку и запуск приложений, состоящих из нескольких компонентов, таких как базы данных, кэши и веб-сервисы. Использование Docker Compose ускоряет процесс развертывания и упрощает его, позволяя запускать все необходимые контейнеры одной командой. Это не только повышает производительность работы, но и снижает вероятность ошибок, связанных с ручной настройкой окружения.

Листинг 6 – Содержимое файла `docker-compose.yml` для серверной части приложения

```
version: '3'

services:

 aowh-test:
  build:
   context: source/API.Apis/bin/Debug/net7.0
   dockerfile: Dockerfile
  hostname: aowh-test
  container_name: aowh-test
  restart: always
  ports:
   - 8089:80
  networks:
   - app

networks:
 app:
  driver: bridge
```

Листинг 7 – DockerFile для серверной части приложения

```
FROM mcr.microsoft.com/dotnet/aspnet:7.0
WORKDIR /app
ADD . .
ENTRYPOINT ["dotnet", "${runningFileName}"]
```

Для клиентской части понадобился только файл Docker. Он приведен в листинге ниже.

Листинг 8 – DockerFile для клиентской части приложения

```
FROM nginx:stable
WORKDIR /web
COPY . .
COPY nginx.conf /etc/nginx/nginx.conf
ENTRYPOINT ["nginx", "-g", "daemon off;"]
```

Вывод по четвертой главе

В ходе проекта была разработана и внедрена согласованная с заказчиком схема базы данных, которая охватывает различные аспекты управления персоналом. Эта схема включает данные о сотрудниках, их должностях, складах, сменах и учете рабочего времени. Тщательное проектирование базы данных позволило создать структурированное и эффективное хранилище данных, отвечающее всем требованиям заказчика.

Одновременно с этим было реализовано и подготовленно к развертыванию на серверах клиентское приложение, шаблон интерфейса которого был согласован с заказчиком. Шаблонирование обеспечивают единый и удобный дизайн, способствующий положительному пользовательскому опыту. Благодаря этому пользователи могут легко ориентироваться в приложении и эффективно использовать его функциональные возможности.

Для серверной части проекта был применен подход Domain-driven design (DDD). Этот подход позволил создать четкую и структурированную модель предметной области, которая отражает все бизнес-процессы и объекты, необходимые для работы системы. Применение DDD обеспечило логическую целостность и ясность кода, что значительно упрощает его поддержку и дальнейшее развитие.

В результате всех проведенных работ было создано функциональное, надежное и производительное приложение, полностью соответствующее всем требованиям заказчика. Новый инструмент управления персоналом поможет компании «Луч» эффективно контролировать посещаемость и качество работы сотрудников, обеспечивая тем самым улучшение общего управления и производительности компании.

ЗАКЛЮЧЕНИЕ

В рамках данной работы было разработано клиентское и серверное приложения транспортной компании для учета заработной платы и контроля качества работы сотрудников складов.

1. Был проведен анализ предметной области и изучены существующие решения, связанные с учетом заработной платы и контролем качества работы сотрудников.

2. Была разработана архитектура серверного приложения, обеспечивающая его масштабируемость и простоту внесения изменений в будущем, и шаблон графического интерфейса пользователя.

3. Были реализованы бэкенд и фронтенд, приложения подготовлены к внедрению.

4. Было проведено тестирование приложения, включая проверку корректности расчетов и работы пользовательского интерфейса.

В компании принято решение о внедрении разработанного продукта. В частности, был утвержден акт о промышленном внедрении.

В дальнейшем планируется использование технологий распознавания лиц, чтобы идентификация сотрудников происходила в автоматическом порядке, что позволит повысить степень контроля на предприятии и снимет с сотрудников необходимость отмечаться самостоятельно.

ЛИТЕРАТУРА

1. Официальный сайт проекта Workday. [Электронный ресурс] URL: https://www.workday.com/ (дата обращения: 22.03.2024 г.).
2. Официальный сайт проекта SAP SuccessFactors. [Электронный ресурс] URL: https://www.sap.com/cis/products/hcm/talent-management.html (дата обращения: 22.03.2024 г.).
3. Официальный сайт проекта Kronos. [Электронный ресурс] URL: https://www.ukg.com/ (дата обращения: 22.03.2024 г.).
4. Официальный сайт проекта TSheets. [Электронный ресурс] URL: https://quickbooks.intuit.com/time-tracking/ (дата обращения: 22.03.2024 г.).
5. Официальная документация по C#. [Электронный ресурс] URL: https://learn.microsoft.com/en-us/dotnet/csharp/ (дата обращения: 15.02.2024 г.).
6. Официальная документация PostgreSQL. [Электронный ресурс] URL: https://www.postgresql.org/docs/ (дата обращения: 15.02.2024 г.).
7. Официальная документация ReactJS [Электронный ресурс] URL: https://react.dev/ (дата обращения: 22.03.2024 г.).
8. Эванс Э. Предметно-ориентированное проектирование (DDD): структуризация сложных программных систем. М.: Вильямс, 2011. – 433 с.
9. Официальный сайт проекта Swagger. [Электронный ресурс] URL: https://swagger.io/ (дата обращения: 22.03.2024 г.).
10. Официальный сайт проекта AutoMapper. [Электронный ресурс] URL: https://automapper.org/ (дата обращения: 22.03.2024 г.).
11. Официальный сайт проекта JWT. [Электронный ресурс] URL: https://jwt.io/ (дата обращения: 22.03.2024 г.).
12. Официальный сайт проекта Tailwind. [Электронный ресурс] URL: https://tailwindcss.com/ (дата обращения: 22.03.2024 г.).
13. Официальный сайт проекта Postman. [Электронный ресурс] URL: https://www.postman.com/ (дата обращения: 22.03.2024 г.).

14. Официальный документация Docker. [Электронный ресурс] URL: https://docs.docker.com/manuals/ (дата обращения: 22.03.2024 г.).

15. Официальная документация Docker Compose. [Электронный ресурс] URL: https://docs.docker.com/compose/ (дата обращения: 22.03.2024 г.).

ПРИЛОЖЕНИЕ. Скриншоты интерфейса

Скриншоты интерфейса приведены на рисунках 1–5.

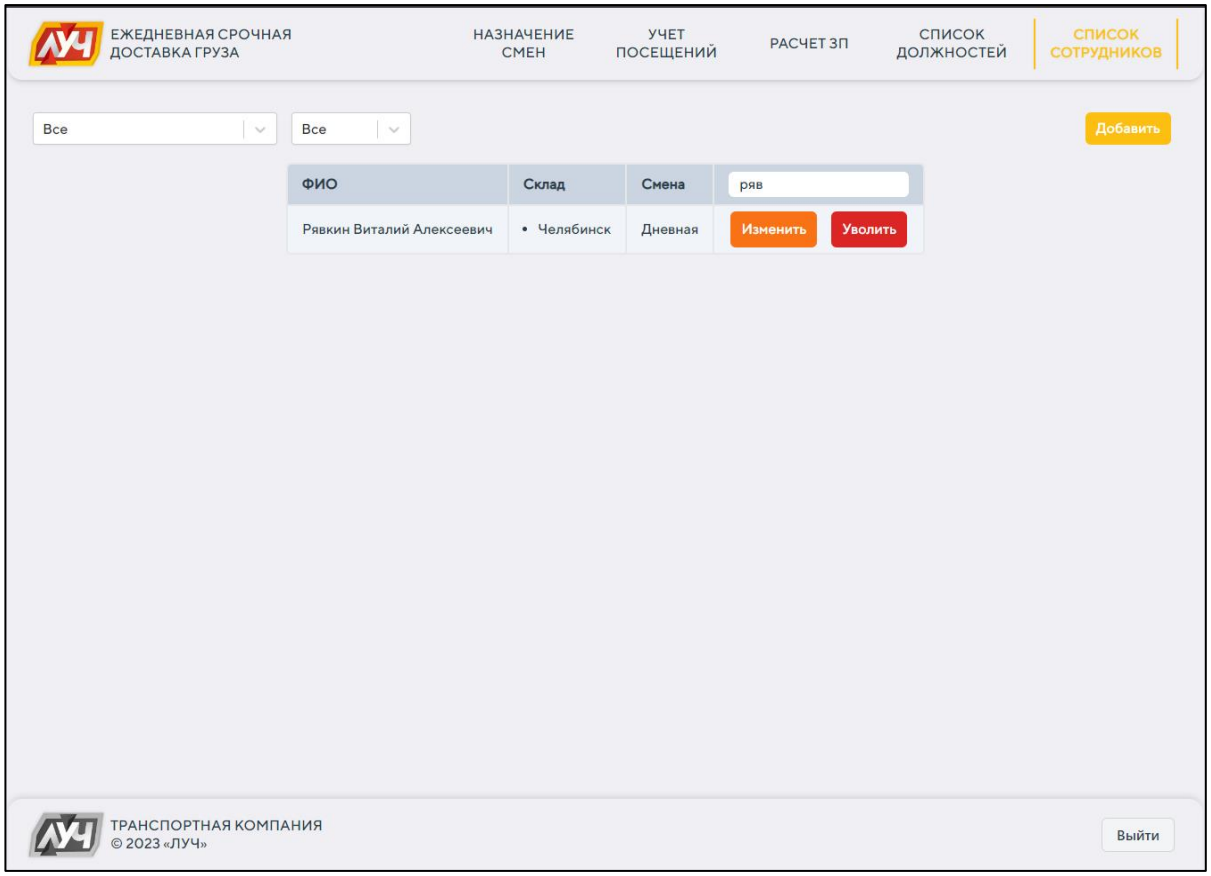
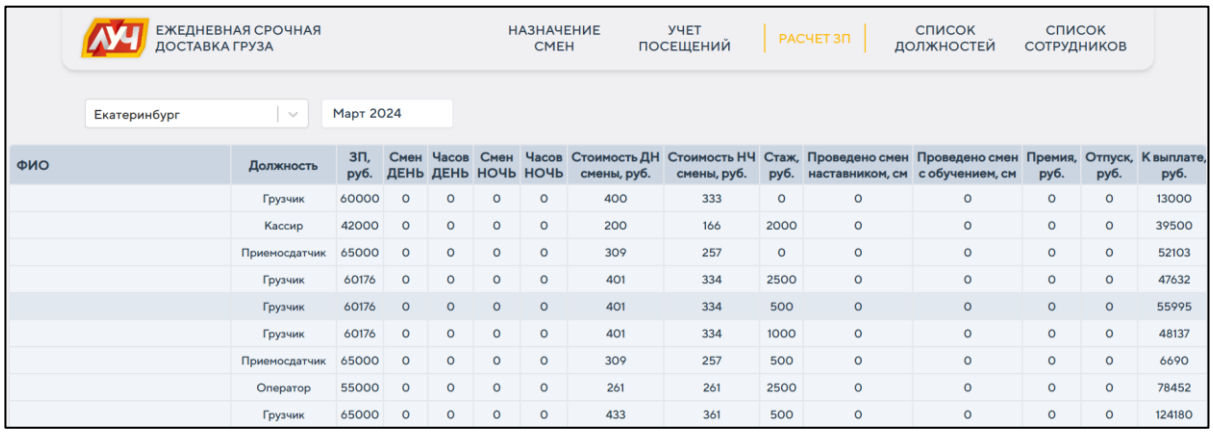


Рисунок 1 – Список сотрудников



ФИО	Должность	ЗП, руб.	Смен ДЕНЬ	Часов ДЕНЬ	Смен НОЧЬ	Часов НОЧЬ	Стоимость ДН смены, руб.	Стоимость НЧ смены, руб.	Стаж, руб.	Проведено смен наставником, см	Проведено смен с обучением, см	Премия, руб.	Отпуск, руб.	К выплате, руб.
	Грузчик	60000	0	0	0	0	400	333	0	0	0	0	0	13000
	Кассир	42000	0	0	0	0	200	166	2000	0	0	0	0	39500
	Приемосдатчик	65000	0	0	0	0	309	257	0	0	0	0	0	52103
	Грузчик	60176	0	0	0	0	401	334	2500	0	0	0	0	47632
	Грузчик	60176	0	0	0	0	401	334	500	0	0	0	0	55995
	Грузчик	60176	0	0	0	0	401	334	1000	0	0	0	0	48137
	Приемосдатчик	65000	0	0	0	0	309	257	500	0	0	0	0	6690
	Оператор	55000	0	0	0	0	261	261	2500	0	0	0	0	78452
	Грузчик	65000	0	0	0	0	433	361	500	0	0	0	0	124180

Рисунок 2 – Расчет заработных плат

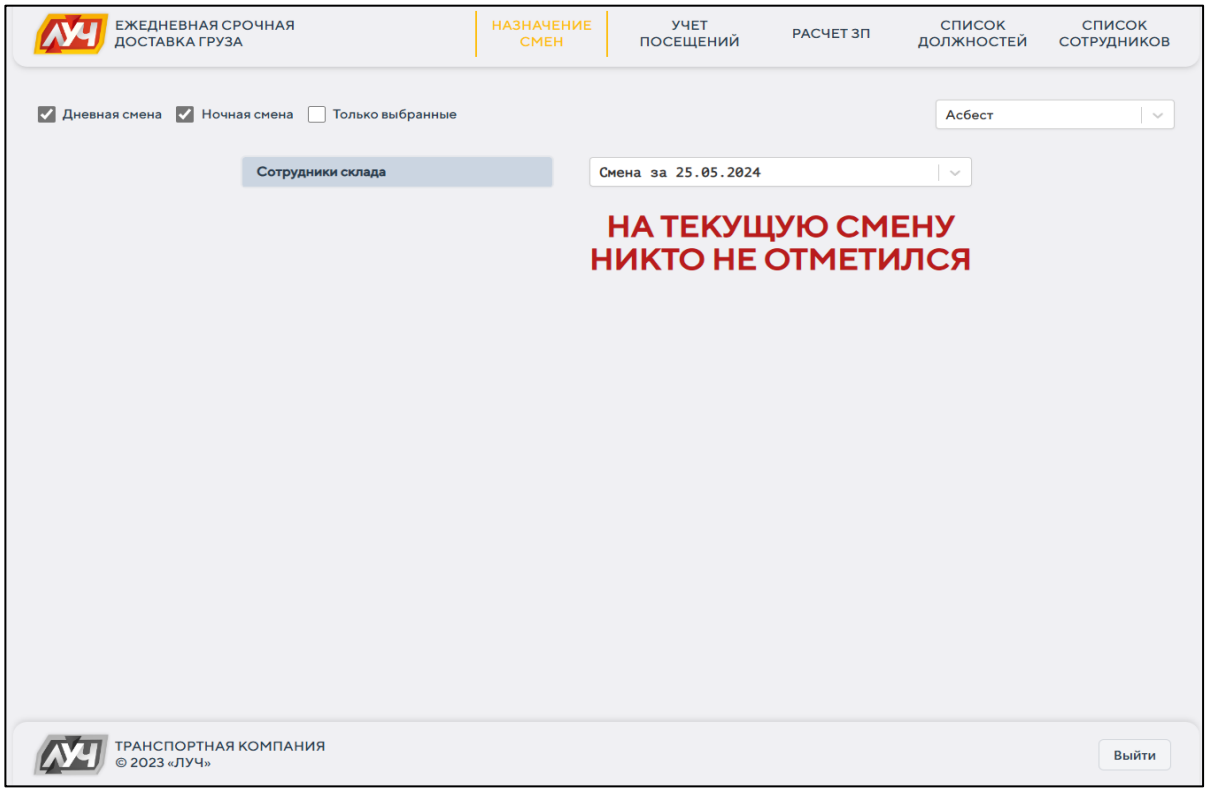
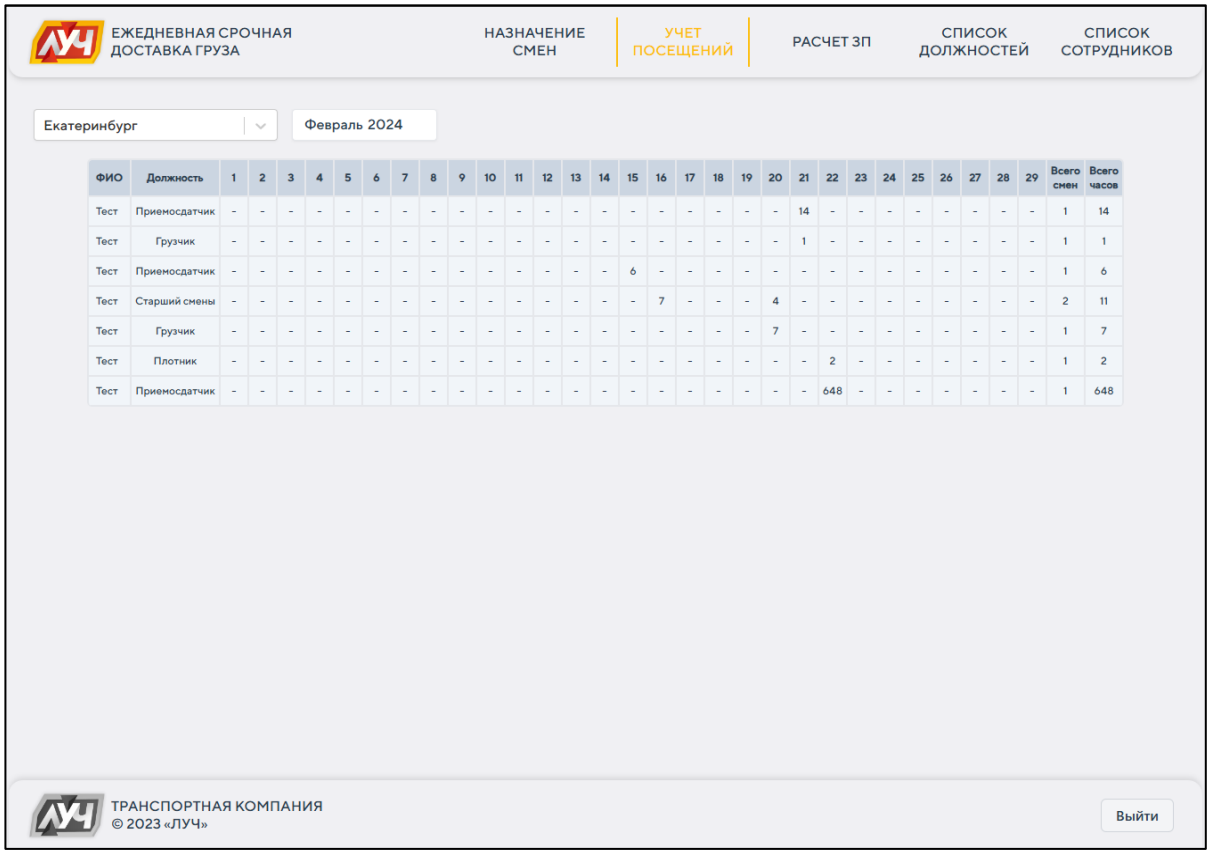


Рисунок 3 – Назначение смен



ФИО	Должность	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	Всего смен	Всего часов
Тест	Приемосдатчик	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	14	-	-	-	-	-	-	-	-	1	14
Тест	Грузчик	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	1	1
Тест	Приемосдатчик	-	-	-	-	-	-	-	-	-	-	-	-	-	-	6	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	6
Тест	Старший смены	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	7	-	-	-	4	-	-	-	-	-	-	-	-	-	2	11
Тест	Грузчик	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	7	-	-	-	-	-	-	-	-	-	1	7
Тест	Плотник	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	-	1	2
Тест	Приемосдатчик	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	648	-	-	-	-	-	-	-	1	648

Рисунок 4 – Учет посещений

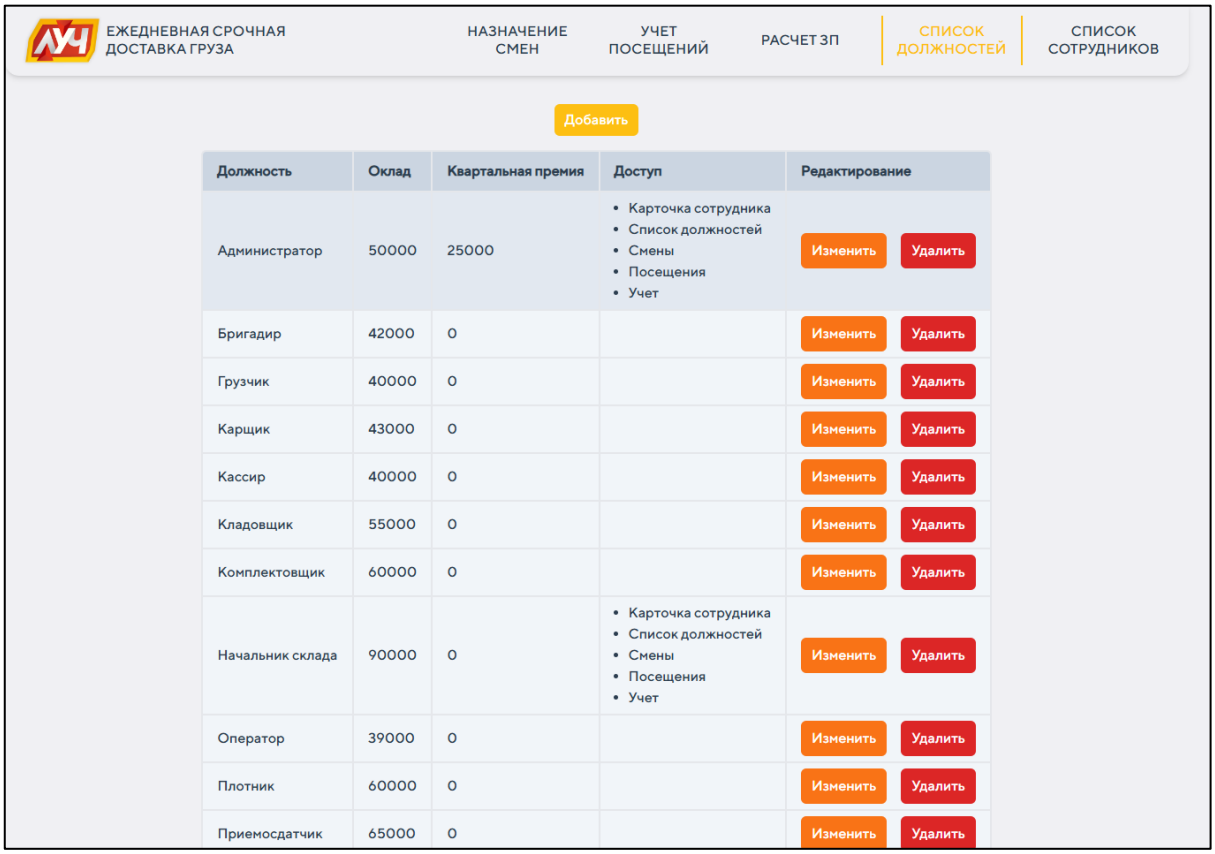


Рисунок 5 – Список должностей