

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждения образования «БЕЛОРУССКИЙ
ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий
Кафедра Программной инженерии
Специальность 1-40 01 01 Программное обеспечение информационных технологий
Специализация 1-40 01 01-10 Программное обеспечение информационных технологий
(программирование интернет-приложений)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту на тему:
Web-приложение «Салон красоты»

Выполнил студент Гиль Виталия Сергеевна
(Ф.И.О.)

Руководитель проекта ст. преп. О. А. Нистюк
(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой к.т.н., доц. В. В. Смелов
(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой _____

Содержание

| | |
|---|----|
| Введение..... | 4 |
| 1 Постановка задачи и обзор аналогичных решений..... | 5 |
| 1.1 Обзор аналогов..... | 5 |
| 1.1.1 Салон красоты «HqBeauty»..... | 5 |
| 1.1.2 Салон красоты «Изуми»..... | 6 |
| 1.1.3 Салон красоты «Сафина»..... | 7 |
| 1.2 Формирование требований..... | 8 |
| 1.3 Вывод по разделу..... | 9 |
| 2 Проектирование web-приложения..... | 10 |
| 2.1 Архитектура приложения..... | 10 |
| 2.2 Функциональные возможности web-приложения..... | 10 |
| 2.3 Проектирование базы данных..... | 14 |
| 2.4 Вывод по разделу..... | 19 |
| 3 Разработка web-приложения..... | 20 |
| 3.1 Программная платформа Node.js..... | 20 |
| 3.2 Программные библиотеки серверной части..... | 20 |
| 3.3 Структура серверной части..... | 20 |
| 3.4 Реализация функций «гостя»..... | 22 |
| 3.5 Реализация функций роли «USER»..... | 24 |
| 3.6 Реализация функций роли «ADMIN»..... | 26 |
| 3.7 Программные библиотеки клиентской части..... | 29 |
| 3.8 Структура клиентской части..... | 30 |
| 3.9 Вывод по разделу..... | 32 |
| 4 Тестирование web-приложения..... | 34 |
| 4.1 Функциональное тестирование для гостя..... | 34 |
| 4.2 Функциональное тестирование для роли «USER»..... | 35 |
| 4.3 Функциональное тестирование для роли «ADMIN»..... | 36 |
| 4.4 Вывод по разделу..... | 39 |
| 5 Руководство пользователя..... | 40 |
| 5.1 Руководство для гостя..... | 40 |
| 5.2 Руководство для роли «USER»..... | 42 |
| 5.3 Руководство для роли «ADMIN»..... | 44 |
| 5.4 Вывод по разделу..... | 46 |
| Заключение..... | 47 |
| Список используемых источников..... | 48 |
| Приложение А..... | 49 |
| Приложение Б..... | 50 |
| Приложение В..... | 53 |
| Приложение Г..... | 56 |

Введение

В современном мире технологий и удобства онлайн-сервисов, сфера красоты и здоровья также начинает активно использовать цифровые инструменты для улучшения своих услуг и взаимодействия с клиентами. Разработка web-приложений для салонов красоты становится важным направлением в индустрии, предлагая инновационные решения для управления бизнесом и обеспечения удобства клиентам.

Подобные приложения предлагают широкий спектр функций, начиная от онлайн-записи на услуги и управления расписанием мастеров до предоставления персонализированных рекомендаций. Вместе с тем, такие платформы обеспечивают салоны красоты необходимыми инструментами для эффективного управления своими ресурсами, а также повышения уровня сервиса.

Именно в этой связи разработка web-приложения для салона красоты представляет собой интересный и перспективный проект, который объединяет в себе элементы технической инновации, дизайна и понимания потребностей индустрии красоты. Важно учитывать, как технические аспекты разработки, так и аспекты пользовательского опыта, чтобы создать продукт, который будет успешно интегрирован в повседневную деятельность салонов красоты и оценен клиентами.

В результате можно получить продукт, который может значительно улучшить взаимодействие между салонами и их клиентами, повысить эффективность бизнеса и улучшить качество предоставляемых услуг.

Целью данного курсового проектирования является разработка программного средства, предназначенного для обеспечения функциональности салона красоты. Для достижения поставленной цели необходимо выполнить следующие задачи:

- проанализировать аналоги;
- определить требования к программному продукту;
- спроектировать проект web-приложения;
- разработать web-приложение;
- протестировать web-приложение;
- описать руководство пользователя.

1 Постановка задачи и обзор аналогичных решений

1.1 Обзор аналогов

В современном мире набирают популярность сервисы, упрощающие процесс бронирования, записи на услуги и получения информации. Пользователи предпочитают делать все онлайн, вместо траты времени на звонки и поездки в нужное место. Также популярность таких сервисов обусловлена удобством использования, получения доступа в любое время и в любом месте без особых усилий.

1.1.1 Салон красоты «HqBeauty»

Рассмотрим в качестве примера web-приложения салона красоты «HqBeauty» [1]. Интерфейс главной страницы сайта представлен на рисунке 1.1.

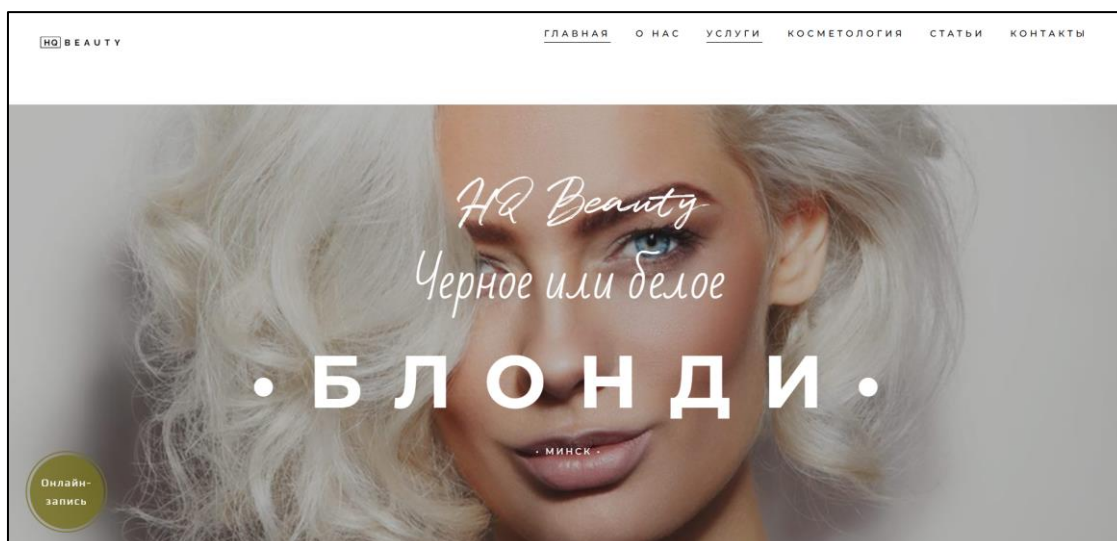


Рисунок 1.1 – Интерфейс главной страницы

Рассмотрим в качестве примера сайт салона красоты «HqBeauty». Интерфейс главной страницы сайта представлен на рисунке 1.1.

Бронь услуги осуществляется в 5 этапов:

- выбор услуги;
- выбор мастера;
- выбор подходящей даты для записи;
- выбор подходящего времени для записи;
- заполнение личной информации;
- подтверждение записи.

Выбор мастера при записи на услугу показан на рисунке 1.2.

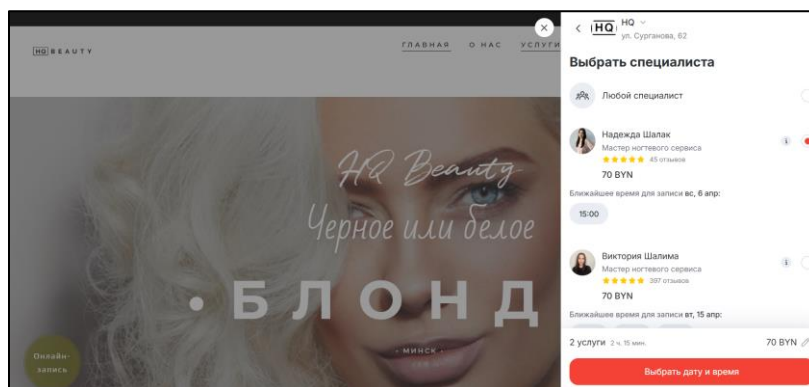


Рисунок 1.2 – Выбор мастера при записи на услугу

Web-приложение выглядит стильно и демонстрирует высокий уровень салона. Понятный интерфейс при заполнении данных, а также проверка их на корректность, позволяют пользователю быстро сделать запись на услугу. В web-приложении присутствует информация о всех предоставляемых услугах и их ценах, а также о мастерах.

Плюсы:

- стильный дизайн;
- понятный интерфейс;
- лёгкая и удобная запись;
- большой выбор услуг.

Минус:

- неочевидность просмотра и оставления отзывов;
- отсутствие упоминаний о возможной обратной связи.

1.1.2 Салон красоты «Изуми»

Следующим аналогом рассмотрим web-приложение салона красоты «Изуми» [2], специализирующийся на современных методах сохранения молодости и красоты, результативности процедур, включая массаж рук, обертывания и консультации по коррекции фигуры и веса. На рисунке 1.3 представлен список услуг салона красоты.

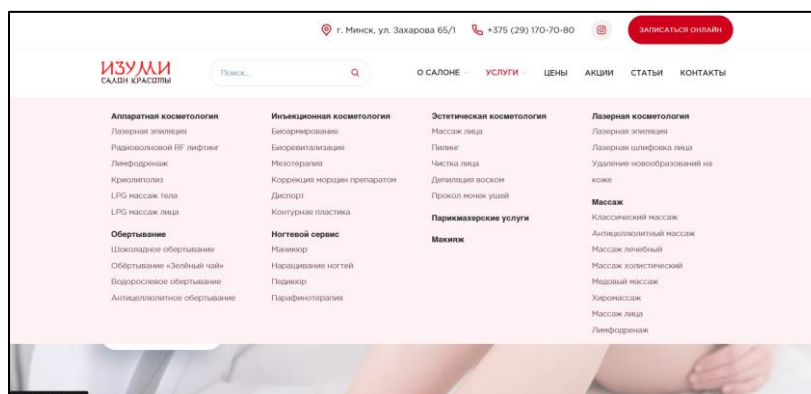


Рисунок 1.3 – Список услуг

Пользователю предоставляется форма для записи на услугу, отображенная на рисунке 1.4. Выбор мастера и просмотр списка доступных мастеров, а также их рейтинга, не предполагаются при заполнении формы, что является минусом для пользователя.

Рисунок 1.4 – Форма записи на услугу

Плюсы:

- стильный дизайн;
- понятный интерфейс;
- удобный формат записи на услугу в виде формы;
- возможность предварительного ознакомления с услугами.

Минус:

- отсутствие списка мастеров;
- отсутствие обратной связи и отзывов.

1.1.3 Салон красота «Сафина»

Далее рассмотрим web-приложение салона красоты «Сафина» [3]. Пользователю предоставляется большой список услуг и мастеров для записи. Запись на услугу происходит в 4 этапа:

- выбор специалиста;
- выбор даты и времени;
- выбор услуги;
- заполнение личной информации;
- подтверждение записи на услугу.

Процесс записи на услугу отображен на рисунке 1.5.

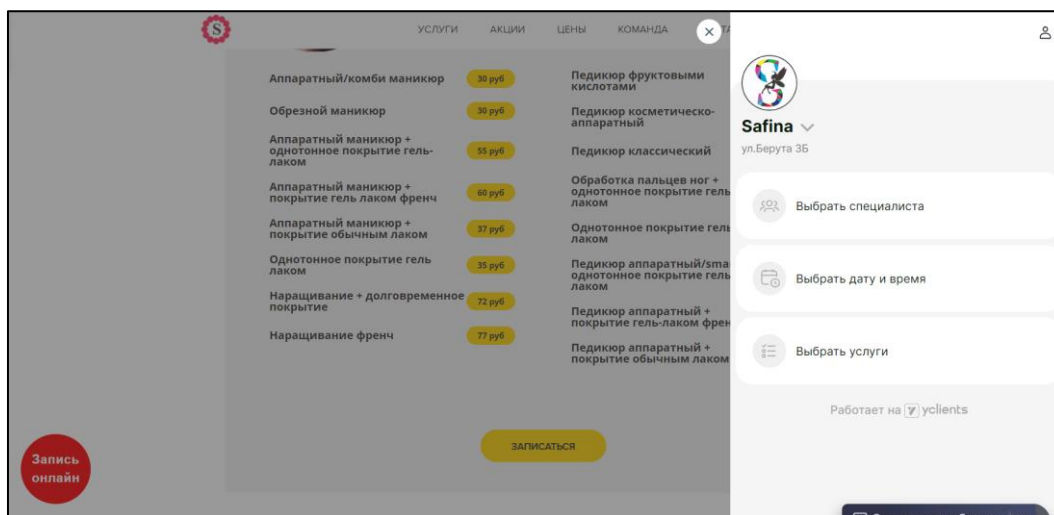


Рисунок 1.5 – Процесс записи на услугу

Плюсы:

- персонализированные рекомендации: система предложения услуг и продуктов на основе предыдущих посещений и предпочтений клиента;
- удобный и понятный формат записи на услугу;
- онлайн-чат с консультантом;
- большой выбор услуг;
- возможность оценить услугу и оставить отзыв.

Минус:

- простой дизайн;

1.2 Формирование требований

Исходя из анализа web-приложений, были определены задачи будущего продукта.

Итогом разработки должно стать web-приложение для просмотра информации об услугах, предлагаемых салоном красоты, а также осуществление записи на эти услуги.

Необходимо разработать несколько интерфейсов: для пользователя и для администратора.

Интерфейс пользователя должен давать возможность просмотра услуг, мастеров, осуществления записи и изменения личной информации о пользователе.

Интерфейс администратора должен позволять проводить операции с содержанием сайта, описаниями. Web-приложение должно быть выполнено в спокойных цветах и оттенках.

Таким образом, в данном курсовом проекте требуется реализовать следующие задачи:

- создать пользовательский интерфейс для взаимодействия с опциями приложения;
- регистрировать и авторизовать пользователей;
- изменять информацию о пользователях;

- просматривать услуги;
- осуществлять запись на услуги;
- осуществлять отмену записи;
- добавлять, изменять и удалять услуги;
- добавлять, изменять и удалять мастеров;
- добавлять и просматривать отзывы.

1.3 Вывод по разделу

На основании анализа аналогов были сформированы основные требования к разрабатываемому web-приложению. Оно должно обеспечивать не только просмотр информации об услугах и мастерах, но и полноценную функциональность записи, включая выбор специалиста, времени и даты, регистрацию пользователей, а также возможность оценки предоставленных услуг. Также предполагается наличие двух интерфейсов: пользовательского и административного, что обеспечит как удобство клиента, так и эффективное управление содержимым со стороны администраторов.

2 Проектирование web-приложения

Разработка архитектуры проекта – важная задача в процессе работы над приложением, потому что в зависимости от неё определяется уровень связности между компонентами приложения, и насколько легко можно будет это приложение расширить.

2.1 Архитектура приложения

Приложение построено на основе клиент-серверной архитектуры.

Клиентская часть (frontend) использует React [13] и MUI Components [17] для реализации пользовательского интерфейса. React предоставляет эффективное модульное решение для создания интерфейса. MUI позволяет использовать готовые шаблоны и компоненты. Чтобы получить актуальные данные и обновить информацию, клиентская часть отправляет HTTP-запросы на сервер.

Серверная часть (backend) реализована с использованием Node.js [20], который является средой выполнения JavaScript на сервере и обеспечивает высокую производительность и масштабируемость. Для создания web-приложений на основе Node.js был выбран гибкий и минималистичный фреймворк Express [4]. Сервер обрабатывает запросы от клиента, включая маршрутизацию запросов к соответствующим обработчикам. Данная часть будет обращаться к базе данных по протоколу TCP.

Для работы с базой данных была выбрана реляционная СУБД PostgreSQL [21]. Node.js сервер использует ORM Prisma [5] для CRUD операций.

Взаимодействие между клиентской и серверной частью приложения осуществляется следующим образом: клиент отправляет HTTP-запросы на сервер с помощью axios. Сервер обрабатывает запросы, выполняет необходимые операции в базе данных и формирует HTTP-ответы в формате JSON. Клиентская часть обрабатывает ответы от сервера и обновляет пользовательский интерфейс. Диграмма развёртывания представлена в [приложении А](#).

Для обеспечения безопасности и управления доступом в web-приложении реализована система аутентификации и авторизации. Аутентификация пользователей осуществляется с помощью токенов JWT (JSON Web Token), которые выдаются при успешном входе в систему. После аутентификации пользователь получает доступ к определённым функциям в зависимости от своей роли (например, пользователь, мастер или администратор). Авторизация реализована на уровне маршрутов сервера, где проверяется наличие и валидность токена, а также соответствие роли пользователя требуемым правам доступа. Это позволяет надёжно разграничить функциональность и защитить конфиденциальные данные.

2.2 Функциональные возможности web-приложения

Функциональные возможности web-приложения отображены в диаграмме вариантов использования, представленной на рисунке 2.1.

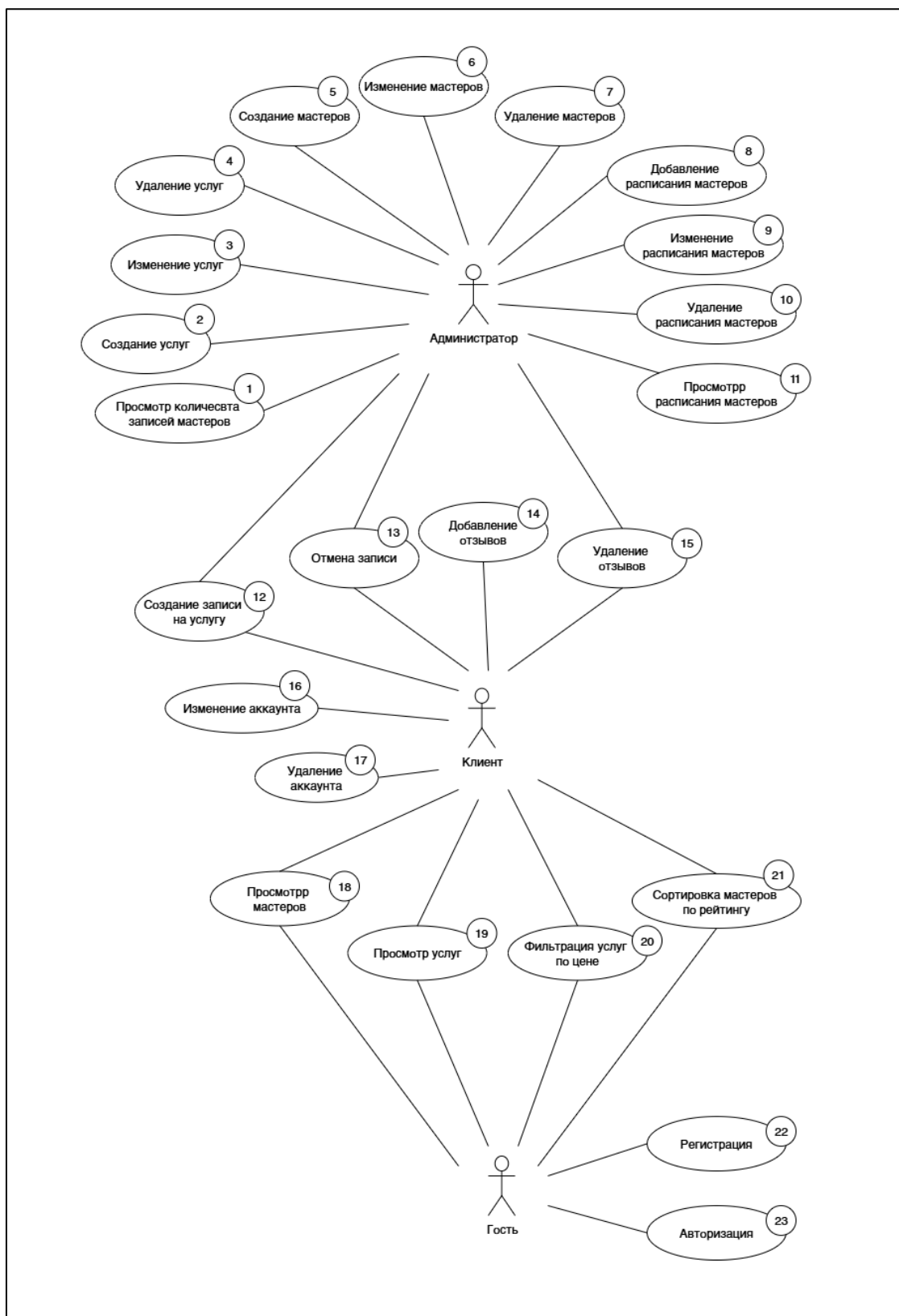


Рисунок 2.1 – Диаграмма вариантов использования web-приложения

Данная диаграмма вариантов использования наглядно демонстрирует функциональность web-приложения «Upgrade» для различных ролей пользователей, таких как администратор, гость и пользователь. Перечень ролей и их назначение приведены в таблице 2.1.

Таблица 2.1 – Назначение ролей пользователей в web-приложении

| Роль | Назначение |
|---------------|---|
| Администратор | Создание, изменение, удаление услуг; Создание, изменение, удаление мастеров; Создание, изменение, удаление расписания мастеров; Удаление отзывов; Просмотр количества записанных клиентов каждого мастера; Просмотр расписания мастеров; |
| Пользователь | Создание и отмена записей на услуги; Создание и удаление отзывов; Просмотр мастеров и услуг; Сортировка мастеров по рейтингу; Фильтрация услуг по цене; Изменение и удаление аккаунта |
| Гость | Регистрация; Авторизация; Просмотр мастеров и услуг; |

Функциональные возможности пользователя с ролью «администратор» представлены в таблице 2.2.

Таблица 2.2 – Функциональные возможности пользователя с ролью «администратор»

| № | Вариант использования | Описание |
|----|--------------------------------------|---|
| 1 | Просмотр количества записей мастеров | Администратор может выбрать дату начала и дату окончания периода для просмотра количества записей к мастерам. |
| 2 | Создание услуг | Администратор может создать новую услугу, корректно заполнив данные. |
| 3 | Изменение услуг | Администратор может изменить существующую услугу, корректно заполнив данные. |
| 4 | Удаление услуг | Администратор может удалить услугу. |
| 5 | Создание мастера | Администратор может создать нового мастера, корректно заполнив данные. |
| 6 | Изменение мастера | Администратор может изменить существующего мастера, корректно заполнив данные. |
| 7 | Удаление мастера | Администратор может удалить мастера. |
| 8 | Добавление расписания мастеров | Администратор может добавить мастеру время работы, указав день, а также время начала и окончания работы. |
| 9 | Изменение расписания мастеров | Администратор может изменить существующее расписание мастера. |
| 10 | Удаление расписания мастеров | Администратор может удалить существующее расписание мастера |

Продолжение таблицы 2.2

| | | |
|----|------------------------------|---|
| 11 | Просмотр расписания мастеров | Администратор может просмотреть расписание мастера, выбрав конкретный день. |
|----|------------------------------|---|

Далее рассмотрим функциональные возможности для пользователя, которые представлены в таблице 2.3.

Таблица 2.3 – Функциональные возможности пользователя

| № | Вариант использования | Описание |
|----|---------------------------|---|
| 12 | Создание записи на услугу | Пользователь может создать запись на услугу, выбрав услугу, время мастера и доступные слоты для записи. |
| 13 | Отмена записи | Пользователь может отменить запись на услугу. |
| 14 | Добавление отзыва | Пользователь может добавить отзыв конкретному мастеру. |
| 15 | Удаление отзыва | Пользователь может удалить написанный им отзыв. |
| 16 | Изменение аккаунта | Пользователь может изменить личные данные, включая имя, номер телефона, электронную почту и пароль. |
| 17 | Удаление аккаунта | Пользователь может удалить свой аккаунт |

Далее рассмотрим функциональные возможности для пользователя с ролью «гость», которые представлены в таблице 2.4.

Таблица 2.4 – Функциональные возможности пользователя с ролью «гость»

| № | Вариант использования | Описание |
|----|---------------------------------|--|
| 18 | Просмотр мастеров | Гость может просмотреть существующих мастеров. |
| 19 | Просмотр услуг | Гость может просматривать существующие услуги. |
| 20 | Фильтрация услуг по цене | Гость может фильтровать услуги по цене, указав минимальную и максимальную стоимость услуги. |
| 21 | Сортировка мастеров по рейтингу | Гость может сортировать мастеров по рейтингу, выбрав сортировку по возрастанию или убыванию. |
| 22 | Регистрация | Гость может создать учетную запись, заполнив необходимые данные, чтобы получить доступ к полному функционалу web-приложения. |
| 23 | Авторизация | Процесс предоставление пользователям прав на выполнение определенных действий с использованием учетных данных, таких как логин и пароль. |

У гостя есть возможности регистрации, просмотра рейтинга и отзывов сотрудника, просмотра сотрудников и услуг, предоставляемых этими сотрудниками.

Роль клиента позволяет делать то же самое, что и гость, но еще можно авторизоваться, оставлять отзывы, записываться на услугу, а также просматривать свой профиль и редактировать информацию о себе.

Администратор может производить различные действия с услугами и сотрудниками, такие как добавление, изменение и удаление, а также просмотр информации.

2.3 Проектирование базы данных

В разрабатываемом программном средстве существует возможность добавления, изменения данных в базе данных, взаимодействие происходит через запросы.

Для реализации поставленной в курсовом проектировании задачи была создана база данных «Upgrade». Для её создания использовалась система управления реляционными базами данных PostgreSQL.

Для базы данных «Upgrade» было разработано 9 таблиц. Взаимосвязь всех таблиц проектируемой базы данных представлена на рисунке 2.3.

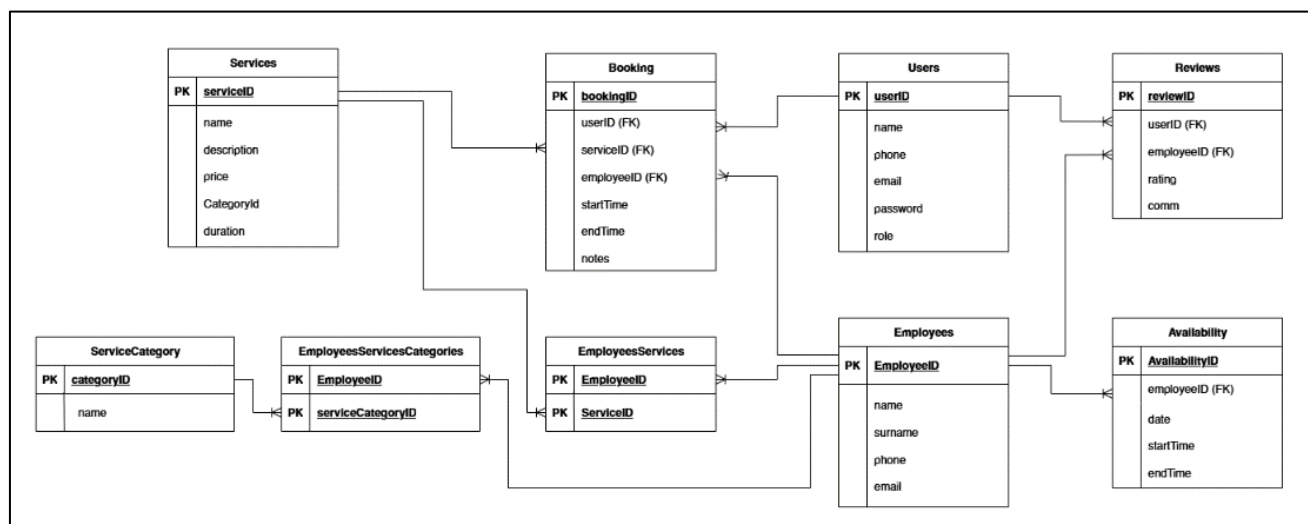


Рисунок 2.2 – Взаимосвязь таблиц базы данных

Схема отражает логическую структуру хранения и обработки данных в системе. Такая организация обеспечивает целостность данных и позволяет эффективно реализовать все предусмотренные функции web-приложения. Назначение таблиц базы данных представлено в таблице 2.5.

Таблица 2.5 – Таблицы базы данных

| Название таблицы | Описание |
|------------------|---|
| Users | Содержит информацию о существующих пользователях |
| Roles | Содержит информацию о предоставляемых ролях, таких как «ADMIN» и «USER» |
| Services | Содержит информацию об услугах, предоставляемых салоном красоты |

Продолжение таблицы 2.5

| | |
|-----------------------------|---|
| Employees | Содержит информацию о сотрудниках салона красоты |
| EmployeesServices | Содержит информацию о связи сотрудника с предоставляемыми им услугами |
| EmployeesServicesCategories | Содержит информацию о связи сотрудника с категориями предоставляемых услуг |
| Booking | Содержит информацию о записях пользователей на услуги |
| Reviews | Содержит информацию об отзывах на сотрудников салона |
| Availability | Содержит информацию о доступных временных слотах мастера для записи на услугу |

Таблица Users представляет собой данные о пользователе и содержит поля, представленные в таблице 2.6.

Таблица 2.6 – Описание таблицы Users

| Название поля | Тип данных | Описание |
|---------------|-------------------|--|
| UserID | integer | идентификатор пользователя, первичный ключ |
| name | text | имя пользователя |
| phone | text | телефон пользователя |
| email | text | адрес электронной почты пользователя |
| role | Роль (User/Admin) | Роль пользователя |
| password | text | пароль пользователя |

Таблица ServiceCategory представляет собой данные о категориях услуг и содержит поля, представленные в таблице 2.7.

Таблица 2.7 – Описание таблицы ServiceCategory

| Название поля | Тип данных | Описание |
|---------------|------------|---|
| categoryID | integer | идентификатор категории, первичный ключ |
| name | text | название категории |

Таблица Services представляет собой данные об услугах и содержит поля, представленные в таблице 2.8.

Таблица 2.8 – Описание таблицы Services

| Название поля | Тип данных | Описание |
|---------------|------------|--------------------------------------|
| seviceID | integer | Идентификатор услуги, первичный ключ |

Продолжение таблицы 2.8

| | | |
|-------------|---------|--|
| name | text | Название услуги |
| description | text | Описание услуги |
| price | decimal | Стоимость услуги |
| duration | integer | Длительность услуги, по умолчанию устанавливается 60 |
| categoryID | integer | Идентификатор категории услуг |

Таблица Employees представляет собой данные о сотрудниках и содержит поля, представленные в таблице 2.9.

Таблица 2.9 – Описание таблицы Employees

| Название поля | Тип данных | Описание |
|---------------|------------|------------------------------------|
| employeeID | integer | Идентификатор сотрудника салона |
| name | text | Имя сотрудника |
| surname | text | Фамилия сотрудника |
| email | text | Адрес электронной почты сотрудника |
| photo | bytea | Фотография мастера |

Таблица EmployeesServices представляет собой данные о связи мастера и предоставляемых им услугами, содержит поля представленные в таблице 2.10.

Таблица 2.10 – Описание таблицы EmployeesServices

| Название поля | Тип данных | Описание |
|---------------|------------|-----------------------|
| employeeID | integer | Идентификатор мастера |
| serviceID | integer | Идентификатор услуги |

Таблица EmployeesServiceCategories представляет собой данные о связи мастера и категориях предоставляемых услуг, содержит поля, представленные в таблице 2.11.

Таблица 2.11 – Описание таблицы EmployeesServiceCategories

| Название поля | Тип данных | Описание |
|-------------------|------------|--------------------------------|
| employeeID | integer | Идентификатор мастера |
| serviceCategoryID | integer | Идентификатор категории услуги |

С помощью таблицы EmployeesServiceCategories можно гибко управлять специализациями мастеров и отображать только релевантные категории услуг при записи.

Таблица Booking представляет собой информацию о записях клиентов на услуги и содержит поля, представленные в таблице 2.12.

Таблица 2.12 – Описание таблицы Booking

| Название поля | Тип данных | Описание |
|---------------|------------|---|
| bookingID | integer | Идентификатор записи на услугу |
| userID | integer | Идентификатор пользователя |
| employeeID | integer | Идентификатор сотрудника |
| serviceID | integer | Идентификатор услуги |
| startTime | timeStamp | Время начала проведения услуги |
| endTime | timeStamp | Время окончания проведения услуги |
| notes | text | Заметки, который содержат дополнительные записи |

Таблица Reviews представляет собой информацию об отзывах, оставленных клиентом на мастера, содержит поля представленные в таблице 2.13.

Таблица 2.13 – Описание таблицы Reviews

| Название поля | Тип данных | Описание |
|---------------|------------|---|
| reviewID | integer | Идентификатор отзыва |
| userID | integer | Идентификатор пользователя, оставившего отзыв |
| employeeID | integer | Идентификатор сотрудника |
| rating | integer | Рейтинг |
| comm | text | Комментарии, написанные в отзыве |

Таблица Availability представляет собой информацию о записях клиентов на услуги и содержит поля, представленные в таблице 2.14.

Таблица 2.14 – Описание таблицы Availability

| Название поля | Тип данных | Описание |
|----------------|------------|--------------------------------|
| availabilityID | integer | Идентификатор записи на услугу |
| employeeID | integer | Идентификатор пользователя |
| date | timeStamp | Дата работы мастера |
| startTime | timeStamp | Время начала работы мастера |
| endTime | timeStamp | Дата окончания работы мастера |

В таблице 2.15 представлены связи, установленные между моделями.

Таблица 2.15 – Таблица связей моделей

| Модель А | Связь | Модель Б | Тип связи |
|-----------------|------------------|-----------------|---|
| Users | 1 ко многим | Reviews | Один пользователь и много отзывов |
| Users | 1 ко многим | Booking | Один пользователь и много записей на услуги |
| Employees | 1 ко многим | Reviews | Один сотрудника и много отзывов |
| Employees | 1 ко многим | Booking | Один сотрудника и много записей на услуги |
| Employees | 1 ко многим | Availability | Один сотрудника и много интервалов доступных слотов |
| Employees | многие ко многим | ServiceCategory | Сотрудники принадлежат к многим категориям |
| Services | 1 ко многим | Booking | Одна услуга может иметь много бронирований |
| Services | многие к одному | ServiceCategory | Услуга принадлежит одной категории |
| ServiceCategory | 1 ко многим | Services | Категория включает много услуг |
| ServiceCategory | многие ко многим | Employees | Категория включает многих сотрудников |
| Booking | многие к одному | Users | Бронирование привязано к пользователю |
| Booking | многие к одному | Employees | Бронирование связано с сотрудником |
| Booking | многие к одному | Services | Бронирование связано с услугой |
| Reviews | многие к одному | Users | Отзыв написан пользователем |
| Reviews | многие к одному | Employees | Отзыв адресован сотруднику |
| Availability | многие к одному | Employees | Интервал принадлежит сотруднику |

Листинг создания базы данных, используя ORM Prisma, представлен в [приложении Б](#).

Таким образом были созданы все необходимые для работы приложения таблицы в базе данных.

2.4 Вывод по разделу

В данном разделе была подробно рассмотрена архитектура и функциональные возможности web-приложения «Upgrade», а также выполнено проектирование базы данных, обеспечивающей хранение и обработку необходимых данных. На основе анализа ролей пользователей — «гость», «пользователь» и «администратор» сформирован полный и четкий перечень операций, доступных каждой из ролей, что позволяет обеспечить безопасность и разграничение прав доступа.

Подробное описание функциональных возможностей каждой категории пользователей отражает реальные бизнес-процессы и требования к системе, включая управление услугами, сотрудниками, расписанием, записями и отзывами. Такая структура позволяет обеспечить удобство использования приложения для разных категорий пользователей, от гостей, которые могут ознакомиться с предлагаемыми услугами, до администраторов, имеющих полный контроль над содержанием и работой сервиса.

3 Разработка web-приложения

3.1 Программная платформа Node.js

Для серверной части проекта была выбрана платформа Node.js с использованием фреймворка Express, который обеспечивает лёгкость и гибкость в разработке web-приложений. Проект структурирован по модулям, каждая из которых отвечает за отдельную часть логики — маршруты, контроллеры, middleware, модели и утилиты. На рисунке 3.1 представлена структура проекта.

3.2 Программные библиотеки серверной части

В процессе разработки серверной части web-приложения для обеспечения её функциональности и повышения эффективности работы системы были использованы программные библиотеки, представленные в таблице 3.1.

Таблица 3.1

| Библиотека | Версия | Назначение |
|-------------------|----------|--|
| Express [4] | v4.19.2 | Основной web-фреймворк для создания серверного API и маршрутов |
| Prisma [5] | v5.12.1 | Основной web-фреймворк для создания серверного API и маршрутов |
| Jsonwebtoken [6] | v9.0.2 | Генерация и валидация JWT-токенов для аутентификации |
| Cors [7] | v2.8.5 | Разрешение CORS-запросов между клиентом и сервером |
| Bcryptjs [8] | v2.4.1 | Хеширование паролей |
| Cookie-parser [9] | v1.4.6 | Парсинг cookies из HTTP-запросов |
| Dayjs [10] | v1.11.11 | Для работы с датами |
| Date-fns [11] | v4.1.0 | Для работы с датами |
| Date-fns-tz [12] | v3.2.0 | Для работы с часовыми поясами |

Использование перечисленных библиотек позволило обеспечить надежную и гибкую архитектуру серверной части приложения. Благодаря им были реализованы ключевые механизмы маршрутизации, работы с базой данных, аутентификации пользователей.

3.3 Структура серверной части

Структура серверной части приложения, реализованная на платформе Node.js, включает в себя ключевые элементы, которые обеспечивают корректную работу и расширяемость приложения.

В таблице 3.2 приведён список директорий проекта разработки web-приложения и назначение файлов, хранящихся в этих директориях.

Таблица 3.2 – Директории серверной части проекта

| Директория | Назначение |
|-------------|--|
| Controllers | Содержит обработчики логики запросов. Контроллеры принимают данные из маршрутов, обрабатывают их и вызывают соответствующие сервисы или работу с базой данных. |
| Middlewares | Директория «Middlewares» содержит промежуточные обработчики, которые нужны для проверки JWT токена и доступности действия исходя из роли пользователя. |
| Rotes | Содержит роутеры, которые реализуют маршрутизацию |
| Prisma | Содержит конфигурации и миграции ORM-библиотеки Prisma: схему базы данных (schema.prisma), миграции и клиент Prisma для взаимодействия с БД. |

Таблица соответствия маршрутов контроллерам и функциям в исходном коде представлена в таблице 3.3.

Таблица 3.3 – Контроллеры и функции маршрутов

| Контроллер | Метод | Маршрут | Метод контроллера |
|--------------------|--------|---------------------------------|------------------------------|
| authController | GET | /getuser | getUserData |
| authController | POST | /registration | registration |
| authController | POST | /login | login |
| authController | POST | /logout | logout |
| authController | PUT | /upduser/:id | updateUser |
| authController | DELETE | /delete/:id | deleteUser |
| bookingController | GET | /getuser/:serviceID | getAvailableDatesForService |
| bookingController | GET | /getbook | Get UserRegistrations |
| bookingController | GET | /availability/:serviceID/:date | getEmployeesAndAvailability |
| bookingController | GET | /getBookings-ByEmpl/:employeeID | getBookingsByEmployeeID |
| bookingController | POST | /booking | getUserRegistrations |
| bookingController | DELETE | /delbook/:registrationID | cancelRegistration |
| employeeController | GET | /getempl | getEmployees |
| employeeController | GET | /getemplbyser/:serviceID | getEmployeesByService |
| employeeController | GET | /getemplbyid/:id | getEmployeeById |
| employeeController | GET | /getavailable-dates/:employeeID | getAvailableDatesForEmployee |

Продолжение таблицы 3.3

| | | | |
|--------------------|--------|--------------------|----------------------------|
| employeeController | POST | /addempl | addEmployee |
| employeeController | PUT | /updempl/:id | updateEmployee |
| employeeController | DELETE | /delemp/:id | delEmployee |
| employeeController | DELETE | /delshedule/:id | delShedule |
| reviewController | GET | /getrating/:id | getAverageRatingByEmployee |
| reviewController | GET | /getrewempl/:id | getReviewByEmployee |
| reviewController | POST | /addreview | addReview |
| reviewController | DELETE | /delreview | delReview |
| serviceController | GET | /getservices | getServices |
| serviceController | GET | /getservbyid/:id | getServById |
| serviceController | GET | /categories | getCategories |
| serviceController | GET | /serviceByCategory | getServicesByCategory |
| serviceController | POST | /addservice | addService |
| serviceController | PUT | /updserv/:id | getServById |
| serviceController | DELETE | /delserv/:id | delServ |

Таблица 3.3 демонстрирует соответствие между HTTP-маршрутами, методами запросов и функциями контроллеров, реализованных в серверной части приложения. Такая структура обеспечивает чёткое разделение логики и упрощает поддержку, масштабирование и тестирование API-интерфейса.

3.4 Реализация функций «гостя»

Для гостя доступна регистрация, которая позволяет ему создать учетную запись в системе. Этот процесс реализован в методе Registration контроллера AuthController. Реализация метода представлена на листинге 3.1.

```

async registration(req, res) {
  try {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      console.log("Неверно введены данные");
      return res
        .status(400)
        .json({ message: "Неверно введены данные", errors });
    }
    const { name, phone, email, password, role } = req.body;
    if (!name || !phone || !email || !password) {
      console.log("All fields are required");
      return res
        .status(400)
        .json({ message: "Все поля должны быть заполнены" });
    }
  }
}

```

```

    const existingUser = await clientPr.users.findUnique({
      where: { email },
    });
    if (existingUser) {
      console.log("Email уже используется");
      return res.status(409).json({ message: "Email уже использу-
ется" });
    }
    const hashedPassword = await bcrypt.hash(password, 5);

    const userRole =
      role && role.toUpperCase() === "ADMIN" ? "ADMIN" : "USER";
    const newUser = await clientPr.users.create({
      data: {
        name,
        phone,
        email,
        password: hashedPassword,
        role: userRole,
      },
    });
    res.status(201).json({
      message: "User зарегистрирован успешно",
      user: {
        userID: newUser.userID,
        name: newUser.name,
        phone: newUser.phone,
        email: newUser.email,
        role: newUser.role,
      },
    });
    console.log("User зарегистрирован успешно");
  } catch (error) {
    console.error(error);
    res.status(400).json({ message: "Ошибка регистрации" });
  }
}

```

Листинг 3.1 – Реализация метода registration

Метод Registration контроллера Auth обрабатывает процесс регистрации нового пользователя. Он выполняет валидацию входных данных, проверку уникальности email, хеширование пароля с использованием bcrypt и сохранение пользователя в базе данных через Prisma ORM. В случае успешной регистрации клиенту возвращается сообщение об успехе и информация о созданном пользователе.

Метод login, представленный в листинге 3.2 реализует процесс аутентификации пользователя.

```

async login(req, res) {
  try {
    const { email, password } = req.body;
    const user = await clientPr.users.findUnique({
      where: { email },
    });
  }
}

```

```

    });
    const isPasswordValid = bcrypt.compareSync(password, user.password);
    if (!user || !isPasswordValid) {
        console.log("Неверный логин или пароль");
        return res.status(401).json({ message: "Неверный логин или пароль" });
    }
    const token = generateAccessToken(user.userID, user.role);
    console.log("-----");
    console.log("userID ", user.userID);
    return res.json({ token });
} catch (error) {
    // next(error);
    console.log(error);
    res.status(400).json({ message: "Ошибка входа" });
}
}

```

Листинг 3.2 – Реализация метода login

Метод login отвечает за процесс входа пользователя в систему. Он проверяет наличие пользователя по email, сверяет введенный пароль с сохранённым хешем и, при успешной аутентификации, возвращает клиенту JWT-токен для дальнейшего доступа к защищённым маршрутам.

3.5 Реализация функций роли «USER»

Основным функционалом пользователя является создание записей на услугу, а также отправка отзывов на сотрудника салона красоты. Функции для создания записей на услуги реализованы в bookingController. В [приложении В](#) представлена реализация функции addBooking, которая позволяет создать запись на выбранную услугу. В листинге 3.3 представлена реализации функции cancelRegistration.

```

async cancelRegistration(req, res) {
    try {
        const { registrationID } = req.params;
        console.log("Идентификатор брони:", registrationID);
        const booking = await clientPr.booking.findUnique({
            where: {
                bookingID: parseInt(registrationID, 10),
            },
        });
        if (!booking) {
            console.log("Бронирование не найдено");
            return res.status(404).json({ message: "Бронирование не найдено" });
        }
        const { employeeID, startTime, endTime } = booking;
        const moscowDate = new Date(startTime);
        moscowDate.setUTCHours(3, 0, 0, 0); // Приведение к началу дня
        await clientPr.schedule.create({

```

```

        data: {
            employee: {
                connect: { employeeID },
            },
            date: moscowDate,
            startTime: new Date(startTime),
            endTime: new Date(endTime),
        },
    });
    await clientPr.booking.delete({
        where: {
            bookingID: booking.bookingID,
        },
    });
    res.status(200).json({
        message: "Бронирование успешно отменено",
    });
    console.log("Бронирование успешно отменено");
} catch (error) {
    console.error("Ошибка при отмене бронирования:", error);
    res.status(400).json({ message: "Ошибка при отмене бронирования"
});
}
}

```

Листинг 3.3 – Реализация метода cancelRegistration

Функция `cancelRegistration` обрабатывает запрос на отмену бронирования услуги. Она находит бронирование по переданному ID, возвращает освободившееся время в расписание сотрудника, удаляет запись из базы данных и отправляет пользователю подтверждение об успешной отмене.

В листинге 3.4 представлена реализация функции `addReview`, которая позволяет пользователю написать отзыв сотруднику.

```

async addReview(req, res) {
    try {
        const { userID, employeeID, rating, comm } = req.body;
        if (!rating) {
            console.log("Оценка должна быть указана");
            return res.status(400).json({ message: "Оценка должна быть
указана" });
        }
        const employee = await clientPr.employees.findFirst({
            where: {
                employeeID: employeeID,
            },
        });
        if (!employee) {
            return res.status(404).json({ message: "Сотрудник не найден"
});
        }
        const newReview = await clientPr.reviews.create({
            data: {

```



```

        userID,
        employeeID,
        rating,
        comm,
    },
    });
    res.status(201).json({
        message: "Отзыв успешно добавлен",
        review: newReview,
    });
} catch (error) {
    console.error("Ошибка добавления отзыва:", error);
    res.status(500).json({ message: "Ошибка добавления отзыва" });
}
}

```

Листинг 3.4 – Реализация метода addReview

Функция addReview отвечает за добавление отзыва пользователя о сотруднике салона красоты. Она принимает ID пользователя, ID сотрудника, оценку и комментарий, проверяет наличие оценки и существование сотрудника, после чего сохраняет отзыв в базе данных и возвращает подтверждение успешного создания.

3.6 Реализация функций роли «ADMIN»

Пользователь с ролью «Администратор» обладает расширенными правами доступа, позволяющими выполнять полное управление данными системы. В его функционал входит добавление, редактирование и удаление услуг, сотрудников (мастеров), а также управление их расписанием. Администратор может просматривать графики работы сотрудников, удалять отзывы пользователей и анализировать статистику по количеству записей к каждому мастеру.

В листинге 3.5 представлена реализация функции addService, которая позволяет администратору создать услугу.

```

async addService(req, res) {
    const { name, description, price, categoryID, duration } =
    req.body;
    if (!name || price === undefined) {
        console.log("Название услуги и цена обязательны");
        return res.status(400).json({ error: "Название услуги и цена
    обязательны" });
    }
    try {
        const existingService = await clientPr.servics.findFirst({
            where: { name },
        });

        if (existingService) {
            console.log("Услуга с таким именем уже существует");
            return res.status(409).json({ message: "Услуга с таким
    именем уже существует" });
        }
    }
}

```

```

    }
    const priceFormatted = parseFloat(price).toFixed(2);
    const serviceData = {
        name,
        description,
        price: priceFormatted,
        categoryID: categoryID ? parseInt(categoryID, 10) : null,
    };
    if (duration !== undefined && duration !== null && !isNaN(parseInt(duration, 10))) {
        serviceData.duration = parseInt(duration, 10);
    } else if (duration !== undefined && duration !== null) {
        console.log("Некорректное значение для длительности услуги.");
        return res.status(400).json({ error: "Длительность услуги должна быть числом (в минутах)." });
    }
    const newService = await clientPr.services.create({
        data: serviceData,
        include: {
            category: true,
        },
    });
    res.status(201).json({ message: "Услуга добавлена успешно: ", service: newService });
    console.log("Услуга добавлена успешно: ", newService);
} catch (error) {
    console.error("Ошибка при добавлении услуги:", error);
    if (error.code === 'P2003') {
        return res.status(400).json({ error: "Указанная категория услуги не существует." });
    }
    res.status(500).json({ error: "Ошибка на сервере при добавлении услуги" });
}
}

```

Листинг 3.5 – Реализация функции addService

Функция `addService` отвечает за добавление новой услуги в систему. Она проверяет обязательные поля (название и цену), а также проверяет, существует ли уже услуга с таким названием. Если все данные корректны, услуга добавляется в базу данных с указанной категорией и длительностью, при этом происходит форматирование цены и проверка на корректность длительности. В случае ошибок, например, если категория не существует или длительность указана некорректно, возвращаются соответствующие сообщения об ошибке.

Администратор также может создавать новых сотрудников и управлять ими. Функция создания сотрудника представлена в [приложении Г](#). В листинге 3.6 представлена реализация функции `delEmployee`, которая позволяет удалять сотрудника из системы.

```

async delEmployee(req, res) {
    const employeeId = parseInt(req.params.id, 10);

```

```

    if (isNaN(employeeId)) {
        return res.status(400).json({ error: "Неверный идентификатор
отзыва" });
    }
    try {
        await clientPr.schedule.deleteMany({
            where: {
                employeeID: employeeId,
            },
        });
        const deletedEmployee = await clientPr.employees.delete({
            where: {
                employeeID: employeeId,
            },
        });
        res.status(200).json({ message: "Сотрудник удален успешно",
deletedEmployee });
    } catch (error) {
        if (error.code === "P2025") {
            res.status(404).json({ error: "Сотрудник не найден" });
        } else {
            console.error("Ошибка при удалении сотрудника:", error);
            res.status(500).json({ error: "Ошибка при удалении сотрудника"
});
        }
    }
}
}

```

Листинг 3.6 – Реализация функции delEmployee

Функция delEmployee предназначена для удаления сотрудника из системы. Перед удалением сотрудника функция также удаляет все записи его расписания. В случае некорректного идентификатора или отсутствия сотрудника в базе возвращается соответствующее сообщение об ошибке. При успешном удалении клиент получает подтверждение и информацию об удалённом сотруднике.

Администратор также управляет расписанием сотрудников, добавляя, обновляя и удаляя расписание. Функция удаления расписания у сотрудника delShedule представлена в листинге 3.7.

```

async delShedule(req, res) {
    const employeeID = parseInt(req.params.id, 10);
    if (isNaN(employeeID)) {
        return res.status(400).json({ message: "Неверный ID сотрудника"
});
    }
    try {
        // Удаление Booking, если нужно очистить их тоже
        await clientPr.booking.deleteMany({
            where: { employeeID },
        });
        await clientPr.availability.deleteMany({
            where: { employeeID },
        });
    }
}

```

```

    });
    return res.json({ message: "Расписание и связанные записи уда-
лены успешно." });
  } catch (error) {
    console.error("Ошибка при удалении расписания:", error);
    return res.status(500).json({ message: "Ошибка при удалении
расписания." });
  }
}
}
async;
}

```

Листинг 3.7 – Реализация функции delSchedule

Функция delSchedule предназначена для удаления расписания выбранного сотрудника по его идентификатору. В процессе выполнения также удаляются все связанные с ним записи бронирования, обеспечивая полную очистку данных, связанных с рабочим графиком сотрудника. В случае ошибок возвращается сообщение об их причине.

3.7 Программные библиотеки клиентской части

Основную часть приложения составляет клиентская часть, потому что она обеспечивает взаимодействие пользователя с продуктом. В процессе разработки клиентской части web-приложения были задействованы программные библиотеки, представленные в таблице 3.4.

Таблица 3.4 – Программные библиотеки клиентской части

| Библиотека | Версия | Назначение |
|--------------------------|-----------|---|
| React [13] | v18.3.10 | Основная библиотека для построения пользовательского интерфейса |
| React-dom [14] | v18.3.1 | Работа с DOM-деревом в браузере, рендеринг компонентов |
| React-router-dom [15] | v6.22.3 | Маршрутизация между страницами (SPA) |
| Axios [16] | v1.6.8 | HTTP-клиент для взаимодействия с сервером (REST API) |
| Jwt-decode [10] | v4.0.0 | Расшифровка JWT-токенов на клиенте |
| Mui Material[17] | v5.15.15s | Компоненты Material UI (новая версия от MUI) |
| @mui/x-date-pickers [17] | v7.2.0 | Расширенные компоненты выбора даты/времени |
| Dayjs [10] | v1.11.11 | Для работы с датами |
| Mobx [18] | v6.12.3 | Система управления состоянием с реактивностью |
| mobx-react-lite [19] | v4.0.7 | React bindings для MobX |

В приведённой таблице представлены основные библиотеки, используемые в клиентской части проекта, а также их назначение и версии. Эти инструменты обеспечивают построение удобного и функционального пользовательского интерфейса, а также надёжное взаимодействие с серверной частью через REST API.

3.8 Структура клиентской части

Основную часть приложения составляет клиентская часть, потому что она обеспечивает взаимодействие пользователя с продуктом. Для клиентской части приложения была использована библиотека React и фреймворк MUI, который позволяет использовать готовые шаблоны и компоненты. С помощью React можно легко создавать интерактивные пользовательские интерфейсы.

Структура директорий проекта и их описания представлена в таблице 3.5.

Таблица 3.5 – Структура директорий клиентской части

| Директория | Назначение |
|------------|---|
| Components | Содержит переиспользуемые пользовательские интерфейсные компоненты, такие как формы, карточки, кнопки и другие элементы интерфейса. |
| Pages | Включает в себя страницы приложения, каждая из которых соответствует определенному маршруту (роуту), например, главная страница, страница авторизации, профиля и т.д. |
| Public | Содержит статические ресурсы, доступные напрямую, такие как изображения, стили, иконки и другие файлы. В данной директории хранятся фотографии, а также index.html |
| Images | Хранит изображения для сотрудников салона красоты |

Представленная структура клиентской части проекта обеспечивает удобную организацию кода и разделение ответственности между различными модулями. Такое разграничение позволяет упростить поддержку и масштабирование приложения, а также способствует повышению читаемости и повторному использованию компонентов интерфейса. Компоненты и их описание представлены в таблице 3.5.

Таблица 3.5 – Компоненты страниц

| Компонент | Описание |
|-----------------------|---|
| AppRout.js | Управляет навигацией на основе роли пользователя и наличия токена аутентификации |
| BookCard.js | Для отображения списка записей пользователя |
| ClientBookingsCard.js | Для отображения сотруднику карточек записей пользователей |
| DateTimePicker.js | Отображает поля для выбора даты и времени |
| EmployeeCard.js | Отображает информацию о сотруднике и предоставляет возможности выбора доступного времени для записи |
| EmployeeForm.js | Отображает форму для добавления или редактирования информации о сотруднике |

Продолжение таблицы 3.5

| | |
|-----------------|---|
| EmployeeList.js | Отображает список сотрудников в виде карточек |
| MainPost.js | Главный компонент, отображающий разметку главной страницы |
| NavMenu.js | Отображает header меню для навигации по приложению |
| Review.js | Отображает форму для добавления отзыва |
| ReviewList.js | Отображает список отзывов в виде карточек |
| ServiceForm.js | Отображает форму для создания и изменения услуг |
| ServiceList.js | Отображает список услуг с виде карточек |

Приведенная таблица 3.5 демонстрирует ключевые React-компоненты, разработанные для реализации функциональности web-приложения. Каждый компонент отвечает за определенный аспект отображения данных и взаимодействия с пользователем, способствуя модульной и поддерживаемой структуре приложения.

Маршруты, по которым будут открываться страницы из директории «pages» хранятся в App.js. Маршруты страниц представлены в листинге 3.8.

```

<div className="App">
  <Container>
    <Routes>
      <Route path="/" element={<Navigate to="/login" />} />
      <Route path="/main" element={<MainPage />} />
      <Route path="/register" element={<RegistrationForm />} />
      <Route path="/login" element={<LoginForm />} />
      <Route path="/services" element={<ServicesPage />} />
      <Route path="/serv/updserv/:id" element={<EditService />} />
      <Route path="/serv/addserv" element={<AddServiceForm />} />
      <Route path="/user" element={<UserPage />} />
      <Route path="/employees" element={<EmployeePage />} />
      <Route path="/empl/updempl/:id" element={<EditEmployee />} />
      <Route path="/empl/addempl" element={<AddEmployee />} />
      <Route path="/empl/stats" element={<ClientsBookings />} />
      <Route path="/rev/getrewempl/:id" element={<ReviewPage />} />
      <Route path="/categories" element={<CategoryPage />} />
      <Route path="/bookingform" element={isAuthenticated? <Booking-
Form /> : <LoginForm />} />{" "}
      <Route path="*" element={<NotFoundPage />} />
    </Routes>
  </Container>
</div>

```

Листинг 3.8 – Реализация маршрутизации App.js

Для проверки роли и в последующем дачи доступа или закрытия используется jwt-token, который хранится в localStorage и при необходимости извлекается оттуда. С помощью токена можно проверить аутентифицирован ли пользователь, а также проверить его роль или другие связанные с ним данные. В листинге 3.9 приведен код файла AppRouter.js, функция которого вызывается в файлах страниц для проверки доступа. Она проверяет наличие токена авторизации в localStorage и роль

пользователя. Если токен отсутствует или роль пользователя не является "ADMIN", компонент перенаправляет пользователя на страницу с ошибкой 404.

```
import React, { useEffect } from "react";
import { useNavigate } from "react-router-dom";
const AppRouter = ({ userRole }) => {
  const navigate = useNavigate();
  useEffect(() => {
    const token = localStorage.getItem("authToken");

    if (!token || (userRole && userRole !== "ADMIN")) {
      console.log("Redirecting to 404 page...");
      navigate("/404");
    }
  }, [userRole, navigate]);

  return null;
};
export default AppRouter;
```

Листинг 3.9 – Содержимое файла AppRouter.js

Если у пользователя нет доступа к определенному ресурсу, либо в url был введен некорректный путь, то пользователя перенаправляет на страницу 404, на которой находится ссылка для перехода на главную страницу. Код файла 404.js представлен в листинге 3.10.

```
import React from "react";
import { Link } from "react-router-dom";
const NotFoundPage = () => {
  return (
    <div>
      <h1>404 - Страница не найдена</h1>
      <p>Кажется, вы попали на несуществующую страницу.</p>
      <Link to="/main">Вернуться на главную</Link>
    </div>
  );
};
export default NotFoundPage;
```

Листинг 3.10 – Содержимое файла 404.js

Страница, представленная в листинге 3.10 информирует пользователя об ошибке 404 и предоставляет удобную ссылку для возврата на главную страницу приложения, улучшая таким образом пользовательский опыт при возникновении ошибок навигации.

3.9 Вывод по разделу

Разработанное web-приложение обладает четко структурированной архитектурой как на серверной, так и на клиентской стороне. Серверная часть, построенная

на Node.js с использованием Express и ORM Prisma, обеспечивает надежное управление данными. Клиентская часть, разработанная на React с применением библиотеки Material UI, предлагает интерактивный и удобный пользовательский интерфейс, разделенный на переиспользуемые компоненты и страницы, навигация между которыми осуществляется с помощью React Router. Такое разделение на модули и использование современных библиотек и фреймворков способствует масштабируемости, поддерживаемости и эффективности разработанного web-приложения.

Особое внимание уделено реализации ключевых функций для разных ролей пользователей. Для роли гостя реализованы функции регистрации и входа в систему с применением валидации, хеширования паролей и генерации JWT-токенов. Пользователи с ролью «USER» могут создавать и отменять бронирования, а также оставлять отзывы, что позволяет им активно взаимодействовать с системой. Роль «ADMIN» наделена расширенными правами, включая управление услугами, сотрудниками и расписанием, что обеспечивает эффективное администрирование.

Таким образом, серверная часть проекта построена на надёжной, гибкой и расширяемой архитектуре, обеспечивающей безопасность, удобство взаимодействия и поддержку основных бизнес-процессов приложения.

4 Тестирование web-приложения

4.1 Функциональное тестирование для гостя

Для того, чтобы выполнять какие-либо действия, пользователю необходимо авторизоваться, либо же перейти по ссылке и зарегистрироваться. Форма регистрации с проверкой на корректные данные представлена на рисунке 4.1.

The registration form is titled "Регистрация". It contains four input fields: "Имя *" with the value "katya", "Номер телефона *" with the value "+375339876756", "Email *" with the value "katya@gmail.com", and "Пароль *". Below the password field, there is a red error message "Неверно введены данные". At the bottom, there is a button labeled "ЗАРЕГИСТРИРОВАТЬСЯ" and a link "Уже есть аккаунт?".

Рисунок 4.1 – Форма регистрации

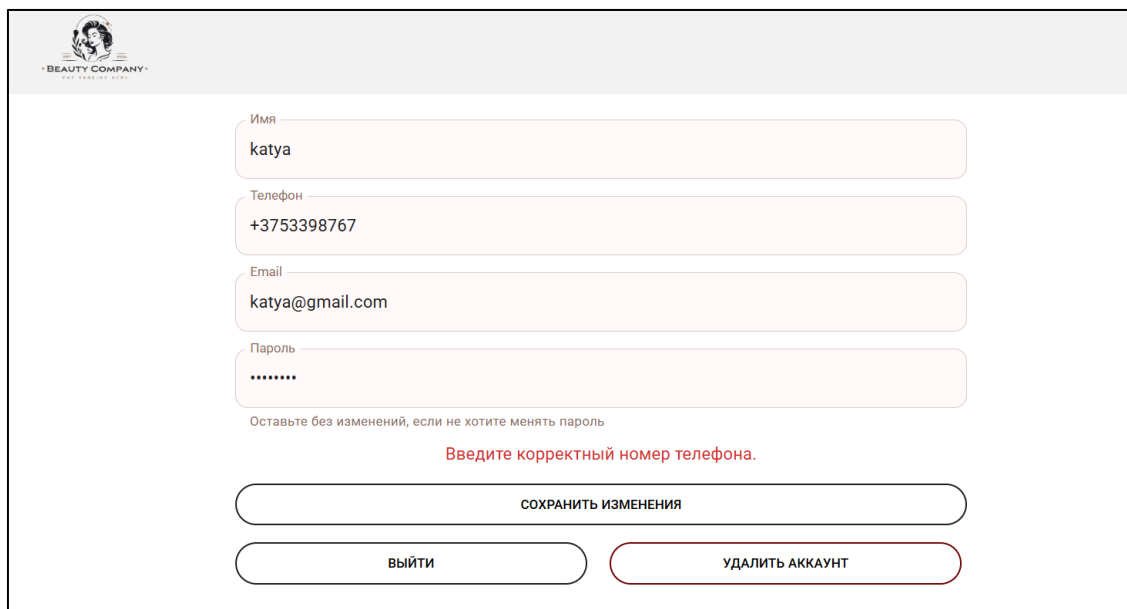
После успешной регистрации, пользователь попадает на страницу входа, для того чтобы войти в аккаунт. При вводе правильного логина и пароля пользователь перенаправляется на страницу профиля, в случае неверного ввода данных, появляются сообщения внизу поля, сигнализируя о неправильном вводе email или пароля. Интерфейс формы входа можно увидеть на рисунке 4.2.

The login form is titled "Войти". It contains two input fields: "Email *" with the value "katya@gmail.com" and "Пароль *" with masked characters "*****". Below the password field, there is a red error message "Неверный логин или пароль". At the bottom, there is a button labeled "ВОЙТИ" and a link "Нет аккаунта? Зарегистрироваться".

Рисунок 4.2 – Форма входа

4.2 Функциональное тестирование для роли «USER»

После ввода корректных данных и успешного входа в аккаунт пользователь попадает на страницу профиля, где отображаются личные данные, которые можно изменить. При вводе некорректных данных и попытке изменения аккаунта пользователь получает сообщение об ошибке. Страница профиля с некорректно введенным номером телефона представлена на рисунке 4.3.



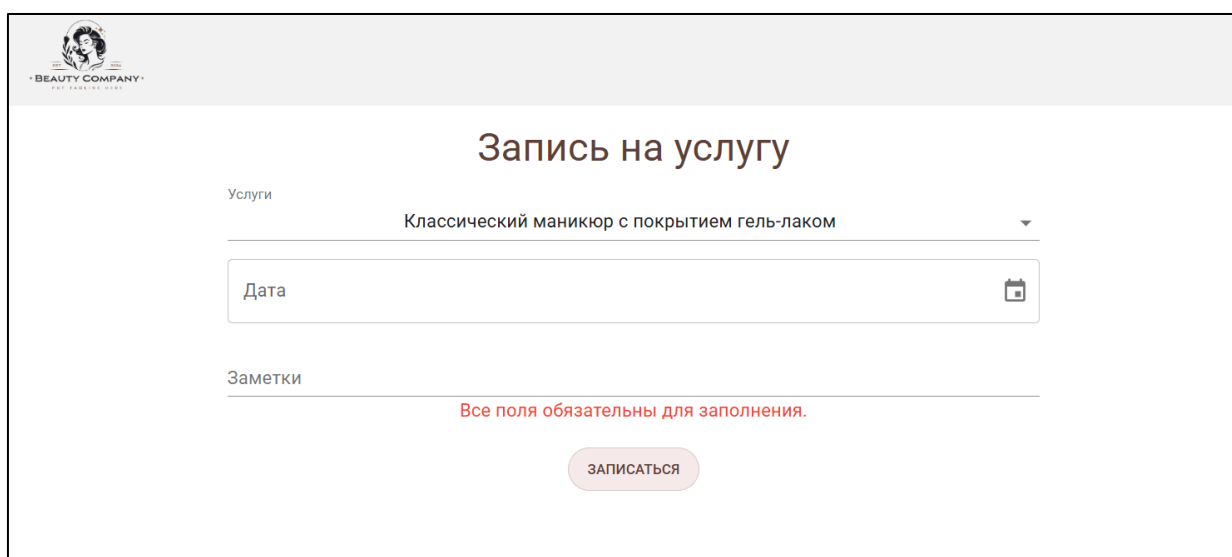
The screenshot shows the user profile page for 'BEAUTY COMPANY'. The profile data is as follows:

| Field | Value |
|---------|-----------------|
| Имя | katya |
| Телефон | +3753398767 |
| Email | katya@gmail.com |
| Пароль | ***** |

Below the fields, there is a message: "Оставьте без изменений, если не хотите менять пароль". A red error message states: "Введите корректный номер телефона." (Enter a correct phone number). At the bottom, there are three buttons: "СОХРАНИТЬ ИЗМЕНЕНИЯ" (Save changes), "ВЫЙТИ" (Logout), and "УДАЛИТЬ АККАУНТ" (Delete account).

Рисунок 4.3 – Страница профиля

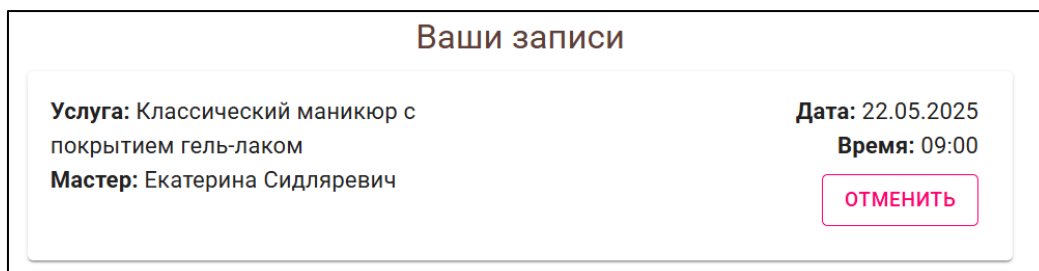
Пользователю доступен функционал записи на услугу. Для того чтобы записаться на услугу, необходимо выбрать услугу, день для записи, после чего пользователю отображаются мастера, работающие в этот день и доступные временные слоты для записи. Страница с формой записи представлена на рисунке 4.4.



The screenshot shows the "Запись на услугу" (Book a service) page. The service selected is "Классический маникюр с покрытием гель-лаком" (Classic manicure with gel polish). The date field is empty and has a calendar icon. Below the date field, there is a red error message: "Все поля обязательны для заполнения." (All fields are required for completion). At the bottom, there is a button labeled "ЗАПИСАТЬСЯ" (Book).

Рисунок 4.4 – Страница записи на услугу

При попытке записи на услугу неавторизованного пользователя перенаправляет на страницу входа. После успешного создания записи, в профиле пользователя появляется карточка с информацией о записи. Карточка с информацией о записи на услугу представлена на рисунке 4.5.

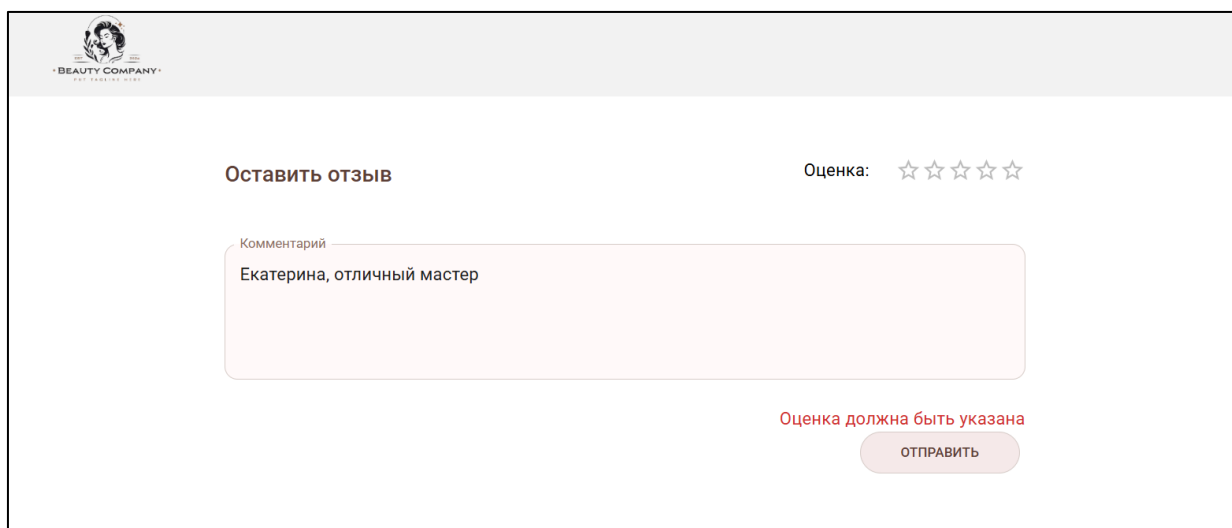


The screenshot shows a card titled "Ваши записи" (Your appointments). It contains the following information:

- Услуга:** Классический маникюр с покрытием гель-лаком
- Мастер:** Екатерина Сидляревич
- Дата:** 22.05.2025
- Время:** 09:00
- A pink button labeled "ОТМЕНИТЬ" (Cancel).

Рисунок 4.5 – Карточка записи на услугу

Пользователь может написать отзыв сотруднику салона. На рисунке 4.6 представлен пример попытки отправки отзыва с отсутствием указания оценки мастера. Чтобы отправить отзыв, необходимо заполнить все поля, после чего, отзыв отобразится в списке на странице сотрудника.



The screenshot shows the "Оставить отзыв" (Leave a review) form on the Beauty Company website. It includes:

- A header with the company logo and name.
- A title "Оставить отзыв" and a rating section "Оценка: ☆☆☆☆☆".
- A text area labeled "Комментарий" with the text "Екатерина, отличный мастер".
- A red error message: "Оценка должна быть указана" (Rating must be specified).
- A pink button labeled "ОТПРАВИТЬ" (Send).

Рисунок 4.6 – Страница с формой для отзывов

Таким образом, система отзывов реализована с обязательной проверкой заполнения всех полей, что позволяет обеспечить полноту и достоверность обратной связи. После успешной отправки отзыв становится доступным для просмотра другими пользователями на странице соответствующего сотрудника.

4.3 Функциональное тестирование для роли «ADMIN»

Администратор может создавать новые услуги. Для успешного создания услуги администратору необходимо корректно заполнить все поля. На рисунке 4.7 представлена страница создания услуги с сообщением об ошибке, в результате некорректно введенных данных.

Рисунок 4.7 – Страница создания новой услуги

После ввода некорректных данных система оперативно уведомляет администратора об ошибке, предотвращая создание неполных или неверных записей о сотрудниках. Это позволяет поддерживать актуальность и достоверность информации в системе управления салоном.

Администратор также может добавлять новых сотрудников салона красоты. Для создания нового сотрудника необходимо ввести корректные данные о сотруднике. На рисунке 4.8 представлена страница добавления сотрудника с сообщением об ошибке.

Рисунок 4.8 – Страница добавления сотрудника

Администратор также управляет расписанием сотрудников. Чтобы добавить расписание, нужно корректно заполнить дату, время начала рабочего дня и время окончания рабочего дня. На рисунке 4.9 отображается ошибка в результате попытки создания расписания, где время начала рабочего дня позже, чем время окончания рабочего дня.

Beauty Company

РАСПИСАНИЕ МАСТЕРОВ

Имя *
Анна

Фамилия *
Грунина

Email *
anna@gmail.com

Сферы услуг
Ресницы

Услуги
Классическое наращивание ресниц

Выберите день
05/22/2025

Время начала
12:00

Время окончания
08:00

Время окончания не может быть раньше времени начала.

ДОБАВИТЬ РАСПИСАНИЕ УДАЛИТЬ РАСПИСАНИЕ

Некорректное время в расписании.

Выбрать фото

Выбран файл: employee4.jpg

Рисунок 4.9 – Ошибка добавления расписания

В расписании также реализована проверка на пересечение и дублирование временных интервалов. Это позволяет исключить ситуации, когда для одного сотрудника создаются несколько смен с пересекающимся временем, обеспечивая целостность и корректность графика работы. На рисунке 4.10 отображается ошибка при попытке добавления расписания с дублирующимся расписанием.

Beauty Company

РАСПИСАНИЕ МАСТЕРОВ

Фамилия *
Грунина

Email *
anna@gmail.com

Сферы услуг
Ресницы

Услуги
Классическое наращивание ресниц

Выберите день
05/22/2025

Время начала
12:00

Время окончания
15:00

Выберите день
05/22/2025

Время начала
12:00

Время окончания
15:00

ДОБАВИТЬ РАСПИСАНИЕ УДАЛИТЬ РАСПИСАНИЕ

На дату 2025-05-22 добавлено одинаковое время расписания.

Выбрать фото

Выбран файл: employee4.jpg

Рисунок 4.10 – Ошибка добавления расписания

Администратор может просмотреть количество записей к мастеру за определенное время, указав начало окончания периода. На рисунке 4.11 отображена ошибка при попытке вывести статистику по количеству записей к мастерам с указанием некорректной даты.

Статистика записей по мастерам

Начальная дата
05/22/2025

Конечная дата
05/21/2025

Начальное время должно быть раньше конечного.

ПОКАЗАТЬ СТАТИСТИКУ

Рисунок 4.11 – Ошибка вывода статистики

Ошибки при попытке вывода статистики с некорректными датами позволяют избежать получения недостоверных данных и помогают администратору своевременно скорректировать параметры запроса. Такая проверка улучшает точность отчетности и повышает надежность работы административного функционала.

4.4 Вывод по разделу

Проведённое функциональное тестирование web-приложения показало, что основные сценарии взаимодействия пользователей с системой реализованы корректно и соответствуют требованиям, предъявляемым к современным информационным системам для салонов красоты. В процессе тестирования были охвачены ключевые роли пользователей: гость, авторизованный пользователь и администратор, а также подробно протестированы их возможности и ограничения при работе с интерфейсом приложения.

Для роли гостя было подтверждено, что система надёжно ограничивает доступ к функционалу, требующему авторизации. Гость может зарегистрироваться, и в процессе регистрации реализована проверка на корректность введённых данных. Система корректно реагирует на ошибочные данные, не позволяя сохранять недопустимую информацию.

Система стабильно обрабатывает как корректные, так и ошибочные действия пользователя, своевременно информируя его об этом через удобные и понятные интерфейсные элементы. Общая архитектура приложения позволяет обеспечить высокую надёжность, удобство использования и безопасность, что особенно важно для сервисов, работающих с личными данными и расписаниями.

5 Руководство пользователя

5.1 Руководство для гостя

При открытии web-приложения пользователь попадает на главную страницу, вид которой приведен на рисунке 5.1. На ней можно перейти по ссылкам «услуги» и «мастера», чтобы просмотреть список категорий услуг и мастеров, соответственно.

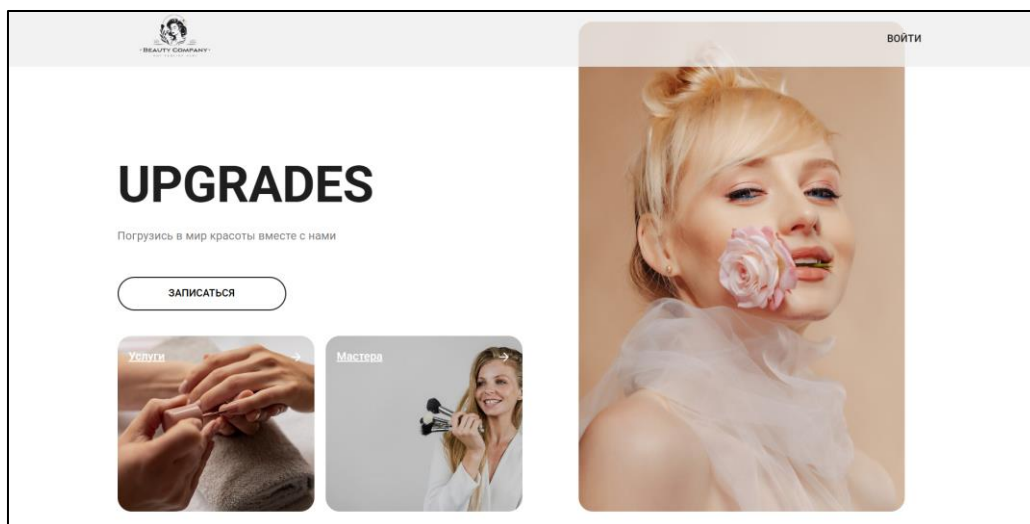


Рисунок 5.1 – Главная страница web-приложения

Страница со списком категорий услуг представлена на рисунке 5.2.

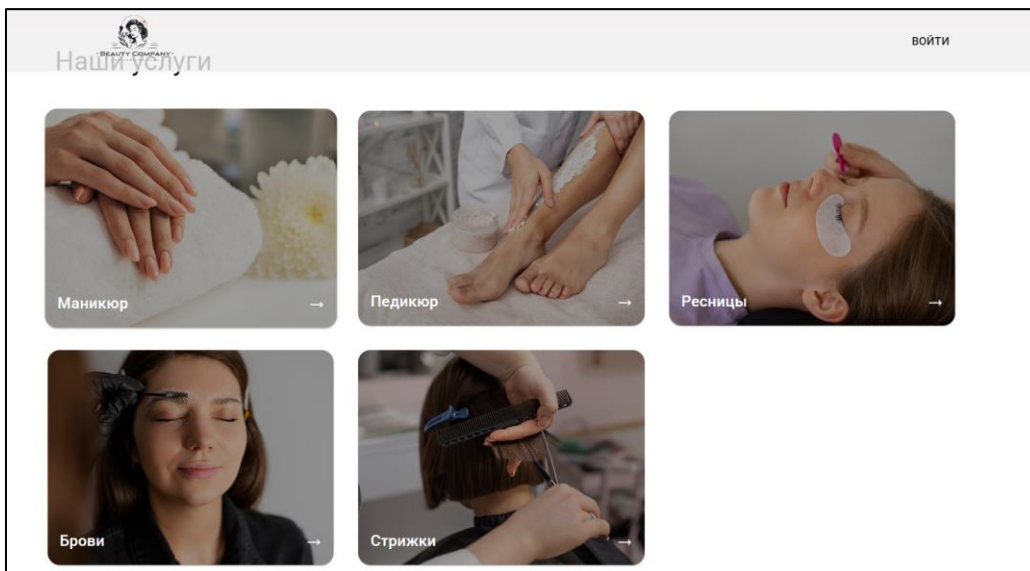


Рисунок 5.2 – Страница категорий услуг

Перейдя на конкретную категорию, можно просмотреть список услуг в этой категории. Страница со списком услуг в выбранной категории представлена на рисунке 5.3.

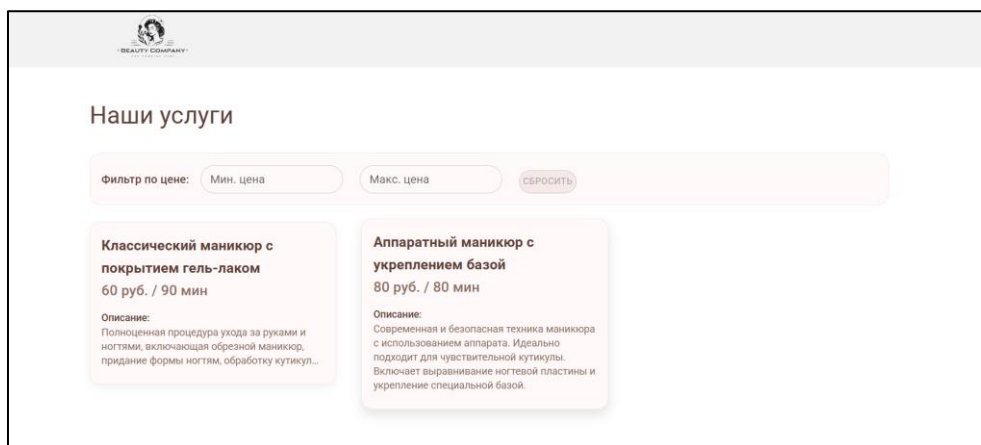


Рисунок 5.3 – Страница категорий услуг

Пользователь может сделать сортировку услуг по цене, заполнив поля с минимальной и максимальной ценой.

Страница со списком мастеров представлена на рисунке 5.4.

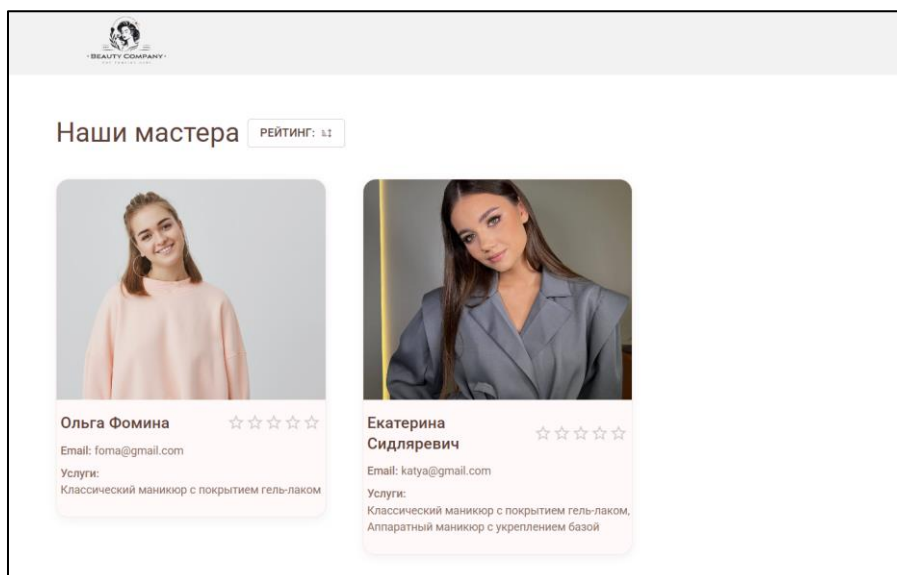


Рисунок 5.4 – Страница со списком мастеров

Перейдя на конкретного мастера, пользователь может просмотреть отзывы, написанные мастеру. Страница с отзывами представлена на рисунке 5.5.

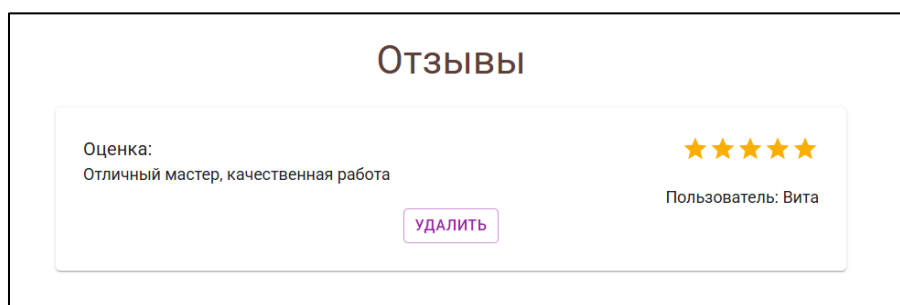


Рисунок 5.5 – Страница с отзывами

Так же на главной странице, представленной на рисунке 5.1 пользователю доступна кнопка «записаться», по нажатию на которую пользователь попадает на страницу входа, представленную на рисунке 5.6.

The screenshot shows a login form titled "Войти" (Login). It contains two input fields: "Email *" and "Пароль *" (Password *). Below the password field is a button labeled "войти" (login). At the bottom of the form, there is a link that says "Нет аккаунта? Зарегистрироваться" (No account? Register).

Рисунок 5.6 – Страница входа

Если аккаунта не существует, можно перейти по ссылке на страницу регистрации, представленную на рисунке 5.7.

The screenshot shows a registration form titled "Регистрация" (Registration). It contains four input fields: "Имя *" (Name *), "Номер телефона *" (Phone number *), "Email *" and "Пароль *" (Password *). Below the password field is a button labeled "ЗАРЕГИСТРИРОВАТЬСЯ" (REGISTER). At the bottom of the form, there is a link that says "Уже есть аккаунт?" (Already have an account?).

Рисунок 5.7 – Страница регистрации

Войдя в аккаунт гость становится пользователем с ролью «USER».

5.2 Руководство для роли «USER»

При входе в аккаунт пользователь попадает на страницу с личной информацией, представленной на рисунке 5.8.

The screenshot shows a user profile page for 'BEAUTY COMPANY'. At the top left is the company logo. The main content area contains several input fields for personal information: 'Имя' (Name) with the value 'Вита', 'Телефон' (Phone) with '+375234567333', 'Email' with 'vita@gmail.com', and 'Пароль' (Password) with masked characters '*****'. Below these fields is a small text note: 'Оставьте без изменений, если не хотите менять пароль'. There are three buttons: a wide 'СОХРАНИТЬ ИЗМЕНЕНИЯ' (Save changes) button, and two smaller buttons, 'ВЫЙТИ' (Logout) and 'УДАЛИТЬ АККАУНТ' (Delete account). At the bottom, the text 'Ваши записи' (Your appointments) is followed by 'На данный момент записей нет' (No appointments at the moment).

Рисунок 5.8 – Страница с личной информацией

Пользователь может оформить запись на услугу, нажав на кнопку «записаться» на главной странице, представленной на рисунке 5.1. По нажатию пользователя перенаправляет на страницу записи, представленную на рисунке 5.9.

The screenshot shows a page titled 'Запись на услугу' (Appointment for service). It features a dropdown menu labeled 'Услуги' (Services). Below it is a date selection field labeled 'Дата' with a calendar icon. Underneath the date field is a text area labeled 'Заметки' (Notes). At the bottom center of the form is a button labeled 'ЗАПИСАТЬСЯ' (Book).

Рисунок 5.9 – Страница записи на услугу

На странице записи пользователь может выбрать услугу, день для записи, после чего отобразится список мастеров и доступных временных слотов для записи на услугу. Заполнив форму правильными данными и нажав на кнопку записаться, у пользователя появляется запись на странице с личной информацией, представленная на рисунке 5.10.

The screenshot shows a page titled 'Ваши записи' (Your appointments). It displays a single appointment card. The card contains the following information: 'Услуга: Каскад' (Service: Cascade) and 'Мастер: Виталий Сафонов' (Master: Vitaliy Safonov) on the left; 'Дата: 19.05.2025' (Date: 19.05.2025) and 'Время: 12:00' (Time: 12:00) on the right. At the bottom right of the card is a button labeled 'ОТМЕНИТЬ' (Cancel).

Рисунок 5.10 – Запись на услугу

Пользователь может отменить созданную запись. По истечению времени, запись удаляется со страницы личной информации пользователя.

Пользователь также может оставить отзыв конкретному мастеру. Страница с формой для написания отзыва представлена на рисунке 5.11.

The screenshot shows a web interface for leaving a review. At the top left is the 'BEAUTY COMPANY' logo. The main heading is 'Оставить отзыв' (Leave a review). To the right, there is a rating section labeled 'Оценка:' followed by five empty star icons. Below this is a large text input field labeled 'Комментарий'. To the right of the input field is a button labeled 'ОТПРАВИТЬ' (SEND). Below the input field is a section titled 'Отзывы' (Reviews). It displays a sample review with the text 'Оценка: Отличный мастер, качественная работа' (Rating: Excellent master, quality work) and five yellow stars. To the right of the stars is the text 'Пользователь: Вита' (User: Vita). Below the review text is a button labeled 'УДАЛИТЬ' (DELETE).

Рисунок 5.11 – Страница с формой для отзывов

Пользователь может удалить написанный им отзыв. Для роли «USER» также доступны все функции роли «гость».

5.3 Руководство для роли «ADMIN»

Для роли «ADMIN» доступен весь функционал роли «USER» и непосредственно функционал администратора.

Администратор может добавить новую услугу. Страница добавления услуги представлена на рисунке 5.12.

The screenshot shows a web interface for adding a new service. At the top left is the 'BEAUTY COMPANY' logo. At the top right is the text 'РАСПИСАНИЕ МАСТЕРОВ' (Master's schedule). The main heading is 'Добавить новую услугу' (Add new service). Below this are several input fields: 'Название услуги' (Service name), 'Описание услуги' (Service description), 'Цена услуги' (Service price) with the value '0', 'Сфера услуг' (Service sphere) with a dropdown arrow, and 'Длительность (в минутах)' (Duration (in minutes)) with the value '60'. At the bottom right is a button labeled 'ДОБАВИТЬ УСЛУГУ' (ADD SERVICE).

Рисунок 5.12 – Страница добавления услуги

Администратор также может добавить нового сотрудника и изменить информацию о нем. Страница изменения сотрудника представлена на рисунке 5.13.

Рисунок 5.13 – Страница добавления сотрудника

Для добавления нового сотрудника необходимо заполнить корректно все поля, а также выбрать фотографию сотрудника.

Администратору также доступна страница для просмотра расписания сотрудников по выбранной дате, представленная на рисунке 5.14.

Рисунок 5.14 – Страница расписания сотрудников

Администратору также доступна страница со статистикой сотрудников. Администратор может просмотреть количество записей к каждому сотруднику. Страница со статистикой представлена на рисунке 5.15.

Рисунок 5.15 – Страница со статистикой

Администратор может выбрать диапазон дат, указав дату начала и дату окончания, за которую ему необходимо просмотреть статистику. По умолчанию дата начала устанавливается на первый день текущего месяца, а дата окончания на текущий день.

5.4 Вывод по разделу

В данном разделе подробно описан функционал веб-приложения, доступный различным ролям пользователей — гостю, пользователю с ролью «USER» и администратору с ролью «ADMIN». Представлены основные страницы приложения и их назначение: главная страница, страницы категорий услуг и мастеров, страницы с отзывами, а также страницы регистрации и входа.

Для роли «USER» описаны возможности оформления и управления записями на услуги, просмотра личной информации, а также написания и удаления отзывов. Пользователю с ролью «USER» доступны все функции гостя.

Роль «ADMIN» включает полный набор функций пользователя, а также расширенный функционал администратора: добавление и редактирование услуг и сотрудников, просмотр расписания и статистики по сотрудникам.

Таким образом, приложение предоставляет четко структурированный и интуитивно понятный интерфейс, учитывающий разные уровни доступа и потребности пользователей.

Заключение

Приложение салон красоты «Upgrade» представляет собой полнофункциональный сервис для организации работы салона красоты. Оно обеспечивает удобный просмотр и выбор интересующих услуг, осуществление записи на подходящую дату и время. Кроме того, пользователи могут просматривать и оставлять отзывы о мастерах, а также ставить оценки.

Администраторы имеют доступ к управлению списком услуг, добавлению новых и изменению существующих, могут управлять сотрудниками, их расписанием, а также отправлять уведомления клиентам по электронной почте.

Для серверной части приложения используется Node.js, в частности фреймворк Express, обеспечивающий высокую производительность и асинхронную обработку запросов. Структура приложения включает middleware, routers, controllers и модель базы данных, обеспечивая удобную абстракцию для работы с данными.

Клиентская часть разработана с использованием библиотеки React, что обеспечивает эффективную и удобную навигацию по страницам приложения. Для облегчения верстки использован фреймворк Material-UI, который предоставляет множество компонентов для стилизации сайта. Axios используется для отправки запросов на сервер.

Для хранения данных о клиентах, услугах, сотрудниках и бронированиях используется СУБД PostgreSQL с ORM Prisma, обеспечивая быстрый и легкий способ доступа к данным.

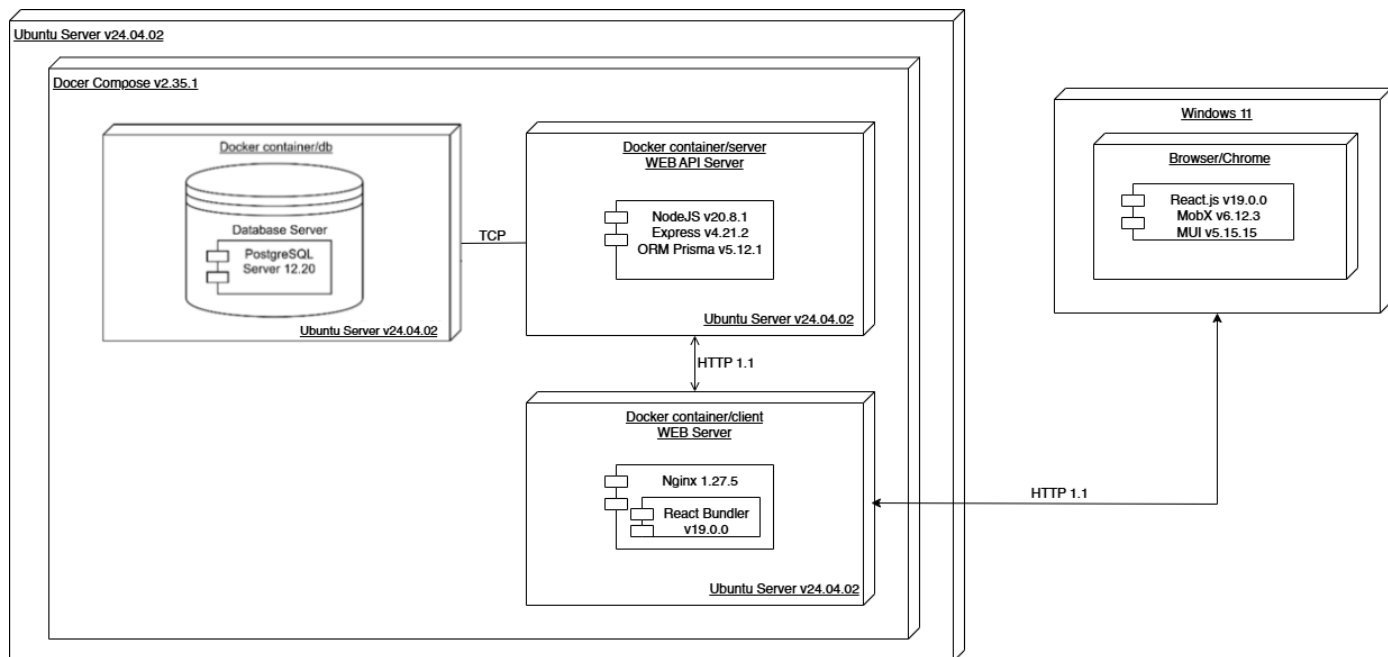
После проведения тестирования сделан вывод о том, что приложение работает корректно и осуществляется проверка на возможные ошибки, для предотвращения нарушения функциональности. Как клиентская, так и серверная части проекта имеют хороший потенциал для будущих модификаций, и на данном этапе программное средство готово к использованию в сети Интернет.

Список используемых источников

- 1 HqBeauty [Электронный ресурс] / Режим доступа: <https://hqbeauty.by/> – Дата доступа: 5.03.2025.
- 2 Изуми [Электронный ресурс] / Режим доступа: <https://изуми.бел> – Дата доступа: 5.03.2025.
- 3 Сафина [Электронный ресурс] / Режим доступа: <https://safina.by/> – Дата доступа: 5.03.2025.
- 4 Express [Электронный ресурс] / Режим доступа: <https://expressjs.com/> – Дата доступа: 10.03.2025.
- 5 Prisma [Электронный ресурс] / Режим доступа: <https://www.prisma.io/> – Дата доступа: 10.03.2025.
- 6 Jsonwebtoken [Электронный ресурс] / Режим доступа: <https://jwt.io/> – Дата доступа: 10.03.2025.
- 7 Cors [Электронный ресурс] / Режим доступа: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/CORS> – Дата доступа: 10.03.2025.
- 8 Bcryptjs [Электронный ресурс] / Режим доступа: <https://www.npmjs.com/package/bcryptjs> – Дата доступа: 21.03.2025.
- 9 Cookie-parser [Электронный ресурс] / Режим доступа: <https://www.npmjs.com/package/cookie-parser> – Дата доступа: 21.03.2025.
- 10 Dayjs [Электронный ресурс] / Режим доступа: <https://day.js.org/> – Дата доступа: 10.03.2025.
- 11 Date-fns [Электронный ресурс] / Режим доступа: <https://date-fns.org/> – Дата доступа: 10.03.2025.
- 12 Date-fns-tz [Электронный ресурс] / Режим доступа: <https://date-fns.org/> – Дата доступа: 10.03.2025.
- 13 React [Электронный ресурс] / Режим доступа: <https://react.dev/> – Дата доступа: 10.03.2025.
- 14 React-dom [Электронный ресурс] / Режим доступа: <https://www.npmjs.com/package/react-dom> – Дата доступа: 10.03.2025.
- 15 React-router-dom [Электронный ресурс] / Режим доступа: <https://reactrouter.com/> – Дата доступа: 10.03.2025.
- 16 Axios [Электронный ресурс] / Режим доступа: <https://www.axios.com/> – Дата доступа: 21.03.2025.
- 17 Mui Material [Электронный ресурс] / Режим доступа: <https://mui.com/> – Дата доступа: 08.04.2025.
- 18 Mobx [Электронный ресурс] / Режим доступа: <https://mobx.js.org/> – Дата доступа: 10.03.2025.
- 19 Mobx-react-lite [Электронный ресурс] / Режим доступа: <https://www.npmjs.com/package/mobx-react-lite> – Дата доступа: 10.03.2025.
- 20 NodeJS [Электронный ресурс] / Режим доступа: <https://nodejs.org/en> – Дата доступа: 10.03.2025.
- 21 PostgreSQL [Электронный ресурс] / Режим доступа: <https://www.postgresql.org/> – Дата доступа: 10.03.2025.

Приложение А

Схема архитектуры приложения



Приложение Б

```

generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

model Users {
  userID      Int           @id @default(autoincrement())
  name        String
  phone       String?
  email       String        @unique
  password    String
  role        Roles         @default(USER)
  reviews     Reviews[]
  registrations Booking[] // Переименовано на Booking, чтобы отражать
  суть
}

enum Roles {
  ADMIN
  USER
}

model ServiceCategory {
  categoryID Int           @id @default(autoincrement())
  name        String        @unique
  services     Services[]
  employees    EmployeesServiceCategories[]
}

model Services {
  serviceID   Int           @id @default(autoincrement())
  name        String        @unique
  description  String?
  price       Decimal        @default(0.00)
  duration    Int           @default(60) // Добавлено поле
  duration в минутах
  categoryID  Int?
  category    ServiceCategory? @relation(fields: [categoryID],
references: [categoryID], onDelete: SetNull, onUpdate: Cascade)
  employees   EmployeesServices[]
  bookings    Booking[]       // Добавлена связь с таблицей
  Booking
}

model Employees {
  employeeID   Int           @id @default(autoincrement())
  name         String
  surname      String

```

```

    email          String          @unique
    services        EmployeesServices[]
    serviceCategories EmployeesServiceCategories[]
    reviews         Reviews[]
    bookings         Booking[]           // Переименовано на
Booking
    photo           Bytes?
    availability     Availability[]       // Добавлена связь с
таблицей Availability
}

model EmployeesServices {
    employeeID Int
    serviceID  Int

    employee Employees @relation(fields: [employeeID], references: [em-
mployeeID], onDelete: Cascade)
    service  Services  @relation(fields: [serviceID], references: [ser-
viceID], onDelete: Cascade)

    @@id([employeeID, serviceID])
}

model EmployeesServiceCategories {
    employeeID          Int
    serviceCategoryID  Int

    employee            Employees @relation(fields: [employeeID],
references: [employeeID], onDelete: Cascade)
    serviceCategory    ServiceCategory @relation(fields: [serviceCatego-
ryID], references: [categoryID], onDelete: Cascade)

    @@id([employeeID, serviceCategoryID])
}

model Booking { // Переименовано из Registration и обновлена структура
    bookingID    Int          @id @default(autoincrement())
    userID       Int
    employeeID   Int
    serviceID    Int
    startTime    DateTime     @db.Timestamptz
    endTime      DateTime     @db.Timestamptz
    notes        String?

    user         Users        @relation(fields: [userID], references: [userID],
onDelete: Cascade)
    employee     Employees @relation(fields: [employeeID], references: [em-
mployeeID], onDelete: Cascade)
    service      Services  @relation(fields: [serviceID], references: [ser-
viceID], onDelete: Cascade)
}

model Reviews {
    reviewID    Int          @id @default(autoincrement())

```

```

    userID      Int
    employeeID  Int
    rating      Int
    comm        String?

    user Users @relation(fields: [userID], references: [userID],
onDelete: Cascade)
    employee Employees @relation(fields: [employeeID], references: [em-
ployeeID], onDelete: Cascade)
}

// Новая модель для хранения интервалов доступности сотрудников
model Availability {
    availabilityID Int @id @default(autoincrement())
    employeeID Int
    date DateTime @db.Timestamptz
    startTime DateTime @db.Timestamptz
    endTime DateTime @db.Timestamptz

    employee Employees @relation(fields: [employeeID], references: [em-
ployeeID], onDelete: Cascade)
}

```

Листинг 1 – Schema Prisma для создания базы данных

Приложение В

```

async addBooking(req, res) {
  try {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({ message: "Неверно введены данные валидации", errors: errors.array() });
    }

    const { userID, employeeID, serviceID, date, startTime: start-
TimeStr, notes } = req.body;

    if (!userID || !employeeID || !serviceID || !date || !start-
TimeStr) {
      return res.status(400).json({ message: "Все обязательные поля (userID, employeeID, serviceID, date, startTime) должны быть запол-
нены" });
    }

    const serviceIdInt = parseInt(serviceID, 10);
    const employeeIdInt = parseInt(employeeID, 10);
    const userIdInt = parseInt(userID, 10);

    if (isNaN(serviceIdInt) || isNaN(employeeIdInt) || is-
NaN(userIdInt)) {
      return res.status(400).json({ message: "ID пользователя,
мастера или услуги некорректны." });
    }

    const service = await clientPr.services.findUnique({
      where: { serviceID: serviceIdInt },
      select: { duration: true },
    });

    if (!service) {
      return res.status(404).json({ message: "Услуга не найдена."
});
    }
    const serviceDuration = service.duration;

    const [hours, minutes] = startTimeStr.split(':').map(Number);
    if (isNaN(hours) || isNaN(minutes) || hours < 0 || hours > 23
|| minutes < 0 || minutes > 59) {
      return res.status(400).json({ message: "Некорректный формат
времени начала." });
    }

    const bookingStartDateTimeLocal = dayjs(Number(date))
      .tz('Europe/Moscow')
      .hour(hours)
      .minute(minutes)
      .second(0)
      .millisecond(0);
  }
}

```

```

        const bookingStartDateTimeUtc = bookingStartDateTimeLocal.utc().toDate();
        const bookingEndDateTimeUtc = bookingStartDateTimeLocal.add(serviceDuration, 'minute').utc().toDate();

        // Финальная проверка доступности слота
        const bookingDayQueryStartUtc = dayjs(bookingStartDateTimeUtc).startOf('day').toDate();
        const bookingDayQueryEndUtc = dayjs(bookingStartDateTimeUtc).endOf('day').toDate();

        const employeeAvailability = await clientPr.availability.findFirst({
            where: {
                employeeID: employeeIdInt,
                date: { // Поле 'date' в Availability должно быть началом дня UTC для мастера
                    gte: bookingDayQueryStartUtc,
                    lte: bookingDayQueryEndUtc,
                },
                startTime: { lte: bookingStartDateTimeUtc }, // Рабочее время начинается до или в момент начала брони
                endTime: { gte: bookingEndDateTimeUtc }, // Рабочее время заканчивается после или в момент конца брони
            },
        });

        if (!employeeAvailability) {
            return res.status(400).json({ message: "Выбранное время выходит за рамки рабочего времени мастера или мастер недоступен в этот день." });
        }

        const overlappingBooking = await clientPr.booking.findFirst({
            where: {
                employeeID: employeeIdInt,
                NOT: {
                    // Если бы был режим редактирования, здесь можно было бы исключить текущую бронь
                    // bookingID: isEditing ? currentBookingId : undefined
                },
                // (SlotStart < BookingEnd) and (SlotEnd > BookingStart)
                startTime: { lt: bookingEndDateTimeUtc },
                endTime: { gt: bookingStartDateTimeUtc },
            },
        });

        if (overlappingBooking) {
            return res.status(409).json({ message: "Выбранное время уже занято. Пожалуйста, выберите другое время." });
        }

        const newBooking = await clientPr.booking.create({

```

```

        data: {
            userID: userIdInt,
            employeeID: employeeIdInt,
            serviceID: serviceIdInt,
            startTime: bookingStartDateTimeUtc,
            endTime: bookingEndDateTimeUtc,
            notes: notes || null,
        },
    });

    res.status(201).json({
        message: "Запись успешно добавлена",
        booking: newBooking,
    });
    console.log("Запись успешно добавлена:", newBooking);

} catch (error) {
    console.error("Ошибка при добавлении записи:", error);
    if (error.code === 'P2003') { // Foreign key constraint failed
        if (error.meta?.field_name?.includes('userID')) {
            return res.status(400).json({ message: "Пользователь не найден." });
        }
        if (error.meta?.field_name?.includes('employeeID')) {
            return res.status(400).json({ message: "Мастер не найден." });
        }
        if (error.meta?.field_name?.includes('serviceID')) {
            return res.status(400).json({ message: "Услуга не найдена." });
        }
    }
    res.status(500).json({ message: "Ошибка на сервере при добавлении записи: " + error.message });
}
}

```

Листинг 2 – Функция addBooking

Приложение Г

```

async addEmployee(req, res) {
  upload.single('photo')(req, res, async (err) => {
    if (err) {
      console.error("Ошибка загрузки файла:", err);
      return res.status(500).json({ message: "Ошибка загрузки файла"
});
    }

    try {
      const { name, surname, email, serviceCategories, services,
availabilities } = req.body;
      const photo = req.file ? req.file.buffer : null;

      if (!name || !surname || !email || !serviceCategories ||
!services) {
        return res.status(400).json({
          message: "Пожалуйста, заполните все обязательные поля
(имя, фамилия, email, категории услуг и услуги)."
        });
      }

      const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
      if (!emailRegex.test(email)) {
        return res.status(400).json({ message: "Некорректный email"
});
      }

      const existingEmployee = await clientPr.employees.findFirst({
        where: { email }
      });

      if (existingEmployee) {
        return res.status(409).json({ message: "Сотрудник с таким
email уже существует" });
      }

      const newEmployee = await clientPr.employees.create({
        data: {
          name,
          surname,
          email,
          photo,
          serviceCategories: {
            create: JSON.parse(serviceCategories).map((categoryID)
=> ({
              serviceCategoryID: categoryID,
            })),
          },
          services: {
            create: JSON.parse(services)
              .map(Number)
              .filter(Number.isInteger)

```

```

        .map((serviceID) => ({ serviceID })),
    },
  },
  include: {
    serviceCategories: true,
    services: true,
  },
});

// Обработка availabilities
if (availabilities) {
  let parsedAvailabilities;
  try {
    parsedAvailabilities = JSON.parse(availabilities);
  } catch (parseError) {
    return res.status(400).json({ message: "Некорректный
формат availabilities" });
  }

  if (Array.isArray(parsedAvailabilities) && parsedAvailabil-
ities.length > 0) {
    const validSlots = parsedAvailabilities.filter(
      (slot) => slot.date && slot.startTime && slot.endTime
    );

    if (validSlots.length === 0) {
      // Просто пропускаем – расписание не задано
      console.log("Расписание не указано – добавим позже при
необходимости.");
    } else {
      const today = new Date();
      today.setHours(0, 0, 0, 0);

      const availabilitySlots = validSlots.map((slot) => {
        const { date, startTime, endTime } = slot;

        const dateObj = new Date(date);
        const startDateTime = new Date(startTime);
        const endDateTime = new Date(endTime);

        if (isNaN(startDateTime.getTime()) || is-
NaN(endDateTime.getTime()) || isNaN(dateObj.getTime())) {
          throw new Error("Некорректный формат даты или
времени в расписании.");
        }

        if (dateObj < today) {
          throw new Error("Нельзя установить интервал доступ-
ности на прошедшую дату.");
        }

        if (startDateTime >= endDateTime) {
          throw new Error("Время начала должно быть раньше
времени окончания.");
        }
      });
    }
  }
}

```



```

        }

        return {
            employeeID: newEmployee.employeeID,
            date: dateObj,
            startTime: startDateTime,
            endTime: endDateTime,
        };
    });

    await clientPr.availability.createMany({ data: availabilitySlots });
    }
}

res.status(201).json({
    message: "Сотрудник успешно добавлен",
    employee: newEmployee,
});

} catch (error) {
    console.error("Ошибка добавления сотрудника:", error);
    if (error.message) {
        return res.status(400).json({ message: error.message });
    }
    return res.status(500).json({ message: "Ошибка добавления сотрудника" });
}
});
}

```

Листинг 3 – Функция addEmployee