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

Кафедра Прикладная математика и информатика
(наименование)

02.03.03 Математическое обеспечение и администрирование информационных систем
(код и наименование направления подготовки / специальности)

Мобильные и сетевые технологии
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Разработка и администрирование web-приложения салона красоты на основе модульного проектирования»

Обучающийся Э. Я. Магеррамов
(Инициалы Фамилия) (личная подпись)

Руководитель к.ф.-м.н. О.В. Лелонд
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Консультант к.п.н., доцент С.А. Гудкова
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2024

Аннотация

Название выпускной работы: "Разработка и администрирование web-приложения салона красоты на основе модульного проектирования".

Работа выполнена в объеме 90 страниц, включая 30 иллюстраций и 4 таблицы. Структура работы включает введение, три главы, заключение, список используемой литературы и приложения.

Тема работы связана с разработкой и администрированием веб-приложения для салона красоты с учетом современных технологий и требований рынка услуг. Работа обосновывает актуальность данной темы в контексте повышения эффективности управления салоном красоты и улучшения опыта клиентов.

Целью работы является создание функционального веб-приложения, способного предоставить клиентам салона красоты удобный доступ к информации о услугах, мастерах и местоположении, а также обеспечить администраторам возможность эффективного управления заявками и статистическими данными.

В результате работы было разработано веб-приложение, которое обеспечивает клиентам удобный доступ к информации о салоне красоты и позволяет отправлять заявки на услуги. Администраторам предоставлена админ-панель для управления заявками и мониторинга статистических данных.

Общая структура работы включает анализ объекта автоматизации, проектирование архитектуры приложения, реализацию функциональных и нефункциональных требований, а также тестирование приложения на функциональность и производительность.

В заключении подведены итоги проведенной работы и обсуждаются ее результаты. Разработанное веб-приложение является эффективным инструментом для салона красоты, способствующим улучшению качества обслуживания клиентов и оптимизации бизнес-процессов.

Abstract

The title of the graduation work is "The development and administration of a web application for a beauty salon based on modular design".

The graduation work consists of an introduction, three chapters, a conclusion, a list of references, and appendices, encompassing a total of 91 pages with 30 illustrations and 4 tables.

The key issue addressed in the graduation work is the development and administration of a web application for a beauty salon, aimed at enhancing management efficiency and improving the customer experience by leveraging modern technologies and meeting market service requirements.

The aim of the work is to provide detailed information on the development of a functional web application that offers beauty salon customers convenient access to information about services, stylists, and locations, while enabling administrators to effectively manage service requests and monitor statistical data.

The graduation work can be divided into several logically connected parts: analysis of the automation object, design of the application architecture, implementation of functional and non-functional requirements, and testing of the application for functionality and performance.

As a result of the work, a web application has been developed that facilitates customer access to salon information and allows for the submission of service requests. Administrators benefit from an admin panel designed for efficient request management and statistical data monitoring.

In conclusion, the results of the work are summarized, demonstrating that the developed web application is an effective tool for the beauty salon, contributing to improved customer service quality and optimized business processes.

Оглавление

Введение

В наше время сфера услуг красоты и заботы о себе становится все более востребованной, представляя значительный сегмент рынка. Вместе с этим растет и потребность в современных технологических решениях, способных оптимизировать процессы управления салонами красоты, повышая их эффективность и конкурентоспособность. В этом контексте разработка и администрирование web-приложения для салона красоты на основе модульного проектирования представляет собой актуальную задачу, обусловленную не только повышением требований к уровню обслуживания клиентов, но и необходимостью оптимизации внутренних процессов управления.

Целью данной работы является разработка и администрирование web-приложения, способного эффективно управлять салоном красоты, предоставляя клиентам удобные инструменты для записи на услуги и редактирования своих данных, а администраторам – возможности мониторинга заявок с доступом к информации о клиентах. Для достижения данной цели необходимо решить следующие задачи:

- разработка архитектуры и функционала веб-приложения с учетом модульного проектирования;
- реализация механизмов авторизации, аутентификации пользователями;
- создание интерфейса для клиентов и администраторов с возможностью записи на услуги и управления информацией соответственно;
- интеграция базы данных для хранения информации о клиентах и заявках на услуги;
- тестирование разработанного веб-приложения для проверки его функциональности и надежности.

Объектом исследования является процесс разработки и администрирования веб-приложения для салона красоты, а его предметом – методы и технологии, используемые при создании и поддержке такого приложения.

Для достижения поставленных целей и задач будет использован комплексный подход, включающий проектирование и реализацию собственного веб-приложения, а также тестирование его работоспособности. В качестве основного метода исследования будет применен метод моделирования для проектирования архитектуры приложения с учётом модульного подхода.

Практическая значимость работы заключается в создании инструмента, способного упростить управление салоном красоты, улучшить взаимодействие с клиентами и повысить его конкурентоспособность на рынке услуг.

Решения, разработанные в рамках данной работы, будут апробированы на практике в условиях реального салона красоты, что позволит оценить их эффективность и применимость.

Общая структура работы предполагает изучение теоретических основ, анализ существующих методов и технологий, разработку и реализацию веб-приложения, а также тестирование и апробацию его функционала.

Глава 1 Сущность объекта автоматизации бизнес-процессов

1.1 Краткая характеристика объекта автоматизации

ООО «Евразэль» является салоном красоты, специализирующимся на предоставлении широкого спектра услуг по уходу за внешностью клиентов. Салон ориентирован на выполнение услуг стрижек, покрасок, маникюра, укладок, уходов для волос и макияжа. Предприятие успешно функционирует на рынке красоты и имеет устойчивую клиентскую базу благодаря высокому уровню предоставляемых услуг и профессионализму своих сотрудников.

В структуре предприятия выделяются следующие ключевые должности:

- директор: ответственен за общее управление салоном, разработку стратегии развития бизнеса и принятие ключевых решений;
- администратор: обеспечивает организацию работы салона, включая прием заказов, управление расписанием мастеров, ведение клиентской базы и обеспечение комфортного пребывания клиентов;
- мастера: специалисты по выполнению услуг стрижек, покрасок, маникюра, укладок, уходов для волос и макияжа, обладающие необходимыми навыками и квалификацией для качественного выполнения своих обязанностей.

Организационная структура ООО «Евразэль» показана на рисунке 1.

Салон красоты ООО «Евразэль» стремится к поддержанию высокого уровня обслуживания, индивидуальному подходу к каждому клиенту и современным тенденциям в области красоты и стиля.

В рамках развития и оптимизации бизнес-процессов салона красоты ООО «Евразэль» было принято решение о разработке веб-приложения, которое позволило бы упростить процесс записи на услуги, улучшить взаимодействие с клиентами и повысить эффективность работы персонала.

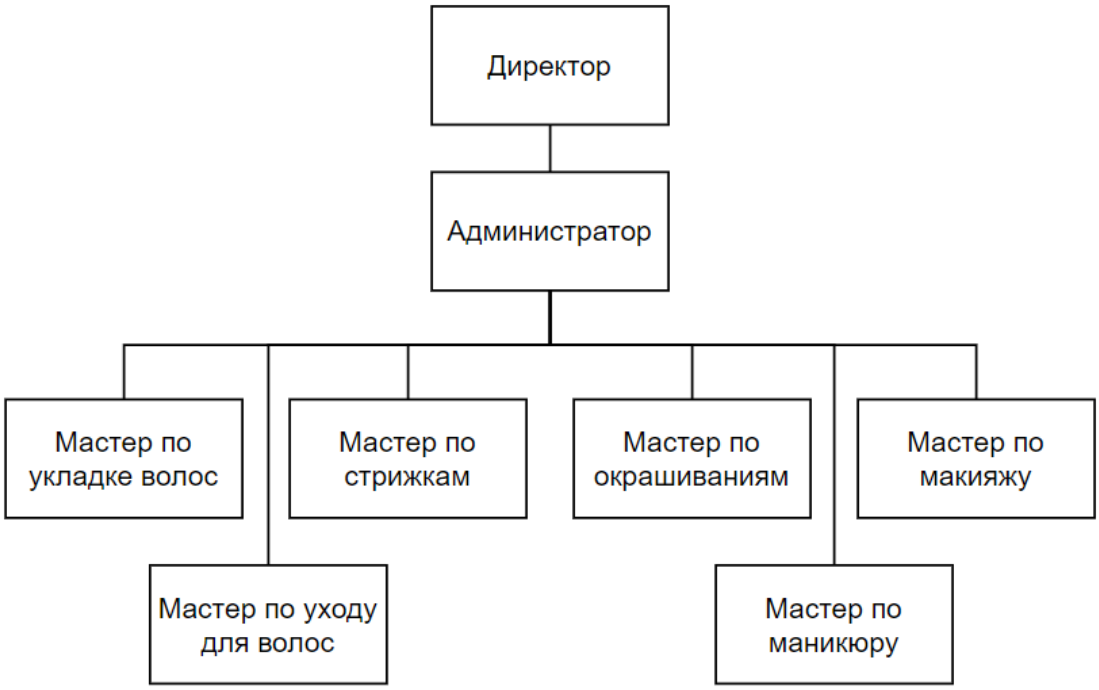

Рисунок 1 – Организационная структура предприятия

Автор данной работы принимал участие в создании и внедрении веб-приложения для салона красоты ООО «Евразэль», предназначенного для автоматизации процесса записи клиентов, управления расписанием мастеров и упрощения административных задач.

1.2 Постановка технического задания на разработку веб-приложения

Необходимо разработать веб-приложение для салона красоты, обеспечивающее клиентам возможность самостоятельного ознакомления с услугами, мастерами, местоположением и описанием салона, а также предоставить возможность отправки заявок. Для администратора создать функциональную админ-панель для управления заявками, а также мониторинга статистических данных об услугах.

Функциональные требования:

– реализация клиентской части для просмотра информации о салоне, услугах, мастерах и отправки заявок;
– авторизация и редактирование профиля пользователей;
– функционал админ-панели для просмотра и управления заявками, а также мониторинга статистических данных;
– адаптивный интерфейс для удобства использования на различных устройствах.

Нефункциональные требования:

– безопасность данных пользователей и администратора;
– производительность приложения при обработке заявок и загрузке информации;
– интуитивно понятный интерфейс для облегчения пользования приложением.

Выводы по главе 1

В рамках первой главы был проведён краткий анализ предприятия. Были выявлены проблемы, решение которых будет обеспечиваться веб-приложением. В заключении главы были сформулированы задачи на разработку веб-приложения.

Глава 2 Проектирование и архитектура веб-приложения

2.1 Выбор технологий и инструментов разработки

При создании веб-приложения особое внимание следует уделить правильному выбору технологий и инструментов [10]. Они должны соответствовать требованиям проекта, обеспечивать эффективное функционирование, а также гарантировать высокое качество и производительность конечного продукта.

В мире современных информационных технологий разработка веб-приложений занимает центральное место [12]. Постоянно растущие потребности пользователей и все более сложные требования к функциональности приложений ставят перед разработчиками ряд сложных задач. В контексте данного процесса возникают ключевые вопросы: какие технологии использовать для реализации пользовательского интерфейса (frontend) и для обеспечения серверной части (backend) веб-приложения?

Ответы на эти вопросы критически важны для успеха проекта и определяют его эффективность, безопасность и масштабируемость. Фронтенд и бэкенд представляют собой две взаимосвязанные, но в то же время различные области разработки, каждая из которых имеет свои уникальные особенности и требует специфических подходов и инструментов [1].

В данном исследовании проводится процесс выбора технологий и инструментов для разработки веб-приложения, начиная с фундаментальных вопросов о том, какие языки и фреймворки выбрать для создания пользовательского интерфейса, и заканчивая выбором инструментов для реализации бэкенда, базы данных и тестирования приложения. Результатом исследования будет обоснованный и грамотно спланированный набор технологий и инструментов, обеспечивающий успешное развитие и функционирование веб-приложения.

Фронтенд и бэкенд представляют собой две важные компоненты веб-приложений, каждая из которых отвечает за определенные аспекты пользовательского опыта и функциональности приложения.

Фронтенд, или клиентская часть, это та часть веб-приложения, которая взаимодействует непосредственно с пользователем. Она отвечает за отображение информации на экране пользователя, обеспечивая пользовательский интерфейс и интерактивность [3]. Фронтенд включает в себя HTML (язык гипертекстовой разметки), CSS (язык стилей) и JavaScript (язык программирования).

Серверная часть, или бэкенд, представляет собой невидимую для пользователя часть приложения, которая отвечает за обработку запросов от клиентской части, выполнение бизнес-логики, взаимодействие с базой данных и обеспечение безопасности и стабильности работы приложения. Разработка бэкенда может вестись на различных языках программирования, таких как JavaScript (с использованием Node.js), Python, Java или PHP, и использовать разнообразные фреймворки и инструменты для создания веб-серверов, обработки запросов и управления данными [5].

Таким образом, фронтенд и бэкенд вместе образуют полноценное веб-приложение, обеспечивая пользователей современным и функциональным интерфейсом, а также обеспечивая надежность, безопасность и эффективность работы приложения на сервере.

При выборе технологий для фронтенд-разработки важно учитывать требования проекта, а также потребности конечного пользователя. В нашем случае, где основная цель - создание простого и интуитивно понятного веб-приложения для записи клиентов в салон красоты, выбор нативных технологий HTML, CSS и JavaScript является обоснованным.

Нативные технологии HTML, CSS и JavaScript обладают неоспоримыми преимуществами, среди которых простота изучения, широкая поддержка, быстрая загрузка страниц и устойчивость к изменениям в стандартах и требованиях к веб-разработке [19]. HTML применяется для построения

структуры веб-страницы и разметки её содержимого. CSS отвечает за стилизацию и оформление элементов, обеспечивая их визуальное представление. JavaScript используется для внедрения интерактивных функций и создания динамичного поведения на странице.

Выбор нативных технологий позволяет нам гибко управлять структурой и оформлением веб-страницы, создавая удобный и интуитивно понятный пользовательский интерфейс без лишних сложностей. Более того, отсутствие зависимостей от сторонних библиотек и фреймворков упрощает процесс разработки, делает код более легким и понятным для поддержки и расширения в будущем.

Таким образом, выбор нативных технологий HTML, CSS и JavaScript обоснован для нашего проекта, где ключевыми критериями являются простота, надежность и эффективность веб-приложения.

При выборе языка программирования и фреймворка для написания бэкенд-части веб-приложения важно учитывать ряд критериев, которые помогут определить наиболее подходящий вариант. Вот несколько таких критериев:

- экосистема и сообщество: язык программирования и фреймворк должны иметь активное и развитое сообщество разработчиков. Это обеспечит доступ к обширным библиотекам, документации, обновлениям и поддержке.
- производительность: язык и фреймворк должны обеспечивать высокую производительность при обработке запросов и выполнении задач бэкенда. Это особенно важно для обеспечения отзывчивости и быстродействия веб-приложения;
- масштабируемость: выбранные технологии должны обеспечивать горизонтальное и вертикальное масштабирование приложения, позволяя ему эффективно масштабироваться с ростом нагрузки и объема данных;

– простота и удобство разработки: язык и фреймворк должны предоставлять интуитивно понятный синтаксис, хорошую документацию и инструменты разработки, что упростит процесс создания и поддержки приложения;
– безопасность: технологии должны предоставлять средства для обеспечения безопасности приложения, включая защиту от атак, управление сессиями, шифрование данных и проверку подлинности;
– гибкость и расширяемость: язык и фреймворк должны быть гибкими и легко расширяемыми, что позволит добавлять новый функционал и вносить изменения без существенных переписываний кода;
– интеграция с другими технологиями: технологии должны хорошо интегрироваться с другими сервисами и инструментами, такими как базы данных, сторонние API и инфраструктура приложения.

Вышеперечисленных критериев достаточно для того, чтобы принять обоснованное решение относительно выбора языка программирования и фреймворка для написания бэкенд-части веб-приложения.

В данном контексте были рассмотрены несколько популярных комбинаций технологий, включая Node.js + Express.js, Python + Flask и Java + Spring Boot. Каждая из этих комбинаций имеет свои преимущества и недостатки, которые необходимо учитывать при выборе наиболее подходящего решения для конкретного проекта.

В данной работе будет проведено сравнение данных технологий с учетом составленных критериев.

На основании данного сравнения будет сделан выбор технологии для разработки бэкенд-части веб-приложения, который будет оптимальным с учетом требований проекта и его целей.

Node.js + Express.js:

– экосистема и сообщество: Node.js и Express.js обладают развитыми и активными сообществами разработчиков, что обеспечивает доступ к обширным библиотекам, документации и поддержке;

- производительность: Node.js и Express.js позволяют создавать высокопроизводительные веб-приложения благодаря асинхронной обработке запросов и эффективному механизму маршрутизации;
- масштабируемость: эти технологии обеспечивают горизонтальное и вертикальное масштабирование приложений, что позволяет им масштабироваться с ростом нагрузки и объема данных;
- простота и удобство разработки: Node.js и Express.js предлагают простой и интуитивно понятный синтаксис, который упрощает процесс разработки и поддержки приложения;
- безопасность: существует множество библиотек и инструментов для обеспечения безопасности в Node.js и Express.js, что позволяет разработчикам создавать безопасные приложения;
- гибкость и расширяемость: Node.js и Express.js гибкие и легко расширяемые, что позволяет разработчикам создавать приложения различного уровня сложности и функциональности;
- интеграция с другими технологиями: эти технологии легко интегрируются с различными базами данных, сторонними API и сервисами [17].

Python + Flask:

- экосистема и сообщество: Python и Flask имеют большие и активные сообщества разработчиков, обеспечивая доступ к обширным библиотекам и ресурсам;
- производительность: Flask обеспечивает хорошую производительность, особенно при обработке HTTP-запросов и запросов к базам данных;
- масштабируемость: Flask обеспечивает горизонтальное и вертикальное масштабирование приложений, позволяя им масштабироваться с ростом нагрузки и объема данных;

– простота и удобство разработки: Python и Flask предлагают простой и понятный синтаксис, что делает разработку приложений более удобной и приятной;
– безопасность: Flask обеспечивает базовые механизмы безопасности, такие как защита от CSRF и XSS атак, но требует использования дополнительных библиотек для обеспечения полной безопасности;
– гибкость и расширяемость: Flask гибкий и легко расширяемый, что позволяет добавлять новый функционал и вносить изменения без существенных переписываний кода;
– интеграция с другими технологиями: Python и Flask хорошо интегрируются с различными базами данных, сторонними API и сервисами, обеспечивая широкие возможности для создания веб-приложений.

Java + Spring Boot:

– экосистема и сообщество: Java и Spring Boot имеют широкое сообщество разработчиков и обширную экосистему;
– производительность: Java с Spring Boot обеспечивает высокую производительность и эффективное использование ресурсов;
– масштабируемость: Spring Boot обеспечивает легкое масштабирование, что делает его подходящим для крупных приложений;
– простота и удобство разработки: Java может быть более громоздким и многословным, чем другие языки, но Spring Boot упрощает создание веб-приложений [21];
– безопасность: Spring Boot обеспечивает широкий набор инструментов для обеспечения безопасности приложений;
– гибкость и расширяемость: Spring Boot гибкий и обеспечивает хорошую расширяемость, но может потребоваться больше усилий для конфигурации;

– интеграция с другими технологиями: Java и Spring Boot хорошо интегрируются с другими технологиями, предоставляя множество инструментов для создания сложных приложений [18].

Все вышеперечисленные свойства выбранных для сравнения технологий представлены в таблице 1.

Таблица 1 – Свойства технологий разработки бэкенда

	Node.js + Express.js	Python + Flask	Java + Spring Boot
Экосистема и сообщество	+	+	+
Производительность	+	+	+
Масштабируемость	+	+	+
Простота и удобство разработки	+	+	-
Безопасность	+	-	+
Гибкость и расширяемость	+	+	-
Интеграция с другими технологиями	+	+	+

Проведенный сравнительный анализ различных комбинаций технологий для разработки бэкенд-части веб-приложения позволяет сделать обоснованный вывод о выборе наилучшего инструментария.

Node.js + Express.js являются оптимальным решением по ряду критериев. Их комбинация обеспечивает высокую производительность, эффективное использование ресурсов сервера, а также простоту и удобство разработки благодаря асинхронной обработке запросов, интуитивно понятному синтаксису и механизму маршрутизации.

Благодаря активному сообществу разработчиков, обширной документации и богатому выбору библиотек Node.js и Express.js предоставляют широкие возможности для создания сложных и масштабируемых веб-приложений. Кроме того, их гибкость и расширяемость

позволяют легко адаптировать приложение под изменяющиеся требования и добавлять новый функционал без существенных затрат.

Таким образом, исходя из проведенного анализа, выбор Node.js + Express.js для разработки бэкенд-части нашего веб-приложения является оптимальным решением, обеспечивающим надежную основу для успешной реализации проекта.

Соответственно, в качестве системы управления базами данных будет выступать PostgreSQL, так как именно данная СУБД имеет предоставляет полную поддержку и функционал для связки Node.js + Express.js.

Для выбора среды разработки, которая будет использоваться при создании веб-приложения, необходимо определить ряд критериев, учитывая требования проекта и предпочтения команды разработчиков. Вот несколько ключевых критериев, которые стоит учесть:

- функциональность: среда разработки должна предоставлять все необходимые инструменты и функциональные возможности для комфортной работы над проектом. Это включает в себя редактор кода с подсветкой синтаксиса, автоматическим дополнением, возможностью отладки, интеграцию с системами контроля версий и другими полезными функциями;
- производительность: среда разработки должна быть быстрой и отзывчивой, чтобы ускорить процесс написания кода, отладки и тестирования приложения;
- поддержка языков и технологий: важно, чтобы среда разработки поддерживала используемые технологии, языки программирования и фреймворки. Это обеспечит комфортную работу с кодом и упростит процесс разработки;
- кроссплатформенность: предпочтение должно отдаваться среде разработки, которая поддерживает максимальное количество платформ;

– сообщество и поддержка: важно, чтобы выбранная среда разработки имела активное сообщество пользователей и обеспечивала достаточную поддержку, например, через документацию, форумы, видеоуроки и т. д.;
– расширяемость и гибкость: хорошая среда разработки должна предоставлять возможности для расширения и настройки под индивидуальные потребности разработчика, например плагины, настройки интерфейса и т. д.;
– лицензия и стоимость: некоторые среды разработки могут быть платными, иметь ограничения по использованию в коммерческих проектах или требовать подписки. Важно учитывать этот аспект при выборе среды разработки.

Учитывая вышеперечисленные критерии, можно определить наиболее подходящую среду разработки для успешной реализации проекта.

В качестве сравнения были выбраны наиболее популярные среды разработки для написания веб-приложений с учётом выявленных требований к технологиям такие, как Atom, Visual Studio Code, Sublime Text и JetBrains WebStorm.

Visual Studio Code (VS Code):

– функциональность: Visual Studio Code обладает обширным набором функций, включая подсветку синтаксиса, автоматическое дополнение кода, интегрированный терминал, отладчик, систему контроля версий и многое другое. Он предоставляет разработчикам широкие возможности для комфортной и продуктивной работы;
– производительность: VS Code отличается высокой производительностью благодаря своей оптимизированной архитектуре и эффективному использованию ресурсов компьютера. Он быстро открывает и обрабатывает файлы, обеспечивая плавный и отзывчивый интерфейс;

- поддержка языков и технологий: VS Code поддерживает различные языки программирования, включая HTML, CSS, JavaScript, а также Node.js для разработки веб-приложений. Он также обладает расширяемой архитектурой, позволяя добавлять поддержку для новых языков и технологий через расширения;
- кроссплатформенность: Visual Studio Code доступен на различных операционных системах, включая Windows, macOS и Linux, что обеспечивает единый опыт разработки независимо от платформы;
- сообщество и поддержка: VS Code имеет активное сообщество разработчиков и официальную документацию, а также широкий выбор расширений, созданных сообществом, что обеспечивает поддержку и обновления;
- расширяемость и гибкость: одним из ключевых преимуществ Visual Studio Code является его высокая расширяемость и гибкость. Разработчики могут использовать разнообразные расширения, чтобы настраивать и улучшать функциональность редактора в соответствии со своими конкретными потребностями и предпочтениями;
- лицензия и стоимость: Visual Studio Code распространяется бесплатно и с открытым исходным кодом под лицензией MIT, что делает его доступным для использования без ограничений и затрат.

Atom:

- функциональность: Atom предоставляет широкий набор функций, включая подсветку синтаксиса, автоматическое дополнение кода, интегрированный терминал и возможность установки различных плагинов для расширения функциональности;
- производительность: Atom обеспечивает удовлетворительную производительность, но может быть несколько медленнее в сравнении с Visual Studio Code при работе с большими проектами и файлами;

- поддержка языков и технологий: Atom поддерживает множество языков программирования, включая HTML, CSS, JavaScript и Node.js, а также обладает возможностью расширения поддержки других языков через установку плагинов;
- кроссплатформенность: Atom также кроссплатформенный редактор и доступен на Windows, macOS и Linux;
- сообщество и поддержка: у Atom также есть активное сообщество разработчиков и многочисленные плагины, однако сообщество и обновления несколько менее активные по сравнению с Visual Studio Code;
- расширяемость и гибкость: Atom также обладает расширяемой архитектурой, позволяющей добавлять новую функциональность через плагины, но может быть несколько менее гибким и удобным в настройке по сравнению с VS Code;
- лицензия и стоимость: Atom также распространяется бесплатно и с открытым исходным кодом под лицензией MIT.

Sublime Text:

- функциональность: Sublime Text предоставляет широкий набор функций, включая подсветку синтаксиса, автоматическое дополнение кода, многооконный режим, возможность установки плагинов и многое другое;
- производительность: Sublime Text известен своей высокой производительностью и быстрым откликом интерфейса, что делает его популярным среди разработчиков;
- поддержка языков и технологий: Sublime Text также поддерживает множество языков программирования, включая HTML, CSS, JavaScript и Node.js, а также позволяет устанавливать плагины для расширения поддержки других языков;
- кроссплатформенность: Sublime Text также кроссплатформенный редактор и доступен на Windows, macOS и Linux;

– сообщество и поддержка: Sublime Text имеет небольшое, но преданное сообщество разработчиков и широкий выбор плагинов, но обновления и развитие редактора могут быть несколько медленнее по сравнению с VS Code;
– расширяемость и гибкость: Sublime Text также обладает расширяемой архитектурой, позволяющей добавлять новую функциональность через плагины, но может быть несколько менее гибким и удобным в настройке по сравнению с VS Code;
– лицензия и стоимость: Sublime Text распространяется на условиях пробной версии с ограниченной функциональностью и требует покупки лицензии для полного доступа к функциям.

JetBrains WebStorm:

– функциональность: WebStorm предлагает богатый набор функций для разработки веб-приложений, включая подсветку синтаксиса, автоматическое дополнение кода, интегрированный отладчик, систему контроля версий, инструменты для тестирования и многое другое;
– производительность: WebStorm известен своей высокой производительностью и эффективным использованием ресурсов компьютера, обеспечивая плавный и отзывчивый интерфейс даже при работе с большими проектами;
– поддержка языков и технологий: WebStorm обладает широкой поддержкой языков программирования и технологий, включая HTML, CSS, JavaScript и Node.js, а также предоставляет инструменты для работы с фреймворками и библиотеками;
– кроссплатформенность: WebStorm также кроссплатформенный редактор и доступен на Windows, macOS и Linux;
– сообщество и поддержка: WebStorm имеет активное сообщество разработчиков и официальную документацию, а также получает регулярные обновления и поддержку от JetBrains;

– расширяемость и гибкость: WebStorm предлагает возможности для настройки и расширения функциональности через плагины и настройки, обеспечивая гибкость и адаптивность под потребности разработчика;
– лицензия и стоимость: WebStorm является платным продуктом с подписочной моделью лицензирования, но предлагает бесплатный пробный период для ознакомления с функциональностью редактора.

Все вышеперечисленный свойства выбранных для сравнения сред разработки представлены в таблице 2.

Таблица 2 – Свойства сред разработки

	VS Code	Atom	Sublime Text	WebStorm
Функциональность	+	+	+	+
Производительность	+	-	+	+
Поддержка языков и технологий	+	+	+	+
Кроссплатформенность	+	+	+	+
Сообщество и поддержка	+	-	-	+
Расширяемость и гибкость	+	-	-	+
Лицензия и стоимость	+	+	-	-

Исходя из проведенного сравнительного анализа различных сред разработки, явно выделяется VS Code как наилучший выбор для разработки веб-приложений на базе HTML, CSS, JavaScript, а также для работы с Node.js и Express.js. VS Code демонстрирует превосходство по всем рассмотренным критериям, предоставляя широкие возможности функциональности, высокую производительность, обширную поддержку языков и технологий, кроссплатформенность, активное сообщество и поддержку разработчиков, а также гибкость и расширяемость. Таким образом, в качестве среды разработки и дальнейшей работы над веб-приложением будет использоваться VS Code.

2.2 Проектирование архитектуры приложения с учётом модульного подхода

При проектировании современных веб-приложений одним из ключевых аспектов является создание гибкой, масштабируемой и удобной для разработки архитектуры [2]. В данном пункте мы рассмотрим подход к проектированию приложения с использованием модульного подхода, который позволяет эффективно организовать код, улучшить его структуру и обеспечить легкость поддержки и расширения.

Модульное проектирование основано на принципе разбиения приложения на отдельные модули или компоненты, каждый из которых отвечает за определенную часть функциональности. Этот подход позволяет создавать независимые модули, которые могут быть легко повторно использованы и изменены без влияния на другие части приложения [4].

В данном пункте мы рассмотрим основные принципы модульного проектирования, его преимущества и способы применения в контексте разработки веб-приложения для салона красоты. Мы также обсудим архитектурные шаблоны и методы, которые помогут нам создать эффективную и масштабируемую архитектуру для нашего проекта.

При разработке веб-приложения для салона красоты необходимо уделить особое внимание проектированию его архитектуры с использованием модульного подхода. Этот подход позволяет создать систему, которая будет легко масштабируемой, гибкой и поддерживаемой, что является ключевым фактором для успешной разработки и эксплуатации приложения [8].

Преимущества модульного подхода:

– гибкость и расширяемость: разбиение приложения на модули позволяет добавлять новую функциональность или изменять существующую без необходимости переписывания всего приложения. Это делает код более гибким и поддерживаемым;

– отделение ответственностей: каждый модуль отвечает за конкретную функциональность или компонент приложения, что упрощает понимание его структуры и облегчает совместную работу разработчиков;
– повторное использование кода: модули могут быть легко повторно использованы в различных частях приложения или даже в других проектах, что повышает эффективность разработки и ускоряет процесс создания нового функционала;
– тестирование: модульный подход упрощает тестирование приложения, поскольку каждый модуль может быть протестирован отдельно, что обеспечивает более высокую степень надежности и качества кода [16].

Принципы проектирования модульной архитектуры:

– выделение функциональных блоков: идентификация ключевых функциональных блоков приложения и их разделение на независимые модули;
– определение интерфейсов: определение интерфейсов взаимодействия между модулями для обеспечения совместимости и упрощения интеграции;
– структурирование данных: определение способов организации данных внутри модулей для обеспечения их эффективного доступа и обработки;
– разделение на уровни: выделение уровней абстракции и разделение модулей на уровни для обеспечения логической и физической чистоты кода.

Проектирование архитектуры веб-приложения салона красоты будет осуществляться за счёт модульного подхода, учитывая разделение приложения на основные компоненты, такие как управление записями и администрирование. Каждый из этих компонентов будет представлен

отдельным модулем, что позволит нам эффективно организовать код и обеспечить легкость его поддержки и расширения.

На основе вышеизложенных принципов необходимо приступить к детальному проектированию архитектуры приложения, учитывая его специфические требования и особенности функционирования.

При разработке веб-приложения для салона красоты планируется использовать архитектурный шаблон MVC (Model-View-Controller), который является одним из наиболее распространенных подходов к построению веб-приложений. Ниже мы рассмотрим, как каждая из компонентов этого шаблона будет использоваться в нашем проекте.

Model (Модель). Модель представляет собой компонент, отвечающий за обработку данных и бизнес-логику приложения. В контексте нашего веб-приложения модель будет представлена объектами, которые отражают структуру данных о клиентах, услугах, записях и других аспектах работы салона красоты. Модель будет обеспечивать доступ к данным, их обновление и валидацию.

View (Представление). Представление отвечает за отображение данных пользователю и взаимодействие с ним. В нашем приложении представления будут представлены HTML-страницами, которые будут формировать интерфейс пользователя для работы с различными функциями салона красоты. Каждое представление будет отражать определенный аспект работы с приложением, например, форму записи клиента на услугу или список доступных услуг.

Controller (Контроллер). Контроллер является посредником между моделью и представлением, управляя потоком данных и обработкой пользовательских запросов. В нашем приложении контроллеры будут обрабатывать HTTP-запросы, полученные от клиентских браузеров, и вызывать соответствующие методы модели для получения или обновления данных. После этого контроллеры будут передавать эти данные представлениям для отображения пользователю.

Для большей наглядности схема архитектуры MVC, описанная выше, которая будет использоваться при разработке веб-приложения для салона красоты представлена на рисунке 2.

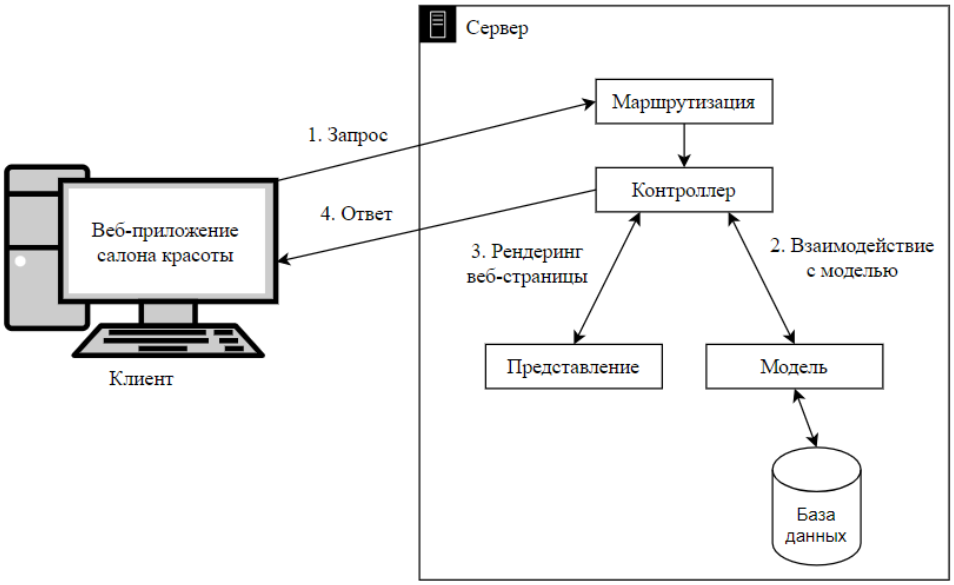

Рисунок 2 – Схема архитектуры MVC

Преимущества применения шаблона MVC:

– разделение обязанностей: использование шаблона MVC позволяет разделить логику приложения на три отдельных компонента, что упрощает его разработку и сопровождение;
– повторное использование кода: благодаря разделению на модель, представление и контроллер, код можно легко повторно использовать в различных частях приложения;
– гибкость и расширяемость: архитектурный шаблон MVC обеспечивает гибкость и легкость внесения изменений в приложение, что позволяет быстро реагировать на изменяющиеся требования и потребности пользователей;

– тестирование: каждый компонент MVC-приложения может быть протестирован независимо от других, что обеспечивает более высокую степень надежности и качества кода.

Диаграмма развёртывания представлена на рисунке 3.

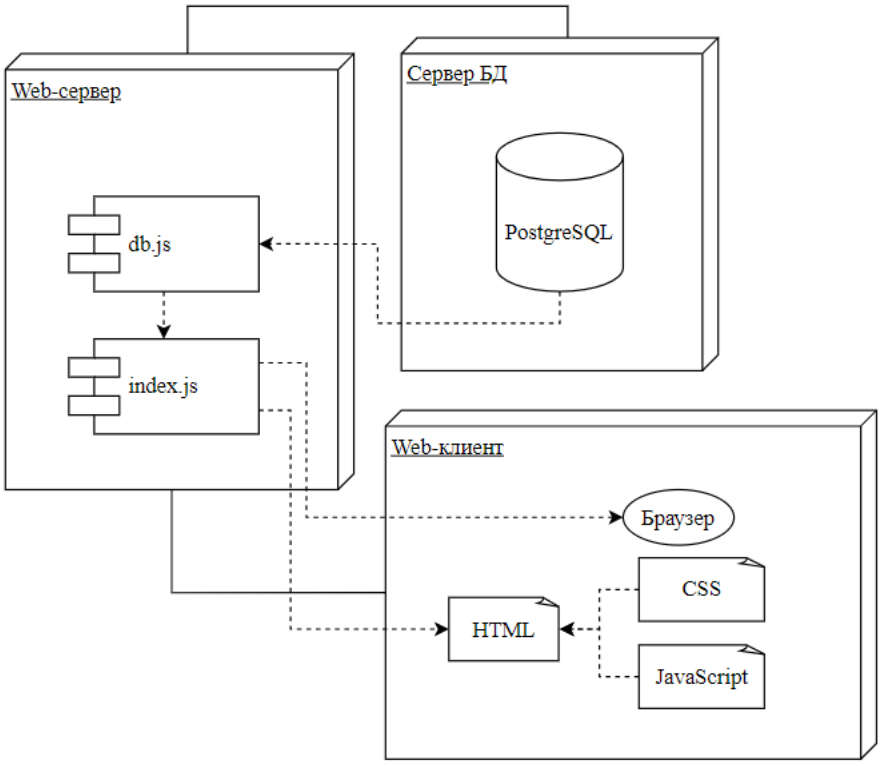

Рисунок 3 – Диаграмма развёртывания

Диаграмма развёртывания, представляющая компоненты, необходимые для функционирования веб-приложения имеет следующие элементы:

– web-клиент: компонент, представляющий интерфейс пользователя веб-приложения, который обеспечивает взаимодействие пользователя с приложением через веб-браузер. Он отображает пользовательский интерфейс, обрабатывает ввод пользователя и отправляет запросы на сервер для получения данных или выполнения операций;

– web-сервер: компонент, представляющий серверное приложение, который принимает и обрабатывает HTTP-запросы от веб-клиентов,

а затем возвращает соответствующие HTTP-ответы. Этот компонент обеспечивает связь между веб-клиентами и бизнес-логикой приложения, обрабатывая запросы, поступающие от клиентов, и управляя передачей данных между клиентом и сервером;

– сервер БД: компонент, представляющий сервер базы данных, который отвечает за хранение и управление данными, используемыми веб-приложением. Он обрабатывает запросы на чтение и запись данных, предоставляя доступ к базе данных веб-приложения, а также гарантирует сохранность и целостность данных.

Взаимодействия между компонентами позволяет веб-приложению эффективно функционировать. На основе принципов модульного подхода были спроектированы модули, работающие в плотном взаимодействии друг с другом, в то же время являющиеся независимыми с точки зрения повторного использования. Визуализация организации модулей web-сервера и зависимостей между ними представлена на рисунке 4.

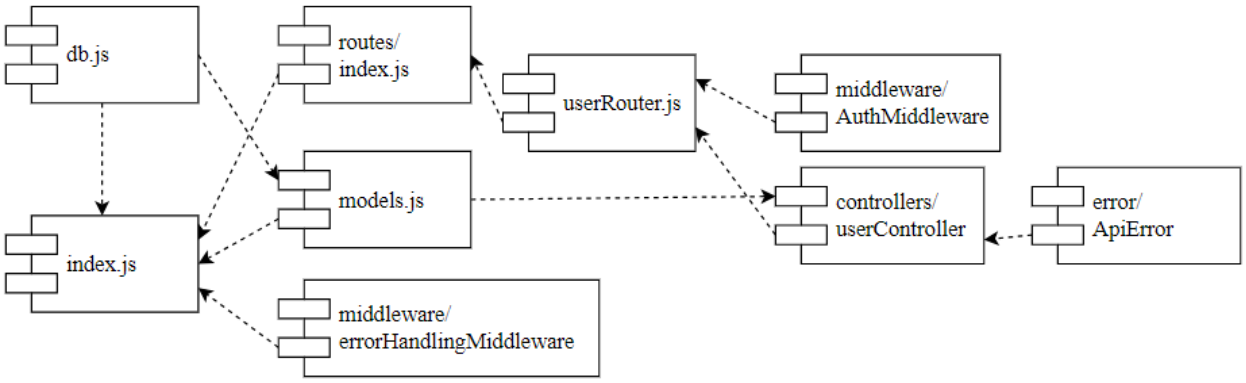

Рисунок 4 – Диаграмма модулей web-сервера

Модульный подход позволяет разбить приложение на независимые компоненты, что упрощает его разработку, тестирование и обслуживание. В процессе разработки архитектуры веб-приложения для салона красоты был использован шаблон MVC (Model-View-Controller), который разделяет

приложение на три основных компонента: модель (Model), представление (View) и контроллер (Controller), что способствует легкости поддержки и развития проекта.

2.3 Проектирование модели данных

В данном пункте рассматривается процесс проектирования структуры баз данных для веб-приложения салона красоты. Целью этапа является определение основных сущностей и их взаимосвязей, которые будут использоваться для хранения информации, необходимой для функционирования приложения.

Разработка модели данных играет важную роль в определении логической организации данных, и она основывается на анализе бизнес-процессов и требований к приложению [6]. В ходе данного этапа устанавливаются структура таблиц, их атрибуты и связи между ними, что обеспечивает эффективное хранение и обработку данных.

Концептуальная модель данных служит важным инструментом при проектировании базы данных, помогающим разработчикам понять структуру данных, определить взаимосвязи между ними и обеспечить эффективную обработку информации и её хранение.

Данная модель данных представляет собой абстрактное представление о структуре данных и их взаимосвязях на уровне высокого уровня абстракции. Она не зависит от конкретной технологии или схемы базы данных и служит для описания бизнес-объектов и их взаимосвязей.

Основные компоненты концептуальной модели данных включают:

– сущности (Entities): представляют собой основные объекты, которые будут храниться в базе данных. Каждая сущность обладает атрибутами, описывающими её характеристики;
– связи (Relationships): определяют отношения между сущностями. Связи могут быть однонаправленными или двунаправленными и

включать различные типы: один-к-одному, один-ко-многим и многие-ко-многим;

– атрибуты (Attributes): представляют собой характеристики сущностей, которые могут принимать определенные значения [15].

Концептуальная модель данных представлена на рисунке 5.

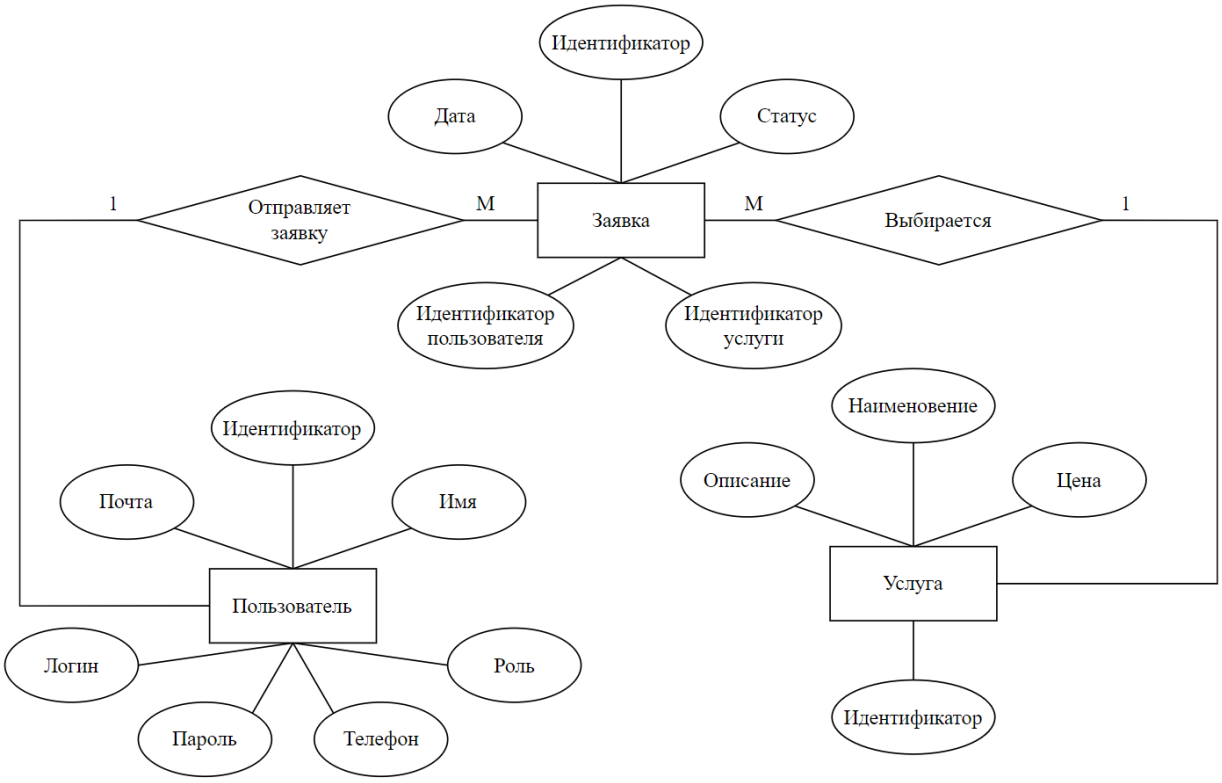

Рисунок 5 – Концептуальная модель данных

Сущности, атрибуты и связи, представленные в данных моделях, были определены после проведения исследования предметной области.

Логическая модель IDEFX – это методология проектирования баз данных, разработанная в рамках семейства IDEF (Integrated Definition for Information Modeling) [20]. IDEFX является расширением методологии IDEF1X и предназначена для создания структурной модели данных, которая описывает сущности, их атрибуты и отношения между ними на более детальном уровне, чем концептуальная модель.

Основные элементы логической модели IDEFX включают:

– сущности (Entities): представляют собой конкретные объекты или классы объектов, которые будут храниться в базе данных. Каждая сущность имеет набор атрибутов, описывающих её свойства;
– атрибуты (Attributes): характеристики сущностей, которые описывают их свойства или характеристики. Каждый атрибут имеет имя и тип данных, определяющий формат хранения значений атрибута;
– отношения (Relationships): определяют связи между сущностями. Отношения могут быть однонаправленными или двунаправленными и могут иметь различные типы, такие как один-к-одному, один-ко-многим и многие-ко-многим;
– ключи (Keys): определяют уникальные идентификаторы для сущностей. Ключи могут быть первичными (Primary Key), уникальными (Unique Key) или внешними (Foreign Key), и они играют важную роль в обеспечении целостности данных.

Логическая модель данных представлена на рисунке 6.

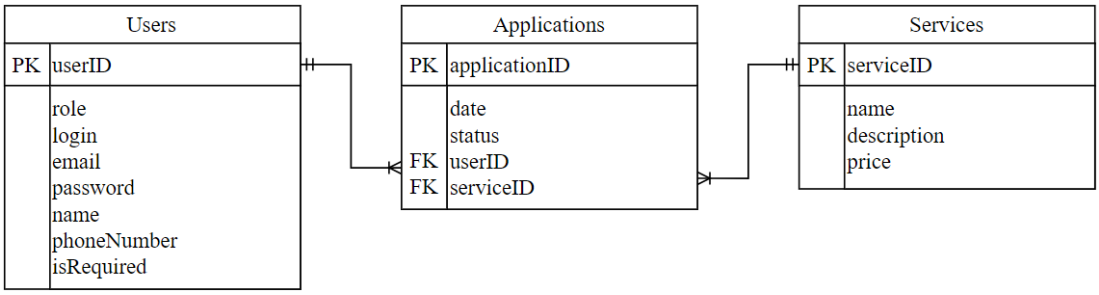

Рисунок 6 – Логическая модель данных

Физическая модель данных – это детализированная спецификация структуры базы данных на уровне, близком к реализации в конкретной системе управления базами данных (СУБД). В отличие от концептуальной и логической моделей, которые описывают данные независимо от конкретной

реализации, физическая модель фокусируется на том, как данные будут храниться, организованы и взаимодействовать с СУБД.

Физическая модель данных представлена на рисунке 7.

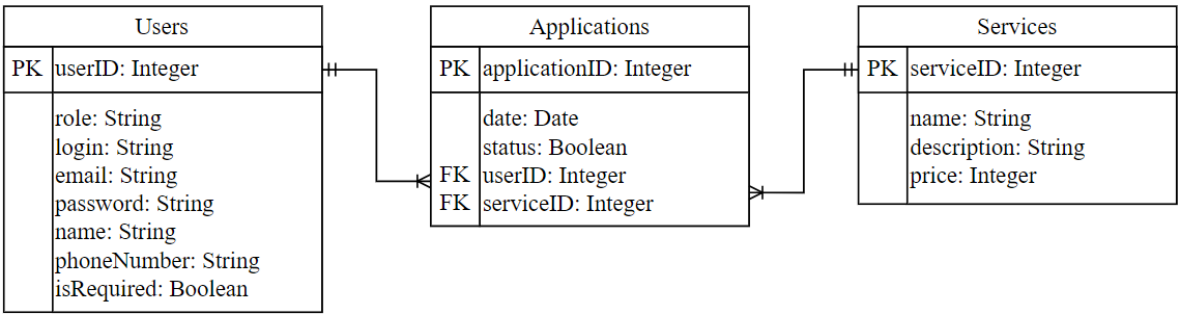

Рисунок 7 – Физическая модель данных

Связи, представленные на рисунке 7, описаны в таблице 3.

Таблица 3 – Связи данных

Ключ связи	Таблица	Подчинённая таблица	Связь
userID (уникальный идентификатор пользователя)	Users (Пользователи)	Applications (заявки)	1:M
serviceID (уникальный идентификатор услуги)	Services (Услуги)	Applications (заявки)	1:M

В таблице 4 представлена структура таблиц.

Таблица 4 – Структура таблиц

Название поля	Тип поля	Описание
Users (Пользователи)		
userID	Integer	ID пользователя (уникальный идентификатор пользователя)
Role	String	Роль (строка, указывающая на роль пользователя: Клиент/Администратор)
Login	String	Логин (строка, содержащая логин пользователя)
Email	String	Почта (строка, содержащая адрес электронной почты пользователя)
password	String	Пароль (строка, содержащая пароль пользователя)
Name	String	Имя (Строка, содержащая имя пользователя)
phoneNumber	String	Номер телефона (строка, содержащая номер телефона пользователя)
isRequired	String	Статус пользователя (поле, содержащую статус пользователя: есть активные заявки/активных заявок нет)
Applications (Заявки)		
applicationID	Integer	ID заявки (уникальный идентификатор заявки на обслуживание)
Date	Date	Дата (поле, содержащее время оформления заявки)
Status	Boolean	Статус заявки (поле, содержащую статус заявки пользователя: заявка активна/заявка неактивна)
userID	Integer	ID пользователя (идентификатор пользователя, отправившего заявку)
serviceID	Integer	ID услуги (идентификатор услуги, выбранной в заявке)
Services (Услуги)		
serviceID	Integer	ID услуги (уникальный идентификатор услуги)
Name	String	Наименование (строка, содержащая наименование услуги)
description	String	Описание (строка, содержащая описание услуги)
Price	Integer	Цена (поле, содержащее стоимость услуги)

Описанные спецификации таблиц будут использоваться при создании структура базы данных в рамках физическое модели данных. Физическая модель данных представляет собой ключевой этап в проектировании баз данных, который определяет конкретные способы реализации структуры данных в выбранной СУБД.

2.4 Проектирование пользовательских сценариев интерфейса

На основе проведённого анализа салона красоты, были сформулированы требования к разрабатываемому веб-приложению.

Functionality (Функциональность):

- пользовательский интерфейс должен обеспечивать навигацию по всем страницам сайта: главная, услуги, мастера, контакты, авторизация, запись, профиль, заявки;
- страницы должны содержать необходимую информацию о салоне красоты, его услугах, мастерах и контактной информации;
- должна быть реализована возможность авторизации для пользователей с последующим доступом к дополнительным функциям, таким как запись на услуги и просмотр профиля;
- для администратора должна быть предусмотрена страница просмотра заявок на обслуживание.

Usability (Удобство использования):

- интерфейс должен быть интуитивно понятным и легким в использовании для всех категорий пользователей;
- навигация по сайту должна быть простой и легко доступной из любой страницы;
- для удобства пользователей следует предусмотреть возможность быстрой связи с салоном красоты через страницу контактов [14].

Reliability (Надежность):

- все функциональные элементы интерфейса должны работать стабильно и без сбоев;
- для безопасности данных пользователей следует обеспечить защиту личной информации и паролей.

Performance (Производительность):

- сайт должен быть оптимизирован для быстрой загрузки на всех типах устройств и подключениях к интернету;

– взаимодействие с интерфейсом должно происходить плавно и без задержек.

Supportability (Поддержка):

– пользовательский интерфейс должен быть адаптирован под различные браузеры и устройства, включая мобильные телефоны, планшеты и настольные компьютеры;

– для удобства обслуживания и дальнейшей разработки следует предусмотреть модульную структуру интерфейса.

На рисунке 8 изображена диаграмма вариантов использования, основанная на вышеперечисленном функционале.

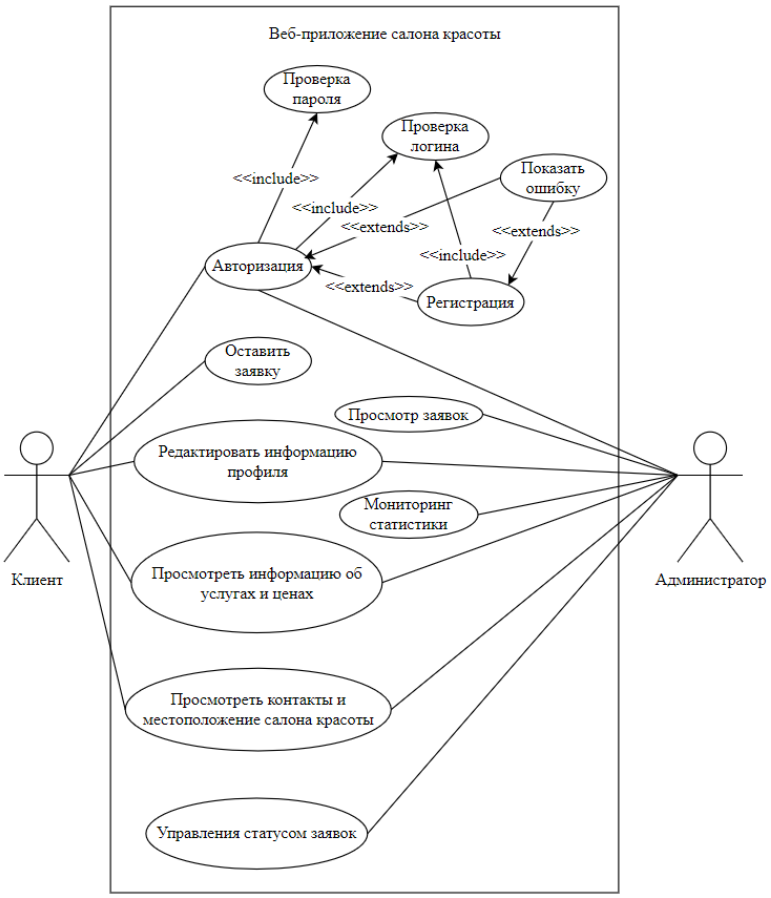

Рисунок 8 – Диаграмма вариантов использования

Диаграмма вариантов использования – это вид диаграммы UML, который описывает функциональность системы с точки зрения её

пользователей. Она позволяет идентифицировать основные действия и взаимодействия между пользователями и системой.

Роли:

- клиент: пользователь сайта салона красоты;
- администратор: пользователь с правами администрирования сайта.

Функция авторизации:

- клиент: вход на сайт с использованием учетной записи;
- администратор: вход на сайт с использованием административной учетной записи.

Функция оставления заявки:

- клиент: оставление заявки на услуги салона;
- администратор: просмотр заявок, управление их статусом.

Функция редактирования информации профиля:

- клиент: изменение личных данных и настроек профиля;
- администратор: изменение личных данных и настроек профиля.

Функция просмотра информации об услугах и ценах:

- клиент: просмотр перечня услуг, их описания и стоимости;
- администратор: просмотр перечня услуг, их описания и стоимости, возможно редактирование.

Функция просмотра информации о контактах и местоположении салона красоты:

- клиент: просмотр адреса салона, контактной информации и карты проезда;
- администратор: просмотр адреса салона, контактной информации и карты проезда, возможно редактирование.

Функция просмотра заявок и управление их статусом:

- клиент: недоступно;
- администратор: просмотр заявок на обслуживание, изменение их статуса.

Функция мониторинга статистики о выбираемых услугах:

- клиент: недоступно;
- администратор: ведение статистики выбираемых услуг и доступ к графическому представлению собираемых данных.

Таким образом, на основе сформулированных требований к разрабатываемому веб-приложению, была создана диаграмма вариантов использования, которая позволила спроектировать все пользовательские сценарии в зависимости от роли пользователя, необходимые для внедрения в разрабатываемое веб-приложение.

Выводы по главе 2

В рамках второй главы были выбраны стек технологий и инструменты для разработки веб-приложения. Также была разработана архитектура с учётом модульного подхода и спроектирована модель данных. По результатам спроектированной архитектуры и модели данных были разработаны пользовательские сценарии и требования к пользовательскому интерфейсу.

Глава 3 Реализация и тестирование веб-приложение

3.1 Краткое описание разработанного решения

Разработанное веб-приложение представляет собой полнофункциональное приложение для салона красоты, предоставляющее пользователям возможность ознакомиться с услугами и мастерами, оставить заявку на обслуживание, а также редактировать свой профиль. Приложение реализовано с использованием следующих технологий:

- фронтенд: HTML, CSS и JavaScript;
- бэкенд: Node.js с использованием фреймворка Express.js;
- база данных: PostgreSQL.

Основной функционал (Приложение А) веб-приложения включают в себя:

а) страницы приложения:
   1) главная страница: содержит основную информацию о салоне красоты и его услугах;
   2) страница просмотра услуг и цен: пользователи могут ознакомиться с перечнем услуг и их стоимостью;
   3) страница просмотра списка мастеров: предоставляет полную информацию о мастерах салона;
   4) страница контактов: содержит контактную информацию и карту проезда до салона.

б) авторизация и регистрация:
   1) авторизация пользователя: после входа пользователю доступна дополнительная функциональность;
   2) регистрация нового пользователя: новые пользователи могут создать аккаунт для получения доступа к дополнительным функциям.

в) запись на обслуживание:

1) отправка заявки: пользователи могут оставить заявку на обслуживание, указав свой номер телефона.

г) редактирование профиля:

1) изменение личных данных: пользователи имеют возможность изменять информацию о себе в профиле.

д) административная панель:

1) просмотр заявок на обслуживание: для администратора доступна страница просмотра заявок с возможностью изменения их статуса и выбором услуги, выбранной клиентом;
2) просмотр статистических данных: для администратора доступна страница просмотра статистических данных о выбираемых услугах в виде графика.

Разработанное решение предоставляет удобный и интуитивно понятный интерфейс как для клиентов салона красоты, так и для его администраторов, обеспечивая полноценное взаимодействие и комфортное пользование.

3.2 Интеграция базы данных и разработка механизмов хранения и обработки данных

Интеграция базы данных является ключевым аспектом разработки веб-приложений, поскольку обеспечивает хранение и доступ к данным, необходимым для функционирования приложения. В данном пункте представлены основные шаги по интеграции базы данных PostgreSQL и разработке механизмов хранения и обработки данных в приложении.

База данных PostgreSQL была выбрана в качестве системы управления базами данных для приложения. Для начала была разработана схема базы данных, которая была реализована с использованием Sequelize - ORM (Object-Relational Mapping) для Node.js. Sequelize обеспечивает удобную работу с базой данных через объекты JavaScript, что значительно упрощает разработку

и обслуживание приложения [22]. Реализация базы данных изображена на рисунке 9.

models > JS models.js > ...

```
const sequelize = require('../db');
const {DataTypes} = require('sequelize');
const User = sequelize.define(
    'user', {
        id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true},
        email: {type: DataTypes.STRING, unique: true},
        password: {type: DataTypes.STRING},
        role: {type: DataTypes.STRING, defaultValue: 'USER'},
        isRequested: {type: DataTypes.BOOLEAN, defaultValue: false},
        name: {type: DataTypes.STRING},
        login: {type: DataTypes.STRING, unique: true},
        phoneNumber: {type: DataTypes.STRING}
    }
);
const Applications = sequelize.define(
    'applications', {
        id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true},
        date: {type: DataTypes.DATE},
        status: {type: DataTypes.BOOLEAN},
        serviceId: {type: DataTypes.INTEGER},
        userId: {type: DataTypes.INTEGER}
    }
)
const Services = sequelize.define(
    'services', {
        id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true},
        name: {type: DataTypes.STRING},
        description: {type: DataTypes.STRING},
        price: {type: DataTypes.INTEGER}
    }
)
```

Рисунок 9 – Реализация базы данных

Интеграция с базой данных.

Для интеграции с базой данных был разработан модуль db.js, который устанавливает соединение с базой данных PostgreSQL. Параметры соединения (название БД, пользователя, пароль, хост и порт) хранятся в переменных

окружения (Приложение Б) для безопасности. Модуль db.js изображен на рисунке 10.

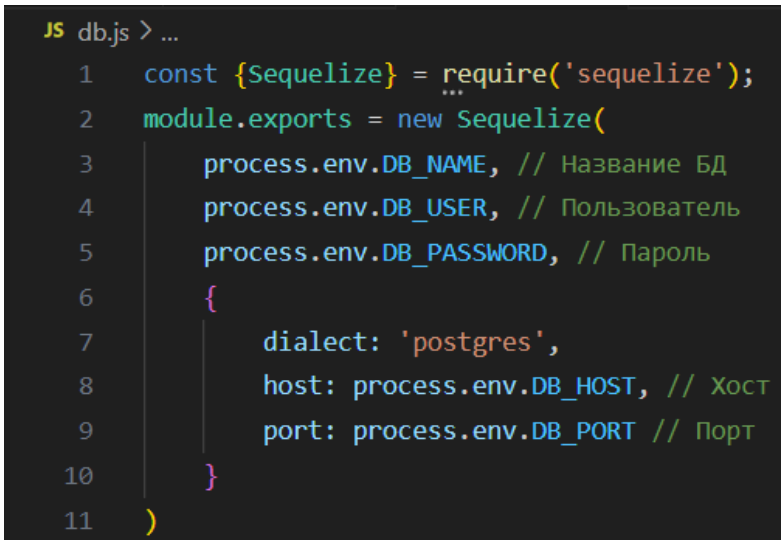

Рисунок 10 – Модуль db.js

Разработка механизмов хранения и обработки данных.

Для работы с данными был разработан модуль контроллеров userController.js (Приложение В), который содержит логику обработки запросов к базе данных. В данном модуле реализованы методы для регистрации и аутентификации пользователей, а также для выполнения других операций с данными пользователей.

Для управления зависимостями и запуска приложения был создан файл package.json (Приложение В), в котором указаны все зависимости приложения, а также скрипты для запуска в режиме разработки и другие сценарии.

Таким образом, интеграция базы данных PostgreSQL и разработка механизмов хранения и обработки данных являются ключевыми компонентами разработки веб-приложения. Благодаря использованию Sequelize ORM и правильной организации кода, была достигнута эффективная интеграция с базой данных и разработаны надежные механизмы обработки данных, обеспечивая стабильную работу и безопасность приложения.

3.3 Реализация модулей записи клиентов и администрирования

В данном пункте описывается реализация модулей, отвечающих за запись клиентов и администрирование заявок на обслуживание. Приведены детали бэкенд и фронтенд реализации, а также приведены примеры кода для наглядности.

Бэкенд: маршруты: маршруты определяют API для взаимодействия с модулями записи клиентов и администрирования [9]. Реализованы следующие эндпоинты:

– /api/user/request – отправка заявки на обслуживание;

– /api/user/requested – получение списка клиентов, оставивших заявку;

– /api/user/accept – изменение статуса заявки на принятую;

– /api/application/service – маршрут, get-запрос которого означает получение списка всех, когда-либо выбранных услуг, и post-запрос которого означает отправку выбранной клиентом услуги

Реализация эндпоинтов записи клиентов и администрирования изображена на рисунке 11.

```
router.patch('/request', UserController.request);
router.get('/requested', UserController.requested);
router.patch('/accept', UserController.accept);
```

Рисунок 11 – Реализация эндпоинтов записи клиентов и администрирования

Бэкенд: контроллеры содержат логику обработки запросов, связанных с модулями записи клиентов и администрирования. В данном модуле реализованы функции для отправки заявки, получения списка клиентов, оставивших заявку, и изменения статуса заявки.

Реализация эндпоинтов администрирования заявок на обслуживание изображена на рисунке 12.

```
router.get('/services', ApplicationController.getApplications);
router.post('/services', ApplicationController.setApplications);
```

Рисунок 12 – Реализация эндпоинтов администрирования заявок на обслуживание

Реализация контроллеров модуля записи клиентов и администрирования изображена на рисунке 13.

```
async request(req, res, next) {
    const {login, number} = req.body;
    const user = await User.findOne({where: {login}});
    if(!user) {
        return next(ApiError.internal('Пользователь с таким логином не найден'));
    }
    if(!number) {
        return next(ApiError.badRequest('Указан неверный номер'));
    }
    await user.update({phoneNumber: number});
    await user.update({isRequested: true});
    return res.json({user});
}

async requested(req, res) {
    const isRequested = true;
    const users = await User.findAll({where: {isRequested}});
    return res.json(users);
}

async accept(req, res) {
    const {email} = req.body;
    const user = await User.findOne({where: {email}});
    await user.update({phoneNumber: null});
    await user.update({isRequested: false});
    return res.json({user});
}
```

Рисунок 13 – Реализация контроллеров модуля записи клиентов и администрирования

Также был реализован модуль администрирования заявок на обслуживание (Приложение В).

Фронтенд:

– скрипт отправки заявки: был разработан скрипт (Приложение В), обеспечивающий отправку заявок на обслуживание на стороне клиента. Он взаимодействует с бэкендом через API для отправки данных о заявке и отображает результат на интерфейсе;
– скрипт отображения списка заявок в административной панели: был разработан скрипт (Приложение В), отображающий список заявок на обслуживание администратору. Данный скрипт получает данные о заявках с бэкенда через API и динамически отображает их на интерфейсе;
– скрипт отправки выбранной клиентом услуги: был разработан скрипт (Приложение В), обеспечивающий отправку выбранной клиентом услуги на сервер. Он взаимодействует с бэкендом через API для отправки данных о выбранной услуге;
– скрипт отображения статистики о выбранных услугах: был разработан скрипт (Приложение В), отображающий данные о выбранных услугах в виде графика. Данный скрипт получает данные о выбранных услугах с бэкенда через API и динамически отображает их на интерфейсе.

Интерфейсы отправки заявок на обслуживание и отображения списка заявок в административной панели изображены на рисунках 14 и 15 соответственно.

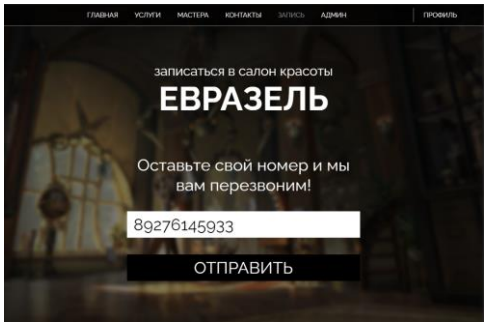

Рисунок 14 – Интерфейс отправки заявок на обслуживание

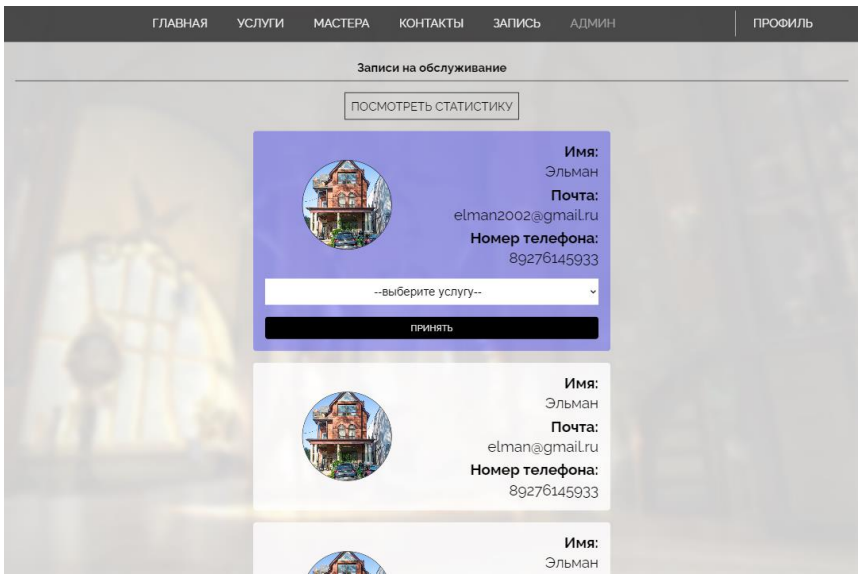

Рисунок 15 – Интерфейс отображения списка заявок

Интерфейс отображения статистических данных о выбранных услугах в виде графика в административной панели изображён на рисунке 16.

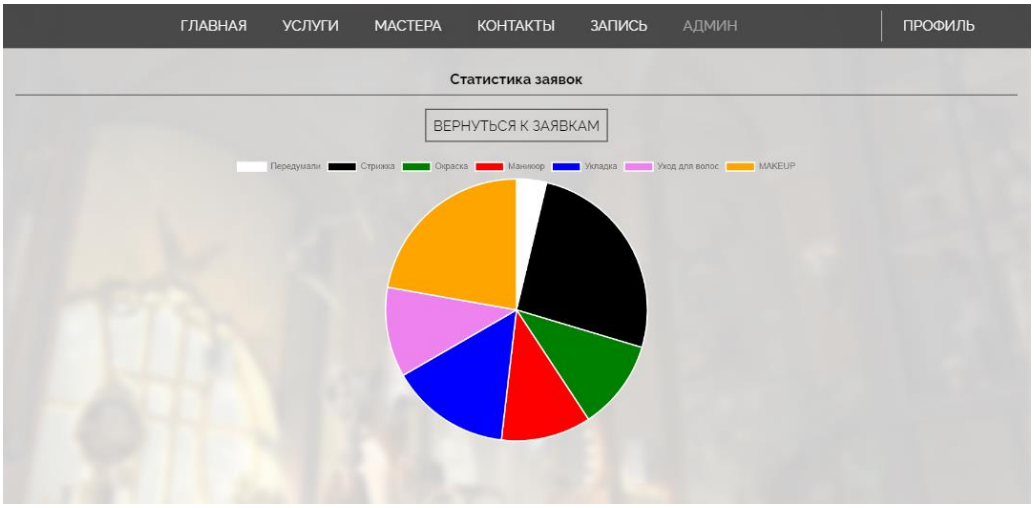

Рисунок 16 – Интерфейс отображения статистических данных о выбранных услугах в виде графика

Реализация модулей записи клиентов и администрирования представляет собой сложный процесс, который включает в себя взаимодействие бэкенд и фронтенд частей приложения. Благодаря правильной

интеграции и обработке данных, пользователи могут эффективно взаимодействовать с системой, оставлять заявки на обслуживание, а администраторы управлять ими, что обеспечивает удобство и эффективность использования приложения.

3.4 Реализация механизмов аутентификации и авторизации пользователей

Данный пункт описывает реализацию механизмов аутентификации и авторизации пользователей в приложении. Процесс включает в себя настройку бэкенд и фронтенд частей приложения, включая создание API эндпоинтов для регистрации, входа в систему, а также соответствующие скрипты на фронтенде.

Бэкенд: маршруты определяют доступные API для регистрации, входа в систему и проверки авторизации пользователей. Реализованы следующие эндпоинты:

– /api/user/registration – отправка заявки на обслуживание;
– /api/user/login – получение списка клиентов, оставивших заявку;
– /api/user/auth – изменение статуса заявки на принятую.

Реализация эндпоинтов аутентификации изображена на рисунке 17.

```
router.post('/registration', UserController.registration);
router.post('/login', UserController.login);
router.get('/auth', AuthMiddleware, UserController.check);
```

Рисунок 17 – Реализация эндпоинтов аутентификации

Бэкенд: контроллеры содержат логику обработки запросов, связанных с аутентификацией пользователей. Здесь реализованы функции регистрации и входа в систему (Приложение В), а также проверки авторизации.

Фронтенд:

– скрипт входа в систему: скрипт (Приложение В), обрабатывающий данные пользователя при входе в систему. Данный скрипт отправляет запрос на сервер для аутентификации и сохраняет полученный токен в локальном хранилище;
– скрипт регистрации: скрипт (Приложение В), обрабатывающий данные пользователя при регистрации. Данный скрипт отправляет запрос на сервер для создания нового аккаунта и сохраняет полученный токен в локальном хранилище;
– скрипт выхода из системы: скрипт, обрабатывающий выход пользователя из системы. Данный скрипт удаляет токен из локального хранилища и перенаправляет пользователя на главную страницу. Скрипт выхода из системы изображён на рисунке 18;
– скрипт проверки авторизации: скрипт, проверяющий авторизацию и роль пользователя при загрузке страницы. Данный скрипт отправляет запрос на сервер для проверки токена и отображает соответствующие элементы интерфейса. Скрипт проверки авторизации изображён на рисунке 19.
– Данные механизмы обеспечивают безопасность и удобство входа и выхода пользователей.

Рисунок 18 – Скрипт выхода из системы

```
scripts > JS check.js > ...
1   const token = localStorage.getItem('token');
2   const headerBarList = document.querySelector('.header__navigation');
3   const loginContainer = document.querySelector('.header__login-container');
4   const adminLinkTemplate = document.querySelector('#admin').content.cloneNode(true);
5   const accountLinkTemplate = document.querySelector('#account').content.cloneNode(true);
6   const signUpLinkTemplate = document.querySelector('#sign-up-link').content.cloneNode(true);
7   if(token) {
8       loginContainer.innerHTML = '';
9       loginContainer.append(accountLinkTemplate);
10      headerBarList.append(signUpLinkTemplate);
11      fetch('http://localhost:5000/api/user/auth', {
12          headers: {
13              authorization: token
14          }
15      }).then(res => {
16          return res.json();
17      }).then(user => {
18          console.log(user.role);
19          if(user.role === 'ADMIN') {
20              headerBarList.append(adminLinkTemplate);
21          }
22      })
23  }
```

Рисунок 19 – Скрипт проверки авторизации

Интерфейсы авторизации и регистрации изображены на рисунках 20 и 21 соответственно.

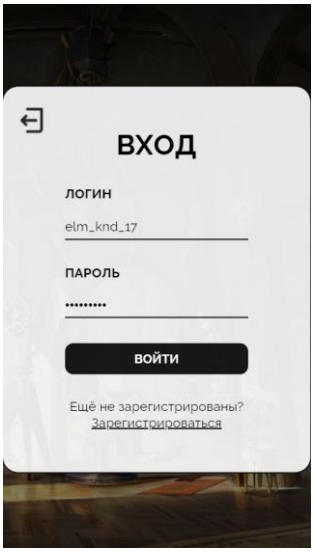

Рисунок 20 – Интерфейс авторизации

Рисунок 21 – Интерфейс регистрации

Интерфейс страницы профиля, имеющей кнопку выхода из аккаунта изображён на рисунке 22.

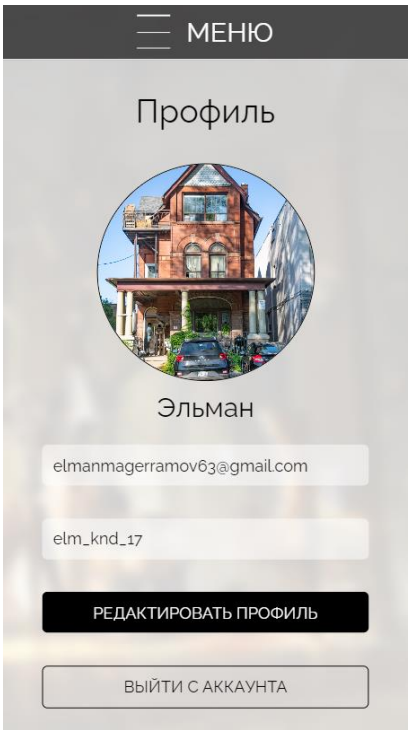

Рисунок 22 – Интерфейс страницы профиля

На серверной стороне были настроены маршруты API для обработки запросов на регистрацию и вход в систему, а также для проверки авторизации. Контроллеры обеспечивают логику обработки этих запросов, включая шифрование паролей, создание и проверку токенов доступа. Middleware используется для проверки токена авторизации при доступе к защищенным ресурсам.

На клиентской стороне были реализованы скрипты для входа и регистрации пользователей, а также для проверки авторизации при загрузке страницы. С помощью локального хранилища браузера сохраняются токены доступа, обеспечивая удобную аутентификацию и авторизацию пользователей.

Все эти механизмы в совокупности обеспечивают безопасность и удобство использования приложения, позволяя пользователям безопасно входить в систему, регистрироваться и получать доступ к защищенным ресурсам.

3.5 Обеспечение безопасности и защиты данных

В разработке веб-приложений безопасность играет критически важную роль. Для обеспечения безопасности и защиты данных в приложении были использованы различные подходы и механизмы [7]. Давайте рассмотрим каждый из них подробно.

Middleware для аутентификации:

– модуль AuthMiddleware.js: данный middleware отвечает за аутентификацию пользователей. Он проверяет наличие токена доступа в заголовке запроса и, если токен присутствует и верен, сохраняет информацию о пользователе в объекте запроса (req.user). В случае отсутствия или неверности токена, middleware возвращает статус ошибки 401. Модуль AuthMiddleware.js изображён на рисунке 23;

– модуль CheckRoleMiddleware.js: данный middleware предназначен для проверки роли пользователя. Он принимает в качестве параметра необходимую роль и проверяет, соответствует ли роль пользователя роли, указанной в параметре. Если роль не совпадает, middleware возвращает статус ошибки 403. Модуль CheckRoleMiddleware.js изображён на рисунке 24.

middleware > JS AuthMiddleware.js > ...

```
1  const jwt = require('jsonwebtoken');
2  module.exports = function (req, res, next) {
3      if (req.method === 'OPTIONS') {
4          next();
5      }
6      try {
7          const token = req.headers.authorization;
8          if(!token) {
9              return res.status(401).json({message: 'Не авторизован'});
10         }
11         const decoded = jwt.verify(token, process.env.SECRET_KEY);
12         req.user = decoded;
13         next();
14     } catch (e) {
15         res.status(401).json({message: 'Не авторизован'});
16     }
17 };
```

Рисунок 23 – Модуль AuthMiddleware.js

middleware > JS CheckRoleMiddleware.js > ...

```
1  const jwt = require('jsonwebtoken');
2  module.exports = function (role) {
3      return function (req, res, next) {
4          if (req.method === 'OPTIONS') {
5              next();
6          }
7          try {
8              const token = req.headers.authorization;
9              if(!token) {
10                 return res.status(401).json({message: 'Не авторизован'});
11             }
12             const decoded = jwt.verify(token, process.env.SECRET_KEY);
13             if(decoded.role !== role) {
14                 return res.status(403).json({message: 'Нет доступа'});
15             }
16             req.user = decoded;
17             next();
18         } catch (e) {
19             res.status(401).json({message: 'Не авторизован'});
20         }
21     };
22 }
```

Рисунок 24 – Модуль CheckRoleMiddleware.js

Middleware для обработки ошибок: модуль ErrorHandlingMiddleware.js: данный middleware отвечает за обработку ошибок в приложении. В случае возникновения ошибки он перехватывает её и возвращает соответствующий HTTP-статус и сообщение об ошибке. Если ошибка является экземпляром класса ApiError, то возвращается пользовательское сообщение об ошибке и соответствующий статус. В противном случае возвращается статус ошибки 500 с сообщением о непредвиденной ошибке.

Модуль ErrorHandlingMiddleware.js изображён на рисунке 25.

Рисунок 25 – Модуль ErrorHandlingMiddleware.js

Обработка пользовательских данных:

– модуль ApiError.js: данный класс представляет собой кастомный класс ошибок. Он используется для создания объектов ошибок с определенным HTTP-статусом и сообщением. Класс содержит статические методы для создания объектов ошибок различных типов, таких как badRequest, internal, forbidden. Модуль ApiError.js изображён на рисунке 26;
– модуль userController.js: в контроллере пользователей (Приложение В) реализованы методы для обработки запросов, связанных с аутентификацией, регистрацией, а также администрированием пользователей. Перед обработкой запросов происходит проверка на наличие и корректность токена аутентификации.

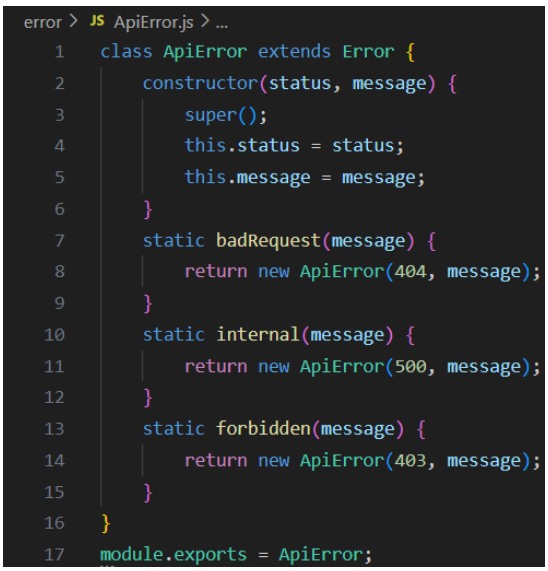

Рисунок 26 – Модуль ApiError.js

Механизм хранения паролей пользователей осуществляется путём его хеширования перед непосредственной записью в базы данных. Тем самым, пароль пользователя не хранится в прямом виде в базе, и при авторизации и вводе пользователем пароля, запросом к базе данных отправляется хешированный пароль, который сравнивается с хешированным паролем, соответствующему указанному пользователем логину. Реализации хеширования пароля при регистрации и авторизации представлены на рисунках 27 и 28 соответственно.

```
async registration(req, res, next) {
    const {email, password, role, name, login} = req.body;
    if(!email || !password) {
        return next(ApiError.badRequest('Некорректный email или пароль'))
    }
    const candidate = await User.findOne({where: {email}});
    if(candidate) {
        return next(ApiError.badRequest('Пользователь с таким email уже существует'));
    }
    const loginCandidate = await User.findOne({where: {login}});
    if(loginCandidate) {
        return next(ApiError.badRequest('Пользователь с таким логином уже существует'));
    }
    const hashPassword = await bcrypt.hash(password, 5);
    const user = await User.create({email, role, password: hashPassword, name, login});
    const token = generateJwt(user.id, user.login, user.role);
    return res.json({token});
}
```

Рисунок 27 – Реализация хеширования пароля при регистрации

```
async login(req, res, next) {
    const {password, login} = req.body;
    const user = await User.findOne({where: {login}});
    if(!user) {
        return next(ApiError.internal('Пользователь с таким логином не найден'));
    }
    let comparePassword = bcrypt.compareSync(password, user.password);
    if(!comparePassword) {
        return next(ApiError.internal('Указан неверный пароль'));
    }
    const token = generateJwt(user.id, user.login, user.role);
    return res.json({token});
}
```

Рисунок 28 – Реализация хеширования пароля при авторизации

Пример хранения хеширования пароля в базе данных изображён на рисунке 29.

id [PK] integer	email character varying (255)	password character varying (255)	role chara
12	elman@gmail.ru	$2b$05$GFFE2IuTeh/IA6Lc4PZgtO5nhrANyJQ87Sc.pggjQqfBl7/YUItce	USEI
13	elmanmagerramov63@gmail.com	$2b$05$WkEE4N1e8.dVDNYeTgfGp.40S0kn5TdMsJctHXS11Jr9Rk9B7.WeC	ADM

Рисунок 29 – Пример хранения хешированного пароля в базе данных

3.6 Тестирование функциональности и производительности приложения

Тестирование функциональности пользовательского интерфейса является важной частью разработки приложения, поскольку обеспечивает проверку соответствия его функций заявленным требованиям и ожиданиям пользователей.

Было проведено тестирование функциональности и производительности клиентской части веб-приложения. Тестирование проводилось с целью проверки основных функций приложения, а также оценки его адаптивности и работоспособности в различных веб-браузерах [11].

Виды тестирования:

– функциональное тестирование: проверка основных функций приложения, таких как регистрация, авторизация, отправка заявок на обслуживание, просмотр заявок на обслуживание администратором, редактирование статуса заявок;
– тестирование адаптивности: оценка работы приложения на различных устройствах (настольные компьютеры, планшеты, мобильные телефоны) с разными разрешениями экранов;
– кросс-браузерное тестирование: проверка работоспособности приложения в различных веб-браузерах (Chrome, Firefox, Safari, Edge и другие) [13].

Функциональное тестирование:

а) регистрация пользователя:
    1) проверка успешного завершения регистрации при корректном заполнении всех обязательных полей формы;
    2) проверка отображения сообщения об ошибке при попытке регистрации с неполными данными.
б) авторизация пользователя:
    1) проверка успешного входа в систему при корректном вводе логина и пароля;
    2) проверка отображения сообщения об ошибке при неверном вводе логина или пароля.
в) отправка заявки на обслуживание:
    1) проверка успешной отправки заявки при корректном заполнении всех необходимых данных в форме заявки;
    2) проверка отображения сообщения об ошибке при попытке отправки заявки с недостаточными данными.
г) просмотр заявок на обслуживание администратором:
    1) проверка корректного отображения списка заявок администратором;

2) проверка возможности изменения статуса заявки администратором.

д) просмотр статистических данных о выбираемых услугах:

1) проверка возможности выбора услуги во время изменения статуса заявки администратором;
2) проверка корректного отображения статистических данных о выбираемых услугах в виде графика администратором.

Тестирование адаптивности:

– проверка корректного отображения интерфейса приложения на различных устройствах с разными разрешениями экранов;
– оценка удобства использования интерфейса на мобильных устройствах.

Кросс-браузерное тестирование: проверка работы приложения в различных веб-браузерах на настольных и мобильных платформах.

В результате тестирования было установлено, что клиентская часть веб-приложения успешно прошла все этапы проверки. Основные функции приложения, такие как регистрация, авторизация, отправка заявок на обслуживание, просмотр заявок на обслуживание администратором, а также редактирование статуса заявок, работают стабильно и без каких-либо замечаний.

Тестирование также показало высокий уровень адаптивности приложения к различным устройствам и разрешениям экранов. Интерфейс приложения отображается корректно и удобно как на настольных компьютерах, так и на мобильных устройствах.

Кроме того, приложение успешно прошло кросс-браузерное тестирование и демонстрирует стабильную работу в различных веб-браузерах.

Таким образом, можно с уверенностью сказать, что клиентская часть веб-приложения полностью соответствует заявленным требованиям и готова к дальнейшему внедрению и использованию пользователем.

Для проверки производительности и надежности серверной части приложения было проведено тестирование с использованием инструмента autocannon. Этот инструмент позволяет имитировать нагрузку на сервер, а также оценивать его производительность и стабильность при различных условиях нагрузки.

Виды тестирования:

– нагрузочное тестирование: Оценка времени обработки запросов и пропускной способности сервера при различных уровнях нагрузки;
– тестирование надежности: Проверка стабильности работы сервера и его отказоустойчивости при длительной нагрузке.

Результаты тестирования:

– нагрузочное тестирование: в результате тестирования было установлено, что сервер успешно справляется с обработкой запросов при различных уровнях нагрузки. Время ответа на запросы остается стабильным, а пропускная способность сервера соответствует заявленным требованиям;
– надежность: в течение тестирования сервер демонстрировал стабильную работу и отсутствие сбоев даже при длительной нагрузке. Не было зафиксировано ни одного случая отказа сервера или недоступности его функционала.

Результаты нагрузочного тестирования серверной части приложения изображены на рисунке 30.

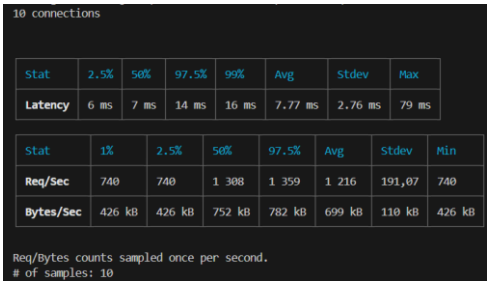

Рисунок 30 – Результаты нагрузочного тестирования серверной части приложения

Результаты тестирования показывают, что серверная часть приложения полностью соответствует требованиям производительности и надежности. Он способен обрабатывать запросы пользователей эффективно и стабильно даже при высоких нагрузках, что делает его готовым к использованию в реальных условиях эксплуатации.

Выводы по главе 3

В данной главе была выполнена разработка и реализация полноценного веб-приложения для управления клиентами и администрирования. От интеграции базы данных и разработки механизмов обработки данных до реализации модулей аутентификации, администрирования и безопасности, каждый этап был тщательно проработан и реализован. Тестирование функциональности и производительности приложения показало его стабильную работу и высокую производительность, что свидетельствует о качественной разработке и эффективной реализации задачи.

В результате работы было создано мощное веб-приложение, способное эффективно управлять клиентами и обеспечивать безопасность и защиту их данных. Разработанные модули и механизмы обеспечивают удобство использования приложения как для обычных пользователей, так и для администраторов. Общий вывод подтверждает успешную реализацию поставленных задач и готовность приложения к реальной эксплуатации.

Заключение

В рамках данной выпускной работы было проведено обширное исследование, затрагивающее различные аспекты разработки и реализации веб-приложения, направленного на упрощение и оптимизацию процессов управления клиентами, а также обеспечение высокого уровня безопасности и защиты данных. В своей работе мы предприняли шаги по анализу организации, определив ключевые потребности и требования к разрабатываемому приложению.

Процесс проектирования приложения был тщательно продуман с учетом модульного подхода и масштабируемости. Мы сфокусировались на выборе подходящих технологий и инструментов, что обеспечило эффективную реализацию архитектуры приложения и создание интуитивно понятного пользовательского интерфейса. Разработанная модель данных была гибкой и адаптивной, что позволило эффективно управлять информацией о клиентах и обеспечивать их конфиденциальность.

Одним из ключевых аспектов работы является реализация механизмов аутентификации и авторизации пользователей, а также обеспечение безопасности и защиты данных. Были разработаны соответствующие middleware для обработки запросов, внедрены механизмы проверки ролей пользователей и обработки ошибок. Тестирование приложения показало высокую функциональность, стабильность и соответствие стандартам безопасности.

В результате работы было успешно разработано веб-приложение, способное эффективно решать задачи управления клиентами и обеспечивать безопасность и защиту данных. Разработанное веб-приложение позволяет упростить и улучшить процесс предварительной записи клиентов на обслуживание в салон красоты, повышая эффективность работы и уровень удовлетворённости сотрудников компании.

Список используемой литературы

1. Астахов, В. П. Введение в веб-программирование. М.: Мир, 2015. 348 с.
2. Баев, Е. И. Системное проектирование информационных систем. СПб.: Питер, 2016. 256 с.
3. Беляев, И. В., & Казанцев, А. А. Разработка интерфейсов для веб-приложений. М.: ГИЦЛ, 2018. 292 с.
4. Бобров, А. Г. Основы модульного проектирования. М.: Эксмо, 2017. 208 с.
5. Волков, С. Н., & Матвеев, Д. С. Технологии и инструменты веб-разработки. М.: Академия, 2019. 384 с.
6. Грачев, П. А. Проектирование баз данных. СПб.: Лань, 2020. 416 с.
7. Иванов, К. Ю. Безопасность веб-приложений. М.: Диалектика, 2021. 312 с.
8. Казаков, В. Л. Модульное проектирование в IT. М.: ИНФРА-М, 2018. 240 с.
9. Лаптев, А. В. Администрирование информационных систем. СПб.: Питер, 2020. 304 с.
10. Новиков, А. П. Современные технологии веб-разработки. М.: Высшая школа, 2019. 352 с.
11. Петров, Д. Е. Инструменты и методики тестирования веб-приложений. М.: БХВ-Петербург, 2017. 270 с.
12. Савельев, М. Н. Разработка и администрирование веб-приложений. СПб.: Лань, 2020. 358 с.
13. Смирнов, В. А. Тестирование и отладка веб-приложений. М.: ГИЦЛ, 2019. 288 с.
14. Тихонов, Ю. А. Основы веб-дизайна. М.: Диалектика, 2018. 324 с.
15. Федоров, Е. И. Введение в базу данных. СПб.: Питер, 2017. 360 с.

16. Хомяков, В. П. Модульное проектирование: теоретические и практические аспекты. М.: Эксмо, 2021. 294 с.

17. Atkinson, R. J. (2016). Web Development with Node.js and Express: Leveraging the JavaScript Stack. O'Reilly Media. 420 p.

18. Coleman, A., & Mednieks, Z. (2017). Programming Android: Java Programming for the New Generation of Mobile Devices. O'Reilly Media. 608 p.

19. Duckett, J. (2014). HTML and CSS: Design and Build Websites. Wiley. 512 p.

20. Freeman, E., & Robson, E. (2020). Head First Design Patterns: Building Extensible and Maintainable Object-Oriented Software. O'Reilly Media. 694 p.

21. Monson-Haefel, R. (2017). Enterprise JavaBeans 3.1: Developing Enterprise Java Components. O'Reilly Media. 846 p.

22. Resig, J., & Bibeault, B. (2018). Secrets of the JavaScript Ninja. Manning Publications. 352 p.

Приложение А

Экранные формы разработанного приложения

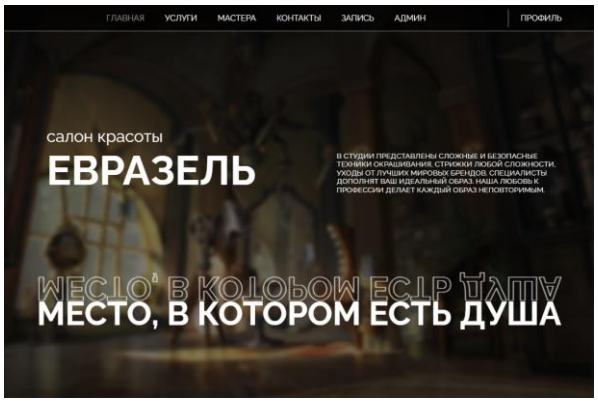

Рисунок А.1 – Хедер главной страницы (ПК версия)

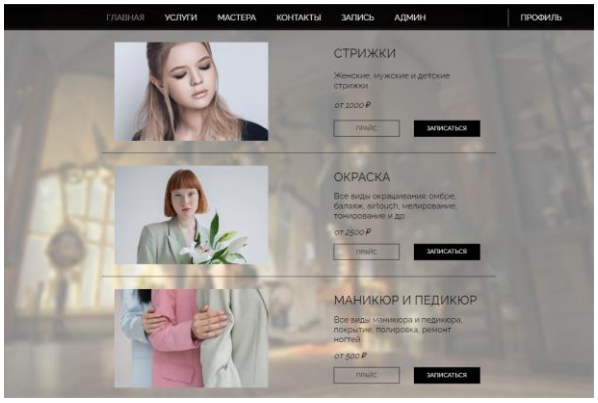

Рисунок А.2 – Тело главной страницы (ПК версия)

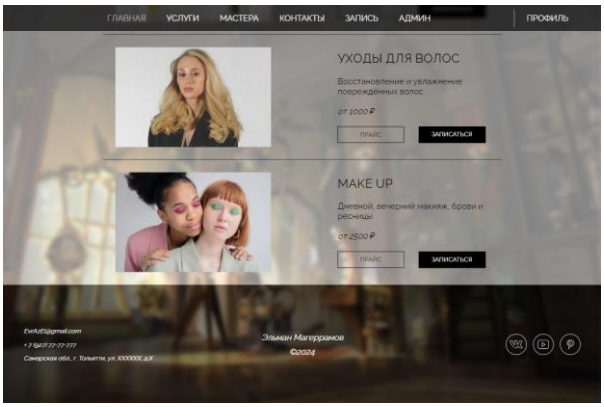

Рисунок А.3 – Футер главной страницы (ПК версия)

Рисунок А.4 – Хедер главной страницы (Мобильная версия)

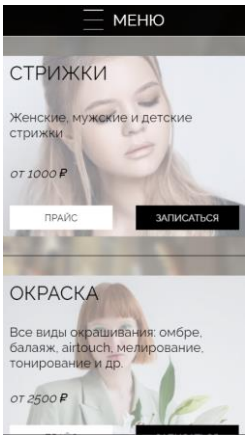

Рисунок А.5 – Тело главной страницы (Мобильная версия)

Рисунок А.6 – Футер главной страницы (Мобильная версия)

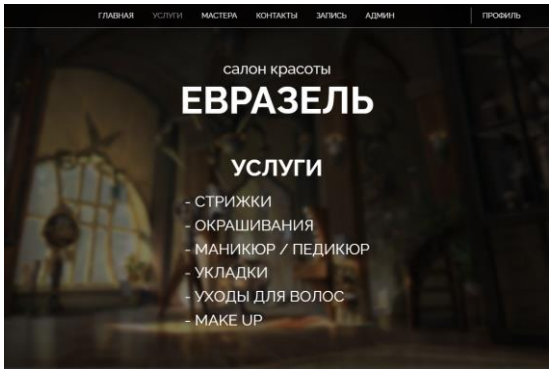

Рисунок А.7 – Хедер страницы услуг (ПК версия)

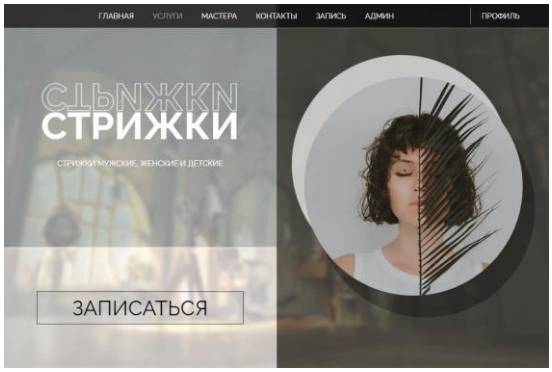

Рисунок А.8 – Секция с наименованием услуги (ПК версия)

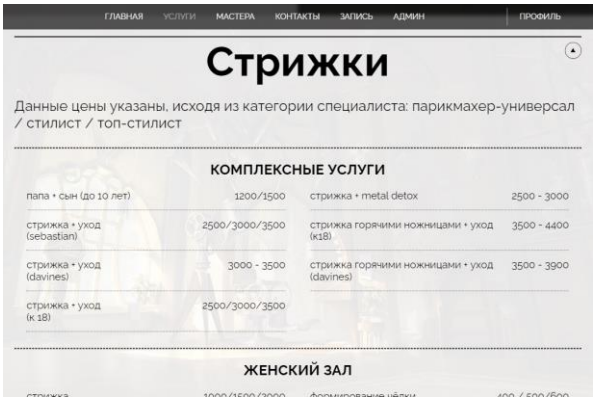

Рисунок А.9 – Секция с прайсом (ПК версия)

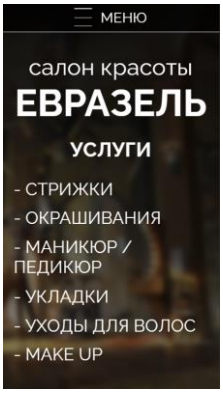

Рисунок А.10– Хедер страницы услуг (Мобильная версия)

Рисунок А.11 – Секция с наименованием услуги (Мобильная версия)

Рисунок А.12 – Секция с прайсом (Мобильная версия)

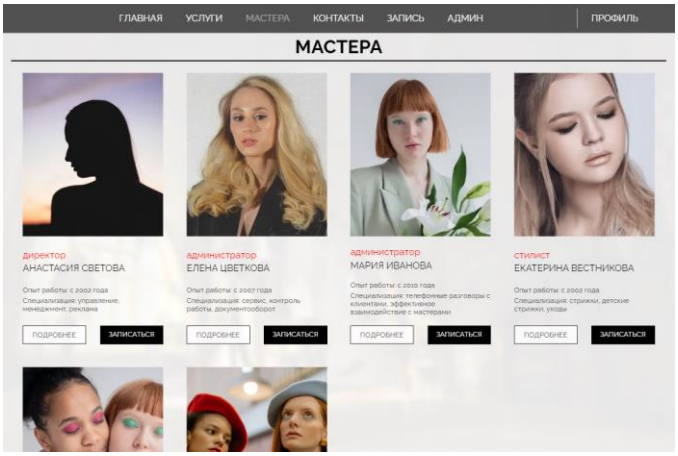

Рисунок А.13 – Список мастеров (ПК версия)

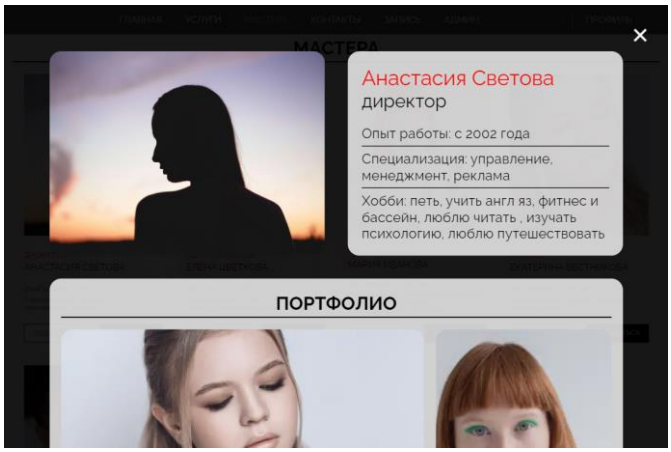

Рисунок А.14 – Подробная информация о мастере (ПК версия)

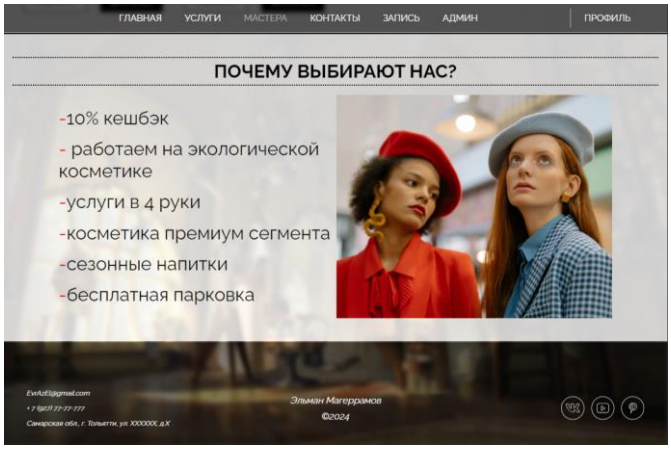

Рисунок А.15 – Футер страница просмотра мастеров (ПК версия)

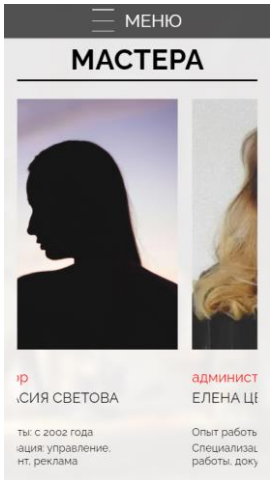

Рисунок А.16 – Список мастеров (Мобильная версия)

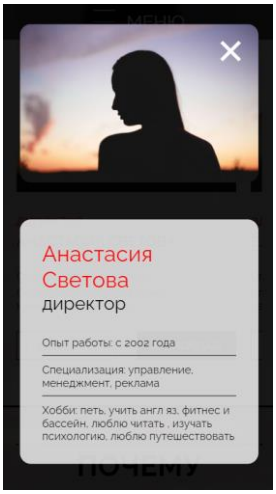

Рисунок А.17 – Подробная информация о мастере (Мобильная версия)

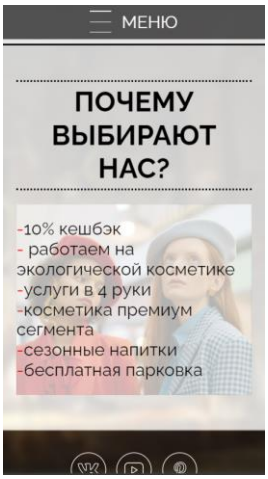

Рисунок А.18 – Футер страница просмотра мастеров (Мобильная версия)

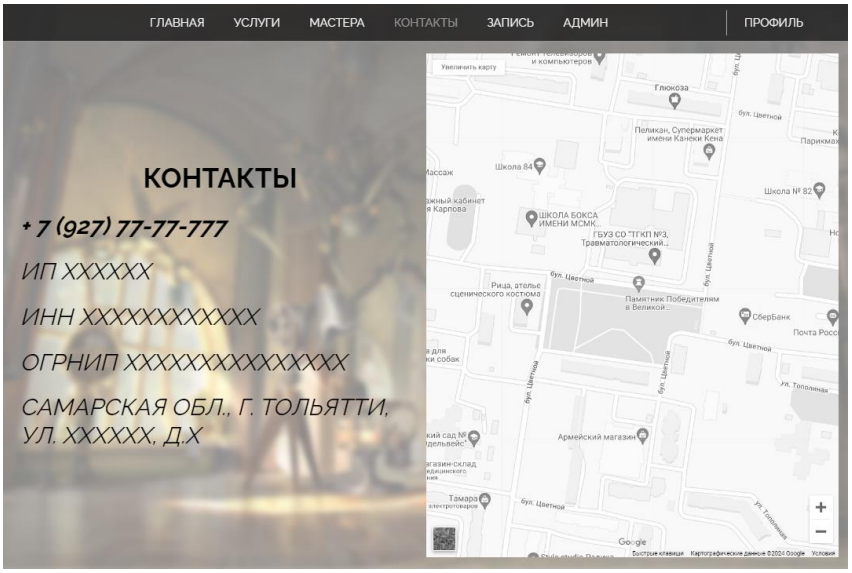

Рисунок А.19 – Тело страницы контактов (ПК версия)

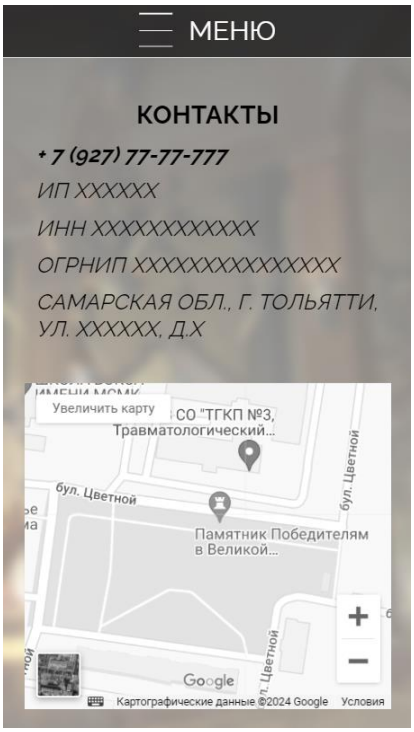

Рисунок А.20 – Тело страницы контактов (Мобильная версия)

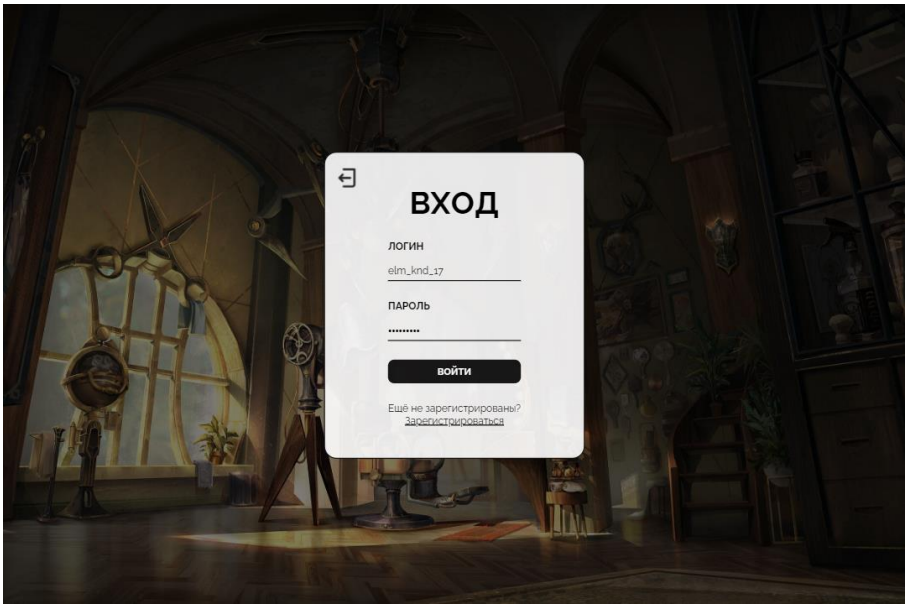

Рисунок А.21 – Страница авторизации (ПК версия)

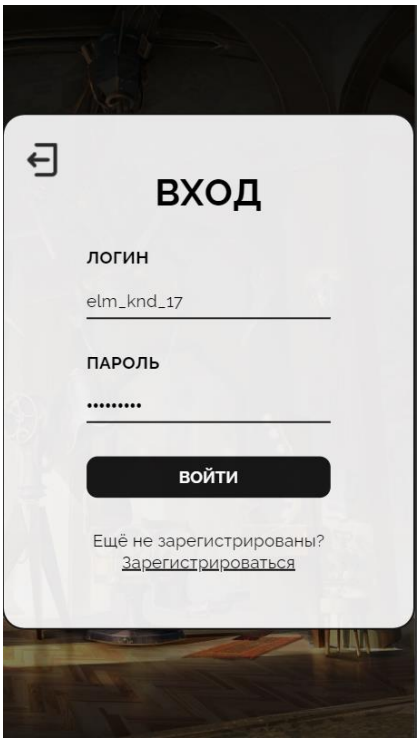

Рисунок А.22 – Страница авторизации (Мобильная версия)

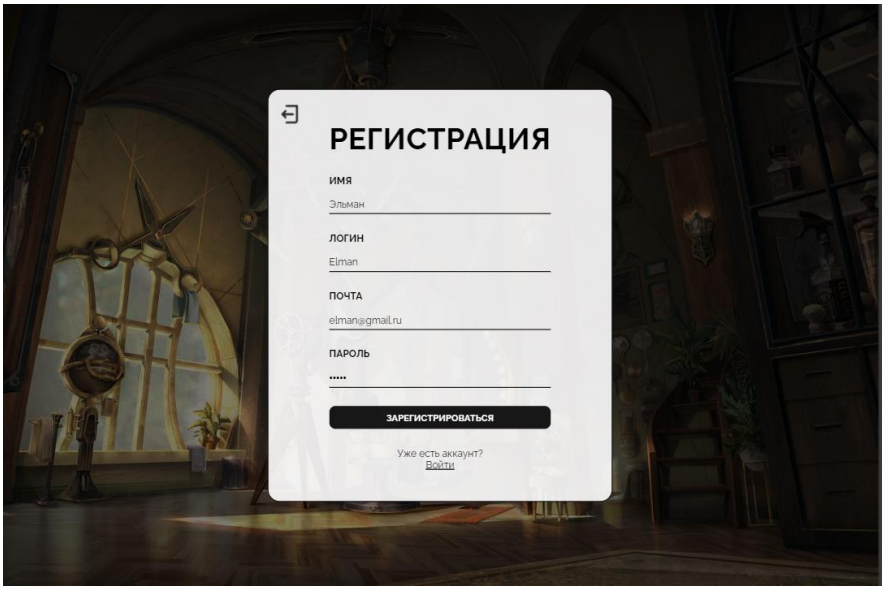

Рисунок А.23 – Страница регистрации (ПК версия)

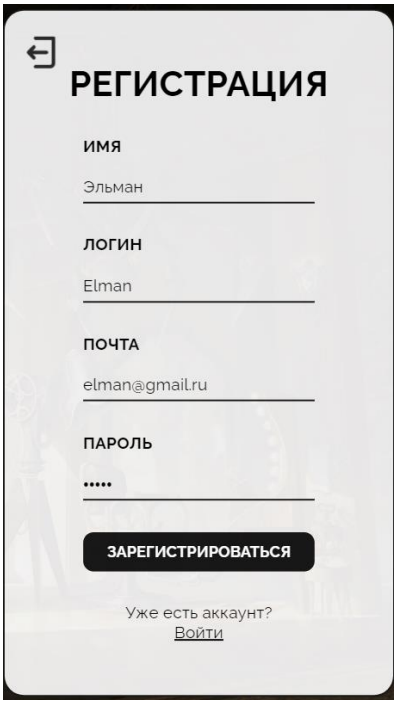

Рисунок А.24 – Страница регистрации (Мобильная версия)

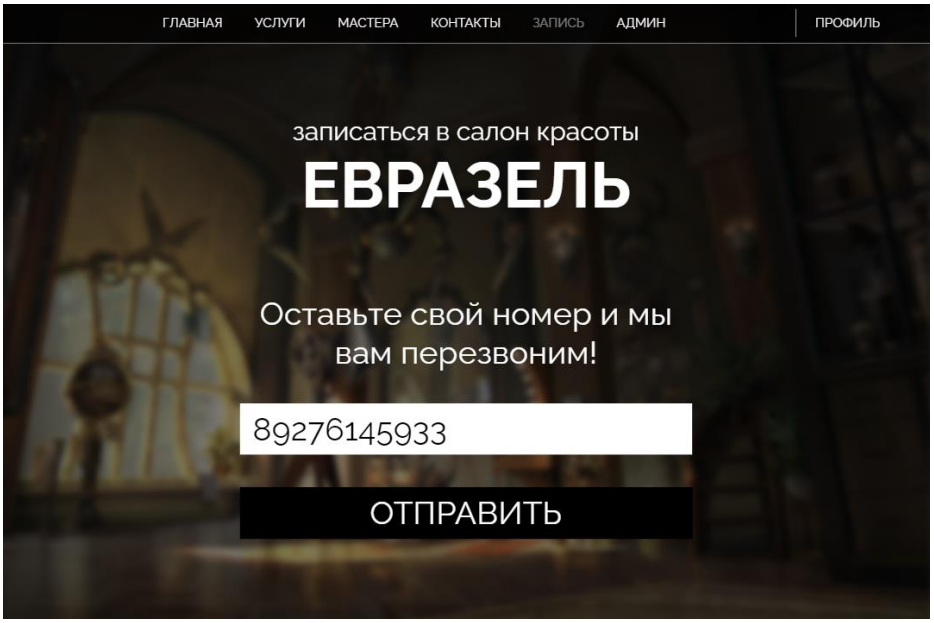

Рисунок А.25 – Страница записи на обслуживание (ПК версия)

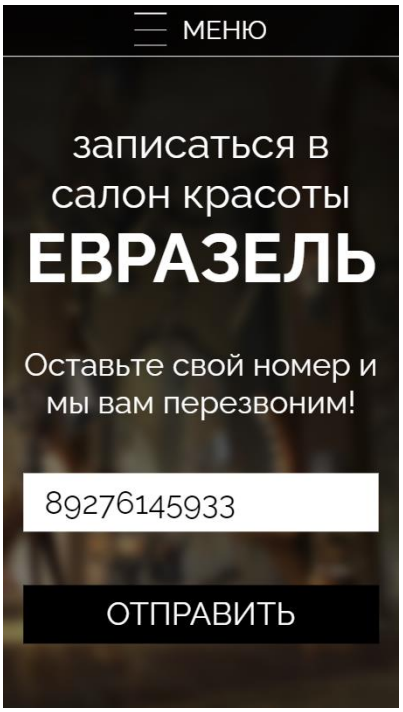

Рисунок А.26 – Страница записи на обслуживание (Мобильная версия)

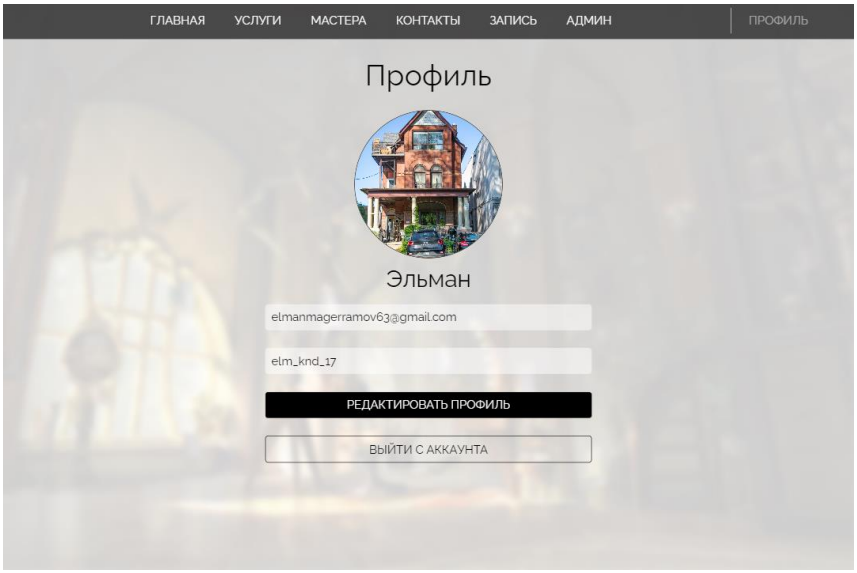

Рисунок А.27 – Страница профиля (ПК версия)

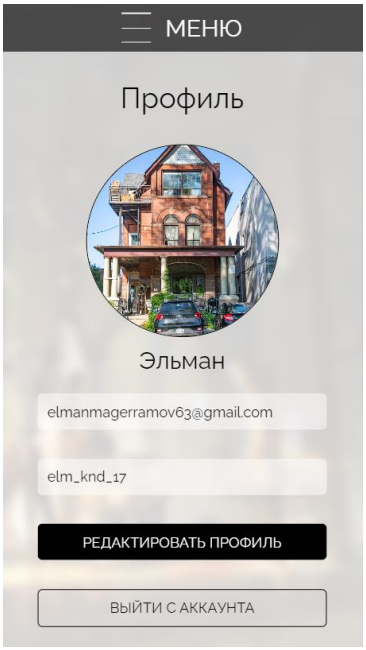

Рисунок А.28 – Страница профиля (Мобильная версия)

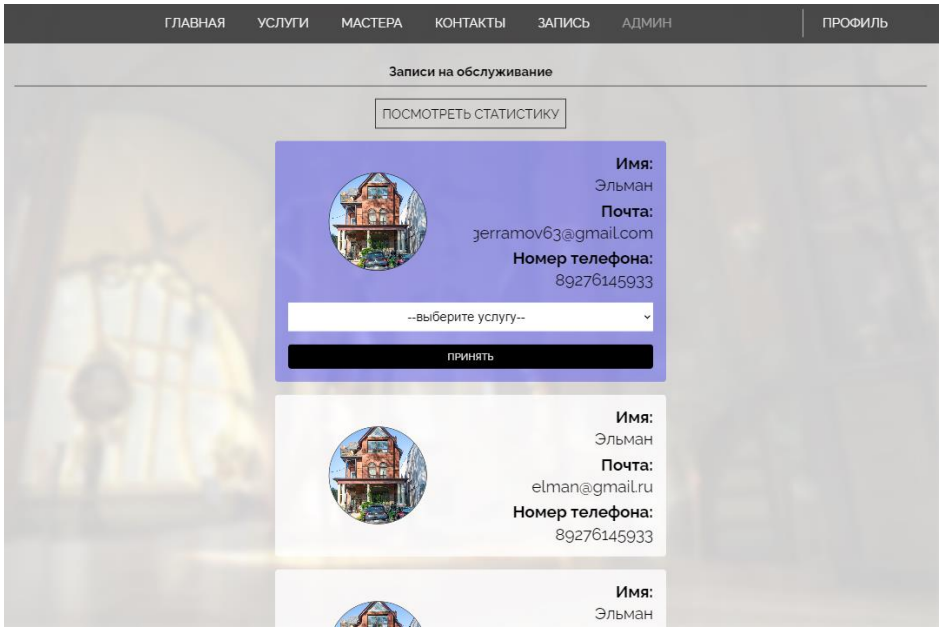

Рисунок А.29 – Страница просмотра заявок на обслуживание (ПК версия)

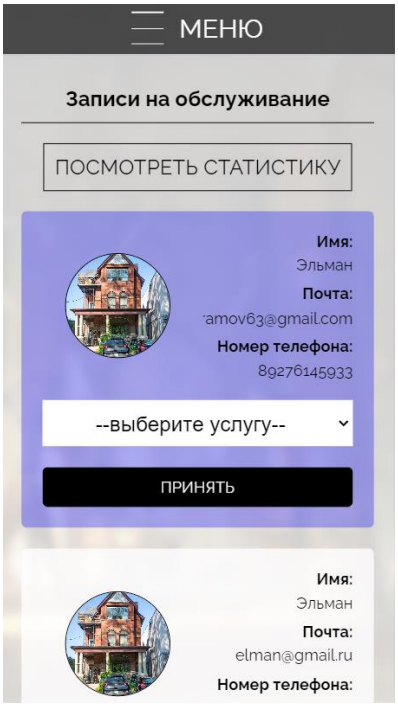

Рисунок А.30 – Страница просмотра заявок на обслуживание (Мобильная версия)

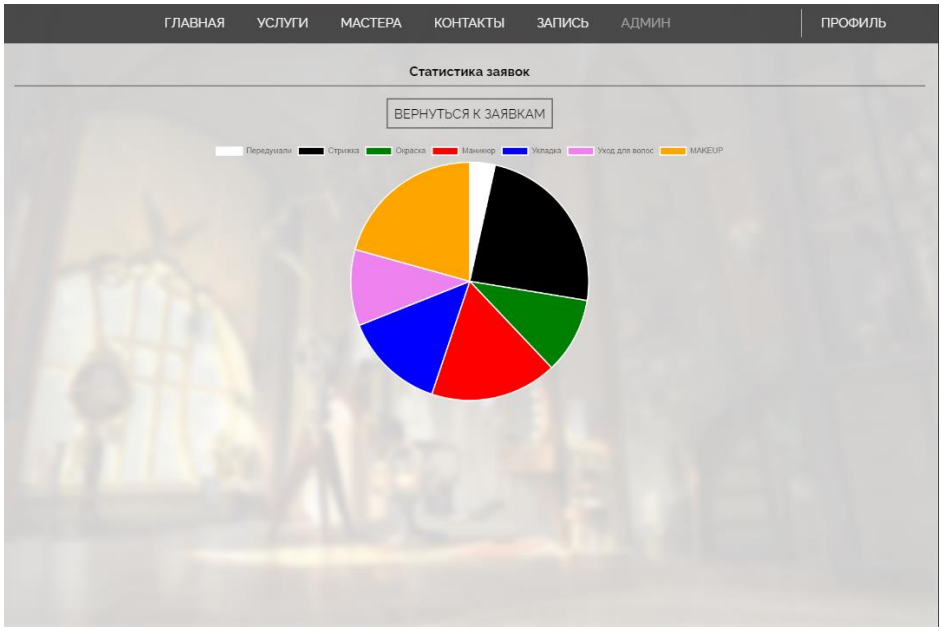

Рисунок А.31 – Страница графика выбираемых услуг (ПК версия)

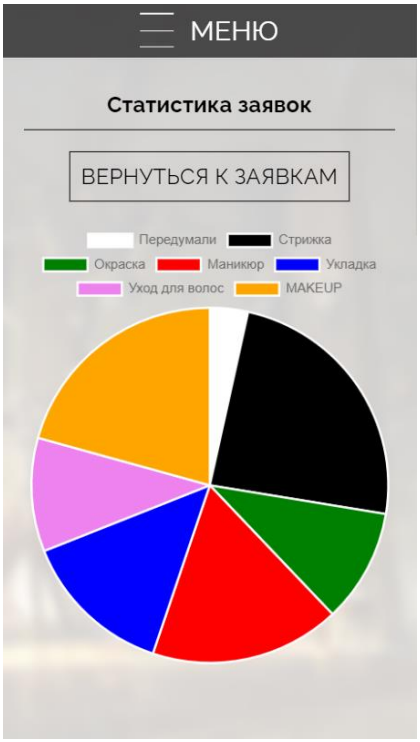

Рисунок А.32 – Страница графика выбираемых услуг (Мобильная версия)

Приложение Б

Переменные окружения

PORT = 5000

DB_NAME = beauty_saloon

DB_USER = postgres

DB_PASSWORD = elman2002

DB_HOST = localhost

DB_PORT = 5432

SECRET_KEY=secret_key_123

Приложение В

Фрагменты программного кода разработанных модулей

Модуль контроллеров пользователей

```
const { where } = require('sequelize');
const ApiError = require('../error/ApiError');
const {User} = require('../models/models');
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');
const generateJwt = (id, login, role) => {
    return jwt.sign(
        {id: id, login, role},
        process.env.SECRET_KEY,
        {expiresIn: '24h'}
    );
}
class UserController {
    async registration(req, res, next) {
        const {email, password, role, name, login} = req.body;
        if(!email || !password) {
            return next(ApiError.badRequest('Некорректный email или пароль'))
        }
        const candidate = await User.findOne({where: {email}});
        if(candidate) {
            return next(ApiError.badRequest('Пользователь с таким email уже существует'));
        }
        const loginCandidate = await User.findOne({where: {login}});
        if(loginCandidate) {
```

```
            return next(ApiError.badRequest('Пользователь с таким логином уже существует'));
        }
        const hashPassword = await bcrypt.hash(password, 5);
        const user = await User.create({email, role, password: hashPassword, name, login});
        const token = generateJwt(user.id, user.login, user.role);
        return res.json({token});
        // return res.json(user);
    }
    async login(req, res, next) {
        const {password, login} = req.body;
        const user = await User.findOne({where: {login}});
        if(!user) {
            return next(ApiError.internal('Пользователь с таким логином не найден'));
        }
        let comparePassword = bcrypt.compareSync(password, user.password);
        if(!comparePassword) {
            return next(ApiError.internal('Указан неверный пароль'));
        }
        const token = generateJwt(user.id, user.login, user.role);
        return res.json({token});
        // return res.json(user);
    }
    async check(req, res, next) {
        // const token = generateJwt(req.user.id, req.user.login, req.user.role);
        // return res.json({token});
```

```
        const login = req.user.login;
        const user = await User.findOne({where: {login}});
        if(!user) {
            return next(ApiError.internal('Пользователь с таким логином не найден'));
        }
        return res.json(user);
    }
    async request(req, res, next) {
        const {login, number} = req.body;
        const user = await User.findOne({where: {login}});
        if(!user) {
            return next(ApiError.internal('Пользователь с таким логином не найден'));
        }
        if(!number) {
            return next(ApiError.badRequest('Указан неверный номер'));
        }
        await user.update({phoneNumber: number});
        await user.update({isRequested: true});
        return res.json({user});
    }
    async requested(req, res) {
        const isRequested = true;
        const users = await User.findAll({where: {isRequested}});
        return res.json(users);
    }
    async accept(req, res) {
```

```
        const {email} = req.body;
        const user = await User.findOne({where: {email}});
        await user.update({phoneNumber: null});
        await user.update({isRequested: false});
        return res.json({user});
    }
    async getRole(req, res) {
        const {login} = req.body;
        const user = await User.findOne({where: {login}});
        if(!user) {
            return next(ApiError.internal('Пользователь с таким логином не найден'));
        }
        return res.json(user.role);
    }
    async editUser(req, res) {
        const {email, name, login, newLogin} = req.body;
        const user = await User.findOne({where: {login}});
        await user.update({name: name});
        await user.update({email: email});
        await user.update({login: newLogin});
        return res.json(user);
    }
}
module.exports = new UserController();
```

Модуль управления зависимостями

```
"name": "server",
```

"version": "1.0.0",
"description": "",
"main": "index.js",
"scripts": {
  "dev": "nodemon index.js"
},
"keywords": [],
"author": "",
"license": "ISC",
"dependencies": {
  "bcrypt": "^5.1.1",
  "cors": "^2.8.5",
  "dotenv": "^16.4.5",
  "express": "^4.19.2",
  "jsonwebtoken": "^9.0.2",
  "pg": "^8.11.5",
  "pg-hstore": "^2.3.4",
  "sequelize": "^6.37.3"
},
"devDependencies": {
  "@babel/cli": "^7.24.1",
  "@babel/core": "^7.24.4",
  "@babel/preset-env": "^7.24.4",
  "babel-cli": "^6.26.0",
  "nodemon": "^3.1.0"
}

Модуль контроллеров заявок на обслуживание

```
const ApiError = require('../error/ApiError');
const {Applications} = require('../models/models');
class ApplicationController {
    async getApplications(req, res, next) {
        const applications = await Applications.findAll();
        let nothingCounter = 0;
        let haircutCounter = 0;
        let coloringCounter = 0;
        let manicureCounter = 0;
        let stylingCounter = 0;
        let caringCounter = 0;
        let makeupCounter = 0;
        applications.forEach(value => {
            if(value.service === 'Ничего не выбрано') {
                nothingCounter++;
            }
            if(value.service === 'Стрижка') {
                haircutCounter++;
            }
            if(value.service  === 'Окраска') {
                coloringCounter++;
            }
            if(value.service  === 'Маникюр') {
                manicureCounter++;
            }
            if(value.service  === 'Укладка') {
                stylingCounter++;
```

```
    }
    if(value.service  === 'Уход для волос') {
        caringCounter++;
    }
    if(value.service  === 'MAKEUP') {
        makeupCounter++;
    }
});
const schedule = [
    {
        service: 'Передумали',
        quantity: nothingCounter
    },
    {
        service: 'Стрижка',
        quantity: haircutCounter
    },
    {
        service: 'Окраска',
        quantity: coloringCounter
    },
    {
        service: 'Маникюр',
        quantity: manicureCounter
    },
    {
        service: 'Укладка',
        quantity: stylingCounter
```

```
            },
            {
                service: 'Уход для волос',
                quantity: caringCounter
            },
            {
                service: 'MAKEUP',
                quantity: makeupCounter
            },
        ];
        return res.json(schedule);
    }
    async setApplications(req, res, next) {
        const {service} = req.body;
        if(!service) {
            return next(ApiError.badRequest('Некорректные данные'));
        }
        const applications = await Applications.create({service});
        return res.json({applications});
    }
}
module.exports = new ApplicationController();
```

Модуль отправки заявок на обслуживание

```
const userToken = localStorage.getItem('token');
const sendButtonForm = document.querySelector('.header__form');
const numberInput = document.querySelector('.header__form__input');
const sendButton = document.querySelector('.header__form__button');
```

```
fetch('http://localhost:5000/api/user/auth', {
    headers: {
        authorization: userToken
    }
}).then(res => {
    return res.json();
}).then(user => {
    console.log(user);
    sendButtonForm.addEventListener('submit', (evt) => {
        evt.preventDefault();
        sendButtonHandler(user);
    })
}).finally(() => {
})

function sendButtonHandler(user) {
    sendButton.textContent = 'Отправка...'
    fetch('http://localhost:5000/api/user/request', {
        method: 'PATCH',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({
            login: user.login,
            number: numberInput.value
        })
    }).then(res => {
        return res.json();
```

```
    }).then(res => {
        console.log(res);
        sendButton.textContent = 'Отправлено';
        sendButton.setAttribute('disabled', true);
    })
}
```

Модуль обработки заявок на обслуживание

```
const requestList = document.querySelector('.request__list');
const requestItemTemplate = document.querySelector('#request');
fetch('http://localhost:5000/api/user/requested')
    .then((res) => {
        return res.json();
    })
    .then((userList) => {
        userList.forEach(user => {
            requestList.prepend(createRequestItem(requestItemTemplate, user));
        });
    })
function createRequestItem(requestItemTemplate, user) {
    const request = requestItemTemplate.content.cloneNode(true);
    const requestItem = request.querySelector('.request__item')
    const requestLogo = request.querySelector('.request__image');
    const requestName = request.querySelector('.request__info--name');
    const requestEmail = request.querySelector('.request__info--email');
    const requestNumber = request.querySelector('.request__info--number');
    const requestButton = request.querySelector('.request__button');
    const service = request.querySelector('.service-select');
```

```
        requestLogo.setAttribute('src',
'https://live.staticflickr.com/65535/51303416171_fd1afa6a84_b.jpg');
        requestName.textContent = user.name;
        requestEmail.textContent = user.email;
        requestNumber.textContent = user.phoneNumber;
        requestItem.classList.add('request__item--not-accepted');
        requestButton.addEventListener('click', evt => {
          evt.target.remove();
          requestItem.classList.remove('request__item--not-accepted');
          fetch('http://localhost:5000/api/user/accept', {
            method: 'PATCH',
            headers: {
              'Content-Type': 'application/json'
            },
            body: JSON.stringify({"email": user.email})
          });
          fetch('http://localhost:5000/api/application/services', {
            method: 'POST',
            headers: {
              'Content-Type': 'application/json'
            },
            body: JSON.stringify({"service": service.value})
          }).then((res) => {
            service.remove();
          })
        });
        return request;
      }
```

Модуль отображения статистики о выбранных услугах

```
const ctx = document.querySelector('#myChart').getContext('2d');
let serviceLabels = new Array();
let serviceData = new Array();
fetch('http://localhost:5000/api/application/services')
    .then(res => {
        return res.json();
    })
    .then(res => {
        res.forEach((element, i) => {
            serviceLabels[i] = element.service;
            serviceData[i] = element.quantity;
        });
        return res;
    })
    .then(res => {
        ctx.canvas.parentNode.style.height = '400px';
        ctx.canvas.parentNode.style.width = '100%';
        const myChart = new Chart(ctx, {
            type: "pie",
            data: {
                labels: serviceLabels,
                datasets: [{
                    label: 'График',
                    data: serviceData,
                    backgroundColor: [
                        'white',
                        'black',
```

```
                'green',
                'red',
                'blue',
                'violet',
            ]
        }]
    },
    options: {
        maintainAspectRatio: false
    }
});
});
```

Модуль авторизации

```
const form = document.querySelector('.form');
const loginInput = form.elements.login;
const passwordInput = form.elements.password;
form.addEventListener('submit', evt => {
    evt.preventDefault();
    fetch('http://localhost:5000/api/user/login', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({
            password: passwordInput.value,
            login: loginInput.value
        })
```

```
    })
    .then(res => {
        return res.json();
    })
    .then(res => {
        if(res.token) {
            localStorage.setItem('token', res.token);
        } else {
            localStorage.removeItem('token');
            alert(res.message);
        }
    })
    .finally(() => {
        if(localStorage.getItem('token')) {
            location.href = `http://${location.host}/html/main-page/main.html`;
        }
    })
});
```

Модуль регистрации

```
const form = document.querySelector('.form');
const nameInput = form.elements.name;
const loginInput = form.elements.login;
const emailInput = form.elements.email;
const passwordInput = form.elements.password;
form.addEventListener('submit', evt => {
    evt.preventDefault();
    fetch('http://localhost:5000/api/user/registration', {
```

```
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({
            email: emailInput.value,
            password: passwordInput.value,
            name: nameInput.value,
            login: loginInput.value
        })
    })
    .then(res => {
        return res.json();
    })
    .then(res => {
        if(res.token) {
            localStorage.setItem('token', res.token);
        } else {
            localStorage.removeItem('token');
            alert(res.message);
        }
    })
    .finally(() => {
        if(localStorage.getItem('token')) {
            location.href = `http://${location.host}/html/main-page/main.html`;
        }
    });
})
```