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

ДОПУСТИТЬ К ЗАЩИТЕ

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

_____ Л.Б. Соколинский

«___»_____ 2024 г.

# Разработка веб-приложения для записи на бьюти-услуги (стартап как диплом)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.03.04.2024.308-344.ВКР

Консультант,
индивидуальный предприниматель
_____ К.С. Ксендзова
«___»_____ 2024 г.

Консультант,
доцент кафедры ЭиФ, к.э.н.
_____ Л.Ш. Морозова
«___»_____ 2024 г.

Научный руководитель,
доцент кафедры СП, к.э.н.
_____ Т.Ю. Шабанов

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

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

Челябинск, 2024 г.

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

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

**ЗАДАНИЕ**
**на выполнение выпускной квалификационной работы бакалавра**
студенту группы КЭ-403
Левшину Артёму Вячеславовичу,
обучающемуся по направлению
09.03.04 «Программная инженерия»

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12)
Разработка веб-приложения для записи на бьюти-услуги (стартап как диплом).
2. **Срок сдачи студентом законченной работы:** 03.06.2024 г.
3. **Исходные данные к работе**

3.1. Документация к программной платформе Node.js. [Электронный ресурс] URL: https://nodejs.org/docs/latest/api/ (дата обращения: 01.02.2024 г.).
3.2. Руководство по языку JavaScript. [Электронный ресурс] URL: https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/ (дата обращения: 01.02.2024 г.).
3.3. Документация Telegram по разработке мини-приложений, информация для разработчиков. [Электронный ресурс] URL: https://core.telegram.org/bots/webapps (дата обращения: 11.02.2024 г.).
3.4. Документация к JavaScript библитеке React. [Электронный ресурс] URL: https://reactnative.dev/docs/ (дата обращения: 09.02.2024 г.).

4. **Перечень подлежащих разработке вопросов**

4.1. Провести обзор аналогичных платформ и предметной области.
4.2. Спроектировать веб-приложение.

4.3. Создать архитектуру проекта.

4.4. Реализовать веб-приложение и протестировать его.

4.5. Спроектировать бизнес-модель проекта.

**5. Дата выдачи задания:** 29.01.2024 г.

**Научный руководитель,**
доцент кафедры СП, к.э.н. Т.Ю. Шабанов

**Задание принял к исполнению** А.В. Левшин

**ОГЛАВЛЕНИЕ**

# ВВЕДЕНИЕ

## Актуальность

Сервис онлайн-записи для различных организаций и типов бизнеса, в том числе для салонов красоты – это удобный инструмент, позволяющий его пользователям организовывать, планировать и записываться на встречи, мероприятия или услуги через интернет. Такой тип сервиса является эффективным и комфортным способом общения как для руководителей и персонала, так и для клиентов.

Благодаря такому сервису решается проблема фиксирования расписания клиентов и их особенностей. У представителей услуг появляется точное расписание записей клиентов. Клиент не будет ждать долго ответа от мастера, так как через календарь можно записаться на услугу напрямую.

Сервисы онлайн-записи предоставляют компаниям и организациям ряд преимуществ. Они позволяют экономить как время пользователей, так и тех, кто предоставляет соответствующие услуги. Возможность бронирования через интернет привлекает новых клиентов и помогает снять нагрузку с работников. Также исключается человеческий фактор, который мог бы привести к ошибкам записи, так как весь процесс становится автоматизированным. Клиенты могут использовать онлайн-сервис в любое удобное время суток и получать напоминания о предстоящих событиях. Поэтому и растет востребованность подобных сервисов.

Более того, такое программное обеспечение предоставляет администраторам салонов инструменты по удаленному управлению процессом работой салона и наглядное представление клиентского потока.

Сервис онлайн-записи – платформа, на которой объединяются многие представители услуг, это позволяет клиенту подобрать наиболее подходящего специалиста, а для бизнесов, еще наращивающих свою клиентскую базу, такие решения сокращают затраты на разработку собственного сайта и устраняют необходимость набора дополнительного персонала для работы с заказчиками.

Распространение в последние годы также получили мини приложения. Они предоставляют доступ к веб-приложению из мессенджеров без необходимости перехода в другие приложения или браузер. При этом взаимодействуя с API мессенджера, например, через Bot API в Телеграм, мини-приложение позволяет упросить процесс аутентификации пользователя, высылать уведомления прямо в боте, за которым закреплено это самое приложение.

Использование мини -приложений в Телеграм, помимо прочего, удобнее еще и в силу того, что многие потенциальные клиенты уже знакомы с интерфейсом самого мессенджера, поэтому им гораздо легче разобраться в принципе работы сервиса и совершить первую запись.

PWA – Progressive Web App, предоставляет возможность создания веб-приложений, которые во многом перенимают функционал нативных скачиваемых приложений, например, пользователь сможет иметь доступ к некоторым своим данным из ярлыка веб-приложения закрепленного на рабочем экране своего устройства.

Более того, применение данного подхода в разработке приложения позволяет сделать его независимым от платформы пользователя, предоставить некоторые функции нативных, специфичных для операционной системы приложений без необходимости их скачивать и занимать память устройства.

**Постановка задачи**

Целью выпускной квалификационной работы является разработка веб-приложения для записи на бьюти-услуги в формате стартап как диплом. Для достижения поставленной цели необходимо решить следующие задачи:

1) провести обзор аналогичных платформ и предметной области;
2) спроектировать веб-приложение;
3) спроектировать архитектуру веб-приложения;
4) реализовать веб-приложение и протестировать его;
5) произвести бизнес-проектирование проекта.

**Структура и содержание работы**

Работа состоит из введения, четырех глав, заключения и списка литературы. Объем работы составляет 63 страницы, объем списка литературы – 28 источников.

В первой главе «Анализ предметной области» рассматривается разбор существующих аналогов и предметной области.

Вторая глава «Анализ требований к программной системе» содержит функциональные и нефункциональные требования, диаграмму вариантов использования и спецификацию основных вариантов использования.

Третья глава «Архитектура системы» содержит общее описание архитектуры системы, описание компонентов и описание принципов взаимодействия элементов системы.

Четвертая глава «Реализация и тестирование» включает описание создания веб-приложения и результаты тестирования.

Пятая глава посвящена бизнес-проектированию. В ней был произведен стратегический и экономический анализ проекта «PlanzUp» в рамках «Стартап как диплом».

В приложении А содержатся исходные коды работы.

В приложении Б содержатся таблицы расчетов экономических показателей по проекту.

В приложении В приведена схема базы данных.

# 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## 1.1. Описание предметной области

Целью данной работы является разработка веб-сервиса для записи в салон красоты. Сервис создается для осуществления удобного и понятного взаимодействия с пользователем при записи на оказание услуг и должно содержать следующие функции:

1) создание записей на определенную дату, время и определенный тип услуг;
2) просмотр клиентом его актуальных записей;
3) отправка напоминаний о записи;
4) получение подтверждения записи от представителя;
5) отмена клиентом его записей;
6) настройка представителем услуг расписания работы для указания свободных слотов.

**Анализ используемых технологий**

Доступность программного продукта в качестве веб-приложения обеспечивает пользователям доступ к нему с большинства современных платформ, при этом, если делать его адаптивным, то предоставляемым сервисом можно пользоваться с мобильных устройств. Это сокращает затраты на разработку и повышает удобство решения.

Мессенджер Телеграм предоставляет возможность для реализации данных сервисов на основе чат-ботов – программ, разрабатываемых для автоматического выполнения задач и взаимодействия с пользователями через текстовый интерфейс, где пользователи отправляют сообщения и получают ответы от бота. При грамотном проектировании возможно создать простой и понятный сценарий взаимодействия с клиентом. Телеграм-бот – это специальный аккаунт, созданный в автоматическом режиме, который позволяет пользователям записываться и корректировать вид услуги в зависимости от своих предпочтений, не покидая сам мессенджер.

## 1.2. Сравнительный анализ аналогов

Ввиду высокой востребованности сервисов онлайн-записи, существует множество платформ, предоставляющих данные услуги. Рассмотрим некоторые решения.

**YCLIENTS** [1]

Это целая экосистема, связанная с онлайн-записью и привлечением клиентов. YCLIENTS предоставляет широкий спектр услуг, таких как онлайн-запись, работа с уведомлениями, финансовый учет, аналитика и многие другие, представленные на рисунке 1. У платформы существует собственное мобильное приложение. Стоит отметить, что сервис предоставляет огромный спектр возможностей и тем не менее требует серьезной с ним интеграции.

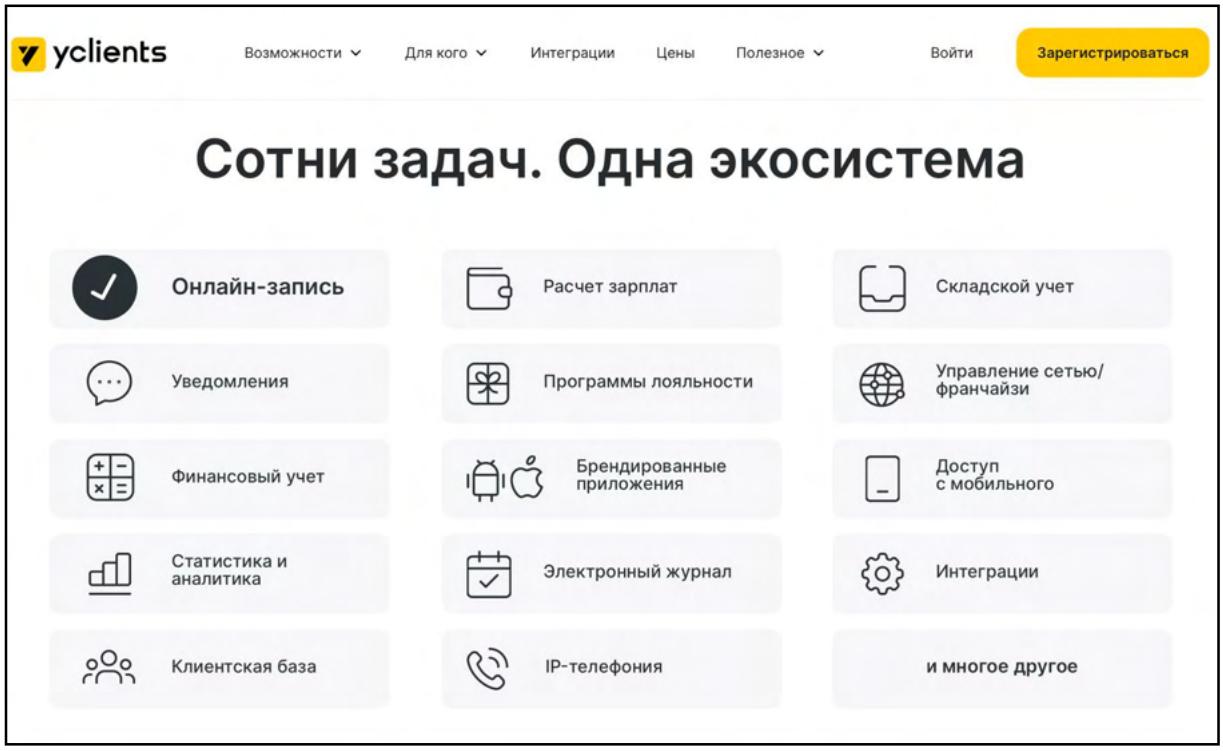


Рисунок 1 – Список услуг, предоставляемых платформой YCLIENTS

Платформа доступна преимущественно на картах, таких как Yandex Maps и 2ГИС и в виде виджетов и ВКонтакте. В мобильном приложении клиента отсутствует возможность поиска услуг, можно записаться лишь на услуги ранее посещенных салонов и мастеров, как показано на рисунке 2.

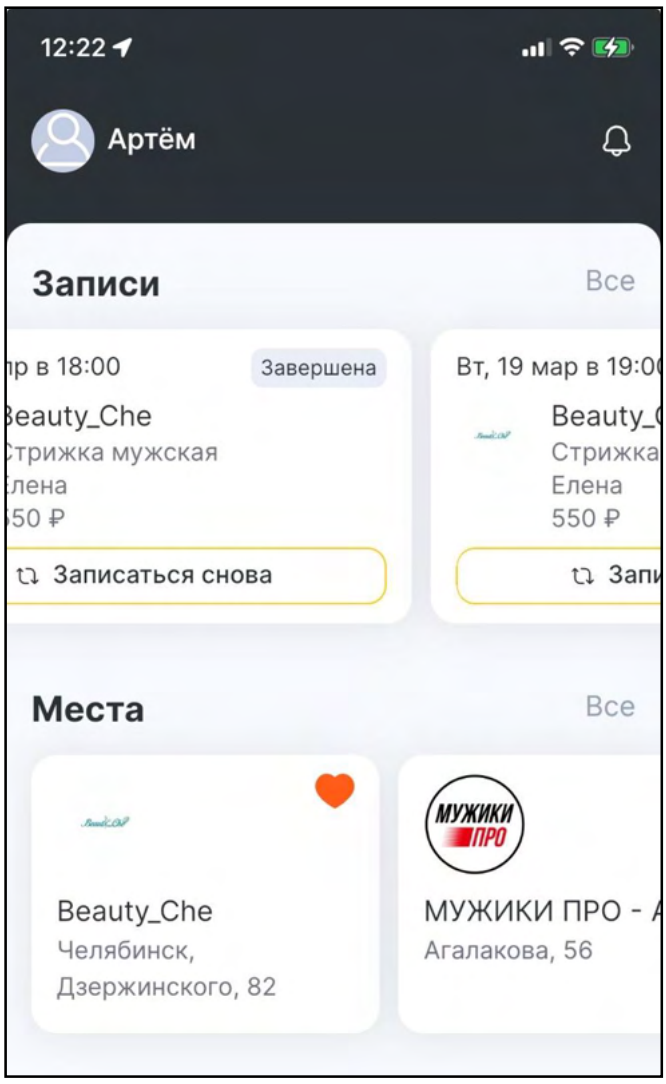


Рисунок 2 – Скриншот мобильного приложения YCLIENTS

У компании существует страница в сети, однако, на ней есть только лэндинг с информацией о компании и продуктах. Для отправки уведомлений доступны ВКонтакте и push-уведомления в собственном приложении YPLACES, для доступа к другим источникам необходимо настраивать интеграции с сервисами партнерами YCLIENTS за дополнительную плату.

**SONLINE** [2]

Программа для салонов красоты с онлайн-записью изображена на рисунке 3. Она менее популярна чем YCLIENTS, во многом с ним схожа, а также предоставляет создание чат-ботов. В отличие от YCLIENTS, данное решение предлагает разработку собственного сайта или мобильного приложения под салон, и уже оно будет отображаться на картах и в интернете.

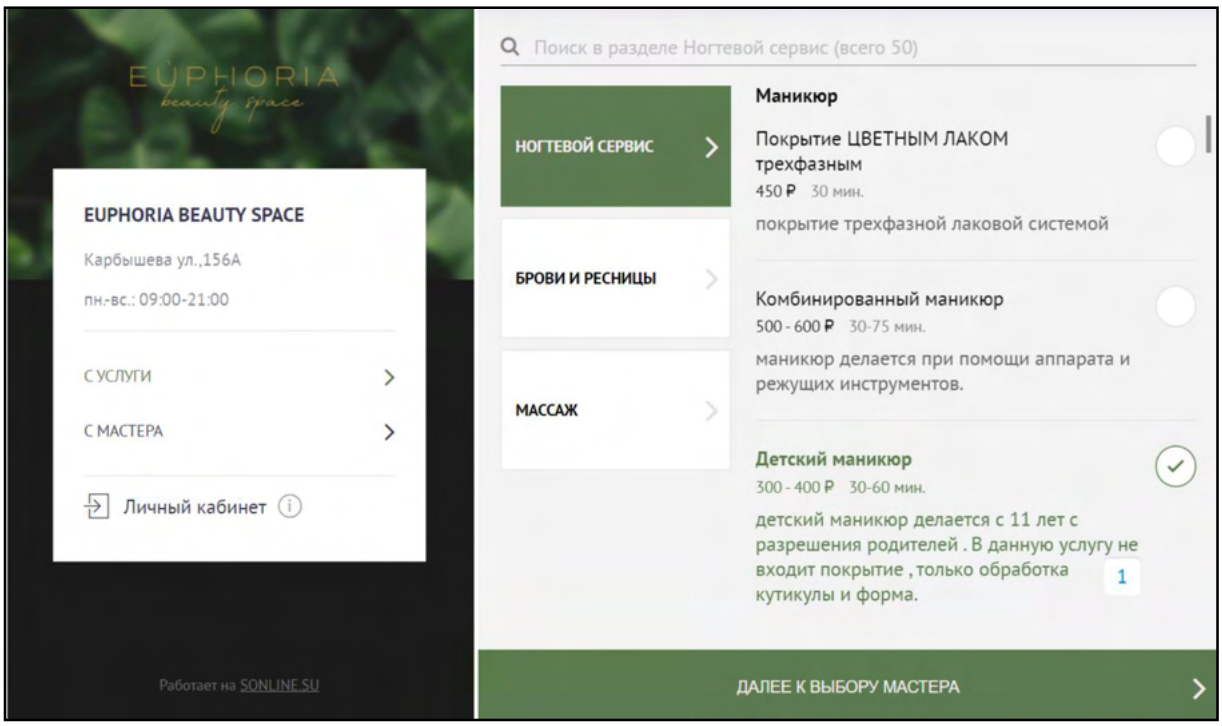


Рисунок 3 – Пример интерфейса сайта, созданного в SONLINE

Помимо этого, платформа заявляет о предоставлении следующих услуг:

1) онлайн-запись с сайта, социальных сетей, поисковиков;
2) функционал для привлечения и удержания клиентов;
3) учет товаров, расчет зарплаты, финансы и аналитика;
4) мобильная версия с максимальным функционалом.

Для работы в системе представителям услуг необходимо связаться с представителями компании и устанавливать специальное программное решение, то есть решение для работников доступно только с устройства, на котором установлена специальное приложение.

**Dikidi**

Данное решение доступно на большем, относительно остальных, количестве платформ. Салоны и мастера здесь располагаются в формате маркетплейса. И к ним клиенты могут записаться через сайт, мобильное приложение, а также виджет ВКонтакте. На рисунке 4 представлена главная страница их сервиса.

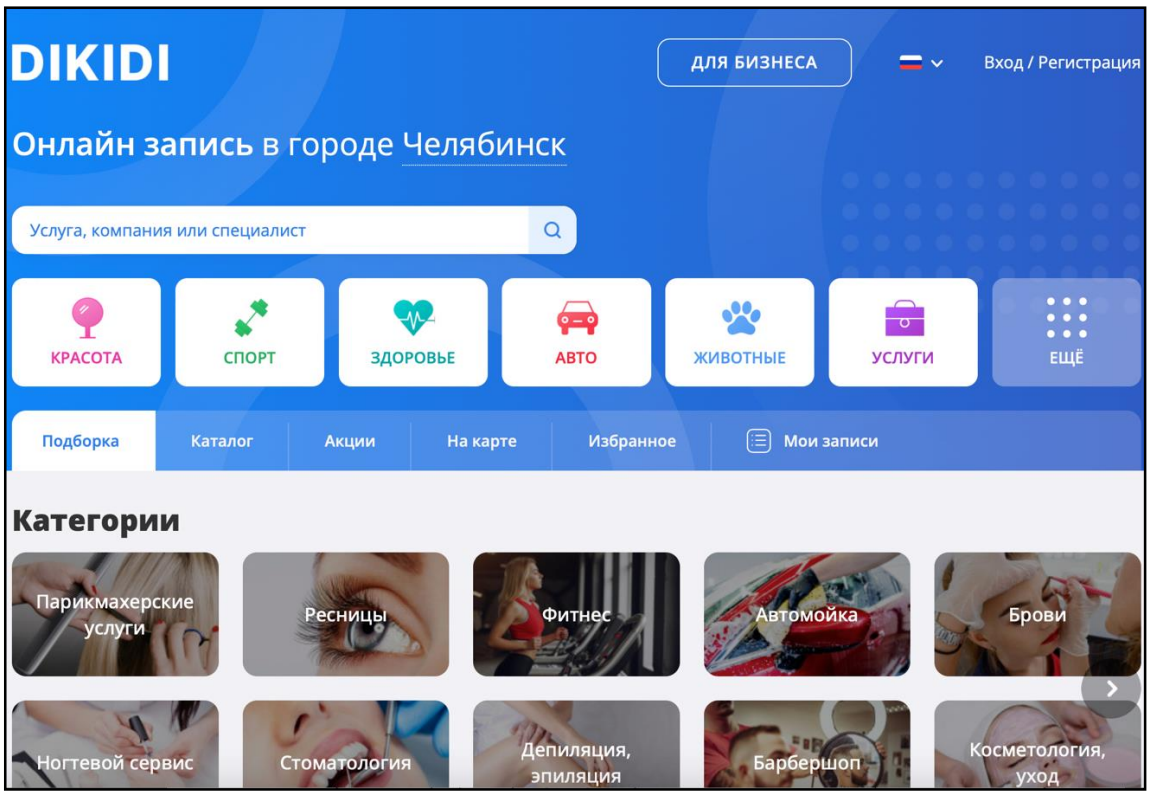


Рисунок 4 – Скриншот главной страницы сайта Dikidi

Сервис делает акцент на своем мобильном приложении, а интерфейс сайта менее функционален.

**Вывод по первой главе**

Таким образом, сравнительный анализ приложений показывает, что в основном аналоги представляют собой крупные платформы, работа с ними нередко подразумевает значительную интеграцию с сервисом, что может служить как достоинством, так и недостатком. Все приложения предоставляют лишь определенное подмножество услуг, акцентируя внимание на определенной платформе, при этом большинство не имеют возможности взаимодействия с клиентами через Телеграм – популярный мессенджер, в котором мастера также общаются с клиентами.

# 2. ПРОЕКТИРОВАНИЕ

## 2.1. Требования к системе

В результате анализа предметной области и обзора существующих решений были сформированы следующие функциональные требования, определяющие какие задачи и действия должны иметь возможность выполнять пользователи.

1. Сервис должен обеспечивать клиенту возможность создавать записи на выбранные услуги салона красоты.
2. Система должна уведомлять клиента о записи за определенное время до начала записи.
3. Сервис должен обеспечивать клиенту возможность выбирать дату и время записи из числа доступных для записи.
4. Сервис должен предоставлять возможность мастеру принимать или отклонять запись клиента.
5. Сервис должен сообщать клиенту об отмене записи мастером.
6. Система должна обеспечивать клиенту возможность добавлять к записи текстовое описание.
7. Система должна обеспечивать доступ к просмотру записей клиента.
8. Система должна обеспечивать возможность отмены предстоящих записей клиента.
9. Система должна ограничивать возможное время записи часами работы заведения.
10. Система должна предоставлять возможность установки иконки приложения на главный экран.

Определяя функциональные требования к системе, можно понять, насколько привлекательным для пользователей будет веб-приложение.

Нефункциональные требования определяют, как при исполнении своего функционала должна работать система. Здесь определяются ограничения на приложение и его составляющие, однако эти требования не связаны

с основной функциональностью. Эти требования также определяют то, насколько качественно будет функционировать система.

1. Система должна отображаться как для мобильных, так и для десктопной устройств.
2. Фронтенд и бэкенд системы должны быть разработаны на языке программирования JavaScript [3].
3. Фронтенд системы должен быть разработан с использованием фреймворка React [8].
4. Бэкенд системы должен быть разработан с использованием фреймворка Node.JS.
5. База данных приложения должна реплицироваться.
6. Система должна использовать технологии PWA.

## 2.2. Диаграмма вариантов использования

На основе требований, предъявляемых к разрабатываемой системе, были разработаны варианты ее использования. На рисунке 5 показаны варианты использования, относящиеся к актерам клиентов салонов.


Рисунок 5 – Диаграмма вариантов использования клиентов

На рисунке 6 представлена вторая часть данной диаграммы, на которой указаны действия доступные остальным актерам сервиса онлайн-записи: представителям бизнеса и работникам.

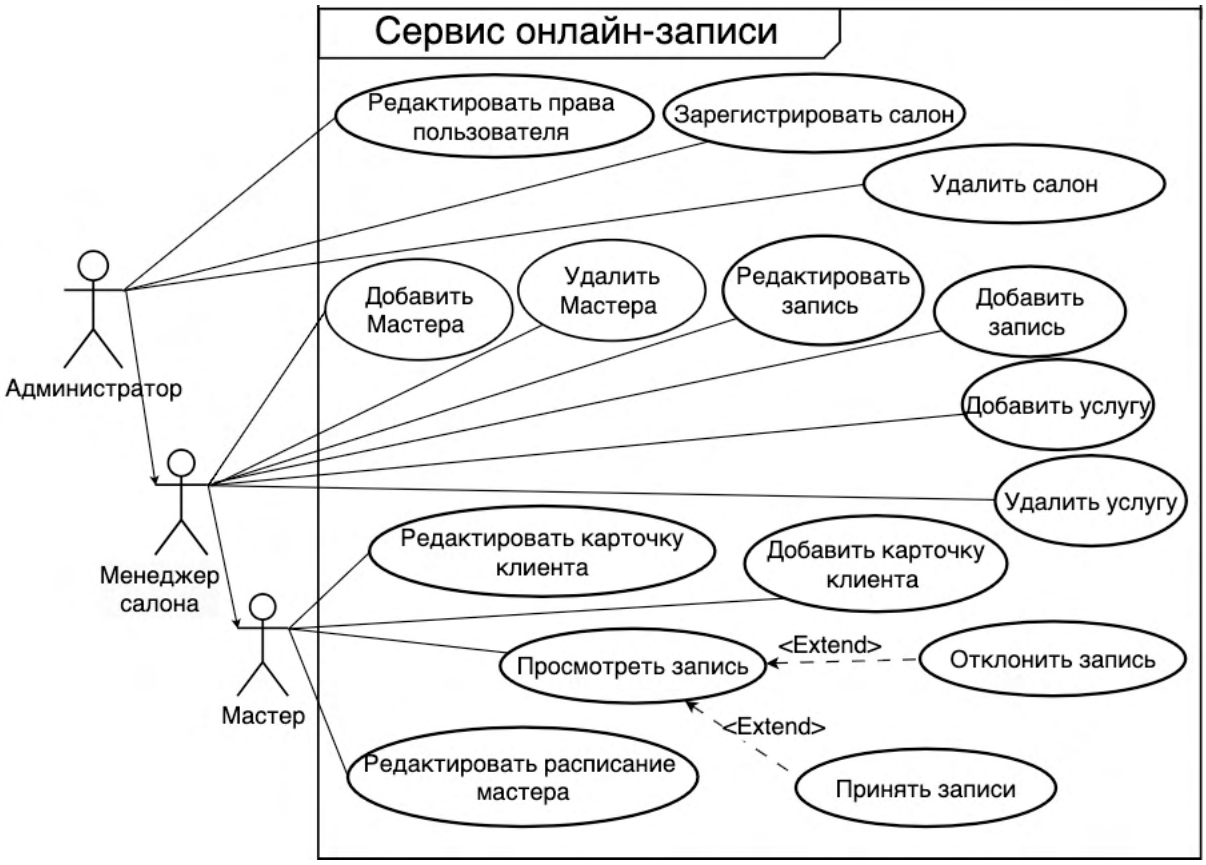


Рисунок 6 – Диаграмма вариантов использования представителей услуг

В ходе проектирования были выделены следующие актеры в системе.

1. Неавторизованный клиент – неидентифицированный пользователь, посещающий сайт без учетной записи.

2. Авторизованный клиент – пользователь сервиса для онлайн-записи, клиент салона красоты.

3. Мастер – пользователь сервиса, являющийся работником салона красоты и оказывающий услуги.

4. Менеджер салона – пользователь сервиса, занимающийся организационными процессами в салоне, имеющий доступ ко всем записям салона.

5. Администратор – пользователь с особыми правами на управление одним или несколькими салонами.

Неавторизованный пользователь обладает меньшим количеством возможностей на платформе, при этом авторизованный клиент наследует эти варианты тем не менее, такой актер может совершать следующие действия:

1) найти подходящую услугу;
2) войти под собственной учетной записью, если она есть;
3) зарегистрироваться на платформе, создав новый аккаунт.

Помимо этого, авторизованному клиенту доступны следующие действия:

1) добавить новую запись на услугу;
2) просматривать свои записи;
3) вносить изменения в расписание собственной занятости, то есть указывать, когда у него есть свободное время для подбора записи;
4) отменить свою ранее добавленную запись;
5) редактировать ранее добавленную запись;
6) получать напоминание о записи.

Мастер может реализовать следующие возможности:

1) просматривать уведомления о добавлении записи клиентом;
2) подтверждать запись клиента;
3) редактировать собственное расписание;
4) отклонять запись своих клиентов;
5) добавлять информацию о клиентах в базу, создавая карточку;
6) получать напоминания о своих сеансах.

Менеджер салона может:

1) просматривать записи клиентов салона;
2) подтверждать запись ко всем подопечным мастерам;
3) отклонять запись клиента салона;
4) добавлять учетную запись новых мастеров в систему;
5) стирать учетную запись мастера из системы;
6) редактировать расписание всех мастеров своего салона;
7) добавлять информацию о клиентах в базу, создавая карточку;

8) добавлять новую услугу в каталог салона;

9) удалять услугу из системы, делая ее недоступной для записи;

10) изменять информацию о записи;

11) редактировать информацию в карточках клиентов салона;

12) добавлять новую запись.

Администратору на платформе доступны те же действия, что и менеджеру, однако он обладает наибольшим количеством прав на платформе и еще может:

1) добавлять новый салон на платформу;

2) удалять салон с платформы;

3) редактировать права остальным пользователям.

**Вывод по второй главе**

В данной главе были рассмотрены установлены требования к системе, соответствие им итогового программного продукта будет во многом определять, насколько он будет устойчив и применим в условиях бизнеса.

Также были выявлены ключевые актеры, для каждого определены варианты того, что каждый может получить от системы, их права на платформе и ограничения, которые необходимо учесть в разработке.

Выявлен перечень интерфейсов, которые необходимо предоставить пользователям системы.

# 3. АРХИТЕКТУРА СИСТЕМЫ

## 3.1. Общее описание архитектуры системы

В настоящее время существует множество подходов к реализации архитектуры веб-приложений. Для реализации сервиса для записи за основу была выбрана клиент-серверная архитектура, для ее описания была выбрана нотация UML [10]. Unified Modeling Language был выбран так как его синтаксис предоставляет возможность визуализации связей составных элементов в проекте, такой подход позволил определить степень самостоятельности элементов архитектуры. В качестве обозначений для составляющих были выбраны компонентные блоки [6], так как каждый из них обеспечивает свою отдельную часть функциональности, полученная диаграмма представлено на рисунке 7.

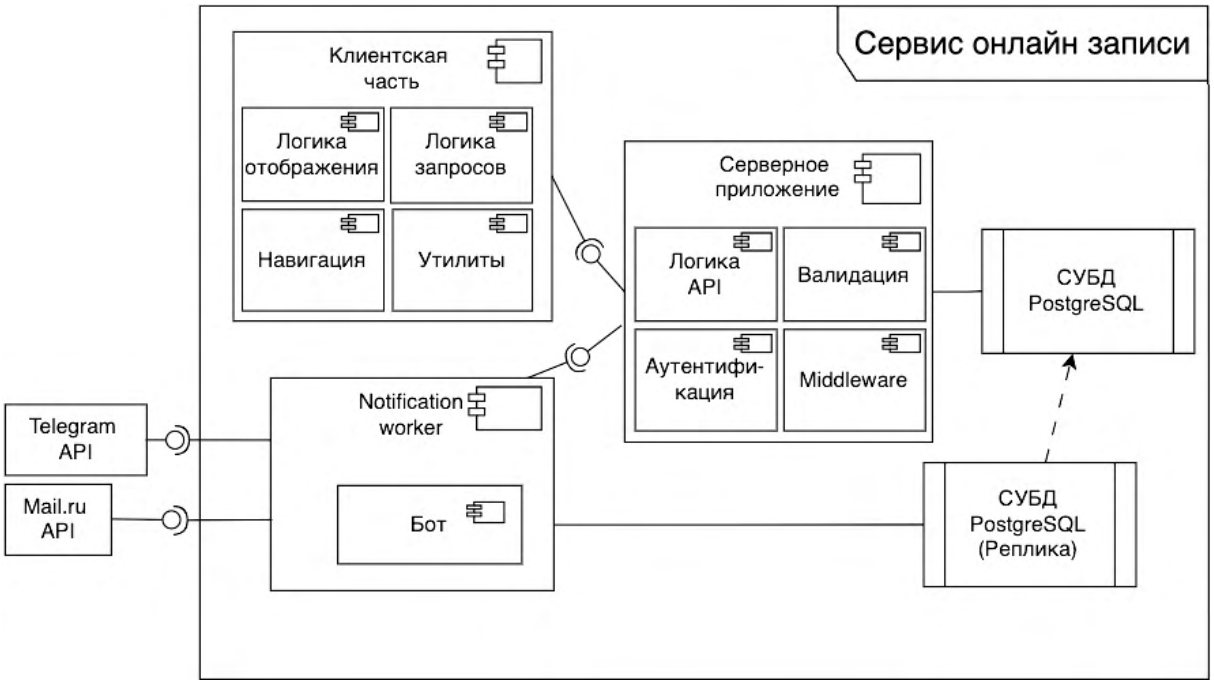


Рисунок 7 – Архитектура веб-приложения

Архитектура состоит из нескольких основных компонентов, включающих клиентское приложение, серверное приложение, а также в архитектуре проекта предусмотрен вспомогательный `Notification worker` для рассылки уведомлений, а за хранение и менеджмент данных отвечает система управления базами данных.

Серверное приложение обеспечивает обслуживание API запросов и их структуру, создавая интерфейсы для клиентской части и бота. Данный компонент также производит обработку и валидацию поступающих данных, что критически важно для всего функционирования системы в целом. Оно определят логику взаимодействия сущностей и аутентификацию пользователей, то как и какие данные могут быть представлены запрашивающей стороне. Исходя из этого, данный элемент по сути является центральным в системе.

Клиентское приложение отвечает за отрисовку интерфейса для клиентов, мастеров, менеджеров и администраторов, а также определяет принцип отображения и навигации по сервису.

Задача данного элемента архитектуры интерпретировать ответы сервера в понятную для пользователя форму, а также запрашивать необходимую информацию, в те моменты, когда это необходимо. Клиентская сторона формирует запросы к серверу на основании действий пользователей. А также в нем отведено место для утилит, упрощающих первичную валидацию, работу с датами и временем и обращение к серверу. То есть утилиты заняты той обработкой информации, которые нужно производить каждый раз для улучшения понимания данных пользователем.

Для увеличения надежности и качества системы на коммуникацию с серверным приложением ставится веб-сервер, он перенаправляет запросы на сервер. Еще одной его задачей является скрытие характеристик компонента, предоставляющего API, а также управление трафиком, что помогает добиться обработки запросов с большей стабильностью и сглаживания нагрузок на сервер.

Бот – это составляющая компонента `Notification worker`, его задача отвечать за взаимодействие с API Telegram. А именно направлять пользователей в мини-приложение и уведомлять их о предстоящих записях. Весь компонент в целом работает как менеджер напоминаний.

Система управления базой данных отвечает за хранение и преобразование данных системы, работу с объектами системы, предоставление их серверному приложению. В архитектуре предусмотрена репликация базы данных. Для уменьшения нагрузки на сервер основная часть запросов от `Notification worker` будет приходиться на реплику базы данных, которая доступна только в режиме для чтения.

Такая структура обусловлена тем, что `Notification worker` будет совершать проверки на необходимость уведомлений с определенной периодичностью, при этом, лишь иногда будет возникать необходимость модифицировать записи в базе данных, что будет производиться уже через API серверного приложения. Таким образом, `Notification worker` можно упростить, так как ему не нужно будет учитывать все связи в базе данных и логику. В то же время, с сервера уходит нагрузка регулярных запросов, а у основной базы данных появляется резервная копия, с которой в случае неполадок может производится бэкап системы.

Интерфейсы Телеграм и почтового хостинга являются внешними по отношению к системе, и для реализации функций онлайн-записи необходимо учесть то какие требования данные интерфейсы предоставляют к запрашивающей стороне, как обеспечить безопасность такого соединения.

### 3.2. Проектирование структуры серверного приложения

В клиент-серверной архитектуре широко используется RESTful API, этот архитектурный подход был выбран за основу в моем проекте для организации общения между сервером и клиентами и между `Notification worker` и сервером, чтобы единообразить интерфейс и упростить процесс разработки коммуникаций с различными компонентами.

Чтобы обеспечить данный подход было принято решение организации структуры бэкенда по принципу Model–Router–Controller (рисунок 8).

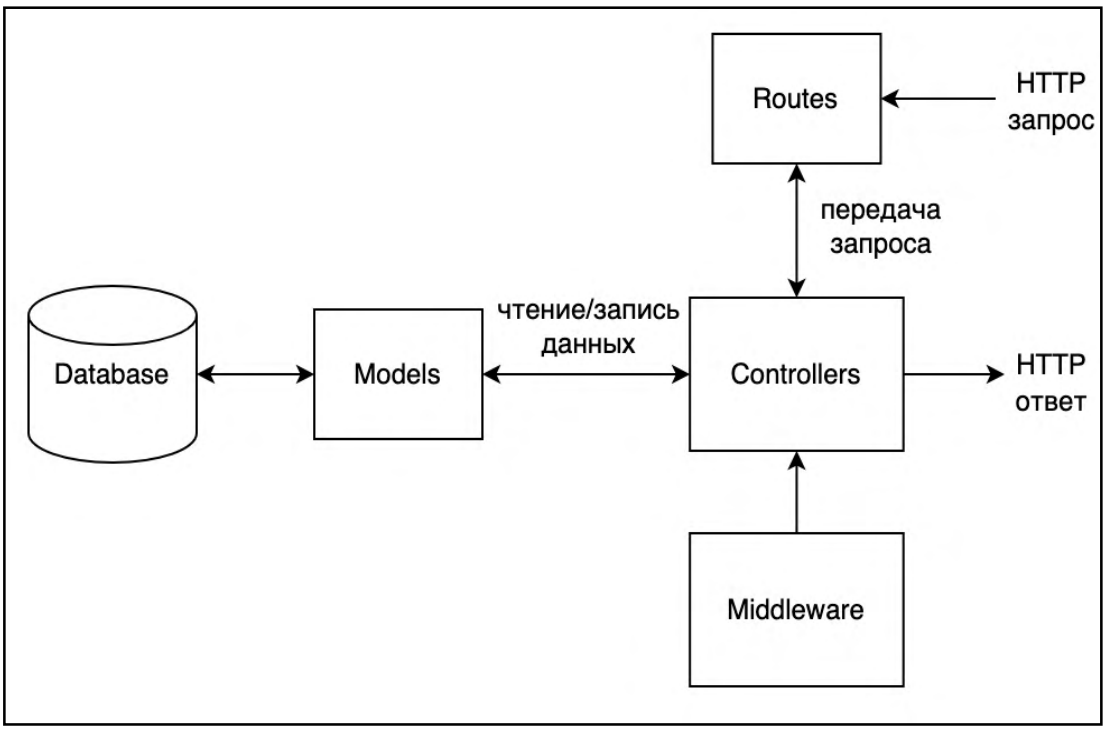


Рисунок 8 – Схема работы подхода Model-Router-Controller

Как показано на схеме, routes принимают и просматривают HTTP запросы, в которых прописаны пути URL. Routes становятся посредниками. Определяется тип URL запроса, какие данные получены и в какой `controller` требуется передать.

Здесь `controller` – это функция, которая непосредственно работает с данными, получаемыми от базы данных с помощью `models`. В свою очередь, `models` – это описания объектов, хранящихся в базе данных, которые необходимо прочитать или записать.

Этот принцип можно расширить с помощью элемента `middleware` – функций часто используемых различными `routes`. Данное дополнение позволило улучшить читаемость кода и оптимизировать методы обработки запросов.

Затем `controller` формирует ответ, в котором содержатся данные, либо ответа или ошибки, то есть HTTP-ответ.

Суть данного подхода в стандартизации запросов ответов и обеспечения модульности разработки бэкенда.

### 3.3. Проектирование базы данных

Схема базы данных представлена на рисунке 1 в приложении Б.

Для создания схемы использовалась модель проектирования Entity Relation (Сущность Связь) в нотации Джеймса Мартина [28]. В ходе проектирования базы данных необходимо было учитывать специфику предметной области. Как можно увидеть на схеме все связи типа многие-ко-многим организованы в базе данных через join-таблицы. В них, помимо идентификаторов-ключей, можно хранить метаданные по типу дат изменения, создания и удаления, что помогает в составлении бизнес-аналитики для представителей услуг.

Связи между `masters` и `services` обозначают, что мастера могут оказывать соответствующие услуги. А когда у `clients` есть связь с `master` или `services`. Это означает, что они отмечены клиентом как предпочитаемые или избранные.

Такие таблицы, как `managers`, `master`, `clients`, `admins` представляют пользователей, поэтому у каждой определена связь с `settings`, так как в настройках содержатся общие для всех пользователей поля в том числе необходимы для аутентификации, указания предпочтений в использовании сервиса.

Так как у каждого салона нужна своя база клиентов, между таблицами `salons` и `clientbases` есть связь, при этом по причине того, что такие данные могут передаваться и разделяться, поэтому здесь была выбрана связь многие-ко-многим. Таким образом, например, филиалы смогут делиться между собой клиентскими базами. В ходе работы с консультантом, индивидуальным предпринимателем, владельцем салона красоты, был проведен ряд интервью и исследований предметной области. Было выяснено, что необходимо хранить информацию о клиентах, такие как предпочтения, описание, чтобы улучшить сервис и персонифицировать услуги. Так как представителям бизнесов необходимо хранить свою информацию о каждом клиенте, для улучшения качества услуг, поэтому была создана таблица

clientcards, в которой представители услуг смогут указывать необходимую информацию о клиенте, объекты в данной таблице могут как иметь, так и не иметь связь с таблицей clients, так как в базе клиентов могут указываться не имеющие учетной записи веб-приложения.

В схеме отражена сущность posts – статьи, в которых связанные с этой таблицей смогут делиться дополнительной информацией о себе на платформе.

Для организации и планирования системы записей были созданы спроектированы следующие сущности:

1) уведомлений – notifications хранящих напоминания клиентов и мастеров для системы напоминаний;

2) периодов или слотов – periods записи в базу данных позволяющие указывать доступность того или иного промежутка времени;

3) расписаний – schedules, здесь агрегируются периоды и записи пользователей;

4) записей на услуги – appointments;

5) карточки клиента – clientcards представляющая клиентов в записях мастеров.

Для пользователей, представленных таблицами master и client создаются расписания, к которым привязываются periods – сущности для описания слотов и занятости как для клиентов, так и для мастеров, appointments, являются непосредственно записями на услуги.

**Вывод по третьей главе**

В ходе работы над архитектурой были описаны подходы и архитектурные принципы, по которым были созданы ключевые компоненты системы.

# 4. РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ СИСТЕМЫ

## 4.1. Средства реализации

Для полноценного функционирования сервиса записи на бьюти-услуги необходимо, чтобы страница приложения могла делать запросы к базе данных, имела доступ к бизнес-логике. А также очень важно, чтобы сервис мог выдерживать нагрузку при растущем количестве клиентов. С этой целью, перед разработкой всего проекта, был проведен обзор популярных решения для разработки бэкенда.

Node.js [13] – это широко применяемая платформа для разработки бэкенда, благодаря своей высокой производительности и масштабируемости. Он позволяет разработчикам использовать один и тот же язык программирования JavaScript для фронтенда и бэкенда, что упрощает процесс разработки. Однако, Node.js может быть сложным для новичков и требует хорошего понимания JavaScript.

Ruby on Rails [14] – это другой популярный выбор для разработки бэкенда. Он предлагает мощные инструменты для быстрой разработки и имеет большое сообщество разработчиков. Однако, Ruby on Rails может быть медленнее, чем Node.js, и требует более глубокого понимания Ruby.

Django [15] – это еще один популярный фреймворк для разработки бэкенда на Python. Он предлагает мощные инструменты для безопасности и управления данными. Однако, Django может быть сложным для новичков и требует хорошего понимания Python.

Flask [16] – это легковесный фреймворк для разработки бэкенда на Python. Он предлагает большую гибкость и простоту использования. Однако, Flask может быть менее мощным, чем другие фреймворки, и требует хорошего понимания Python.

Spring [17] – это популярный фреймворк для разработки бэкенда на Java. Он предлагает мощные инструменты для управления данными и безопасности. Однако, Spring может быть сложным для новичков и требует хорошего понимания Java.

После проведенного анализа предметной области и рассмотрения вариантов решений для реализации проекта, исходя из специфики веб-приложения и рассматриваемых требований, для работы были выбран Node.js. Несмотря на порог вхождения, эта программная платформа обладает преимуществами, которые крайне важны для сервиса онлайн-записи: масштабируемость, производительность.

Для работы над фронтендом необходимо было определить решение, которое дало бы возможность создавать графические интерфейсы. Оно должно поддерживаться большинством браузеров, а также позволять динамически отображать данные, поступающие с бэкенда и делать их интерактивными.

Под эти характеристики подходит React [8] – данная библиотека позволяет создавать компоненты, которые являясь связкой JavaScript и HTML, дают возможность отображать элементы страниц, реагирующие на действия пользователей и ответы серверного компонента. Это достигается за счет хранения состояний внутри самих компонентов и вызовов callback-функций.

Более того, компоненты данной библиотеки можно переиспользовать, то есть вызывать на различных страницах веб-приложения и в разных участках страниц. А за счет возможности передавать в них параметры от родительских элементов, разработка на нем становится более гибкой.

React является одной из самых популярных платформ разработки фронтенда, и под него создано множество библиотек, в разработке это стало преимуществом, так как это позволило подстраивать его под нужды проекта. Под React написаны библиотеки UI-компонентов, что предоставляет возможность ускорить разработку и создавать удобные интерфейсы, затрачивая меньшее количество времени на проработку их дизайна. В качестве одной из таких библиотек был взят Ant Designt, в нем есть компоненты стандартных форм.

Также стоит отметить, что при сочетании Node.js и React упрощается процесс разработки всего приложения, так как в обоих решениях используется язык JavaScript, а значит язык, на котором будет разрабатываться логика будет единым, что снизит вероятность ошибок при интерпретации передаваемых данных.

База данных построена на основе СУБД PosgreSQL. Для того, чтобы общаться с ней и интерпретировать данные в системе был применен Node.js фреймворк Sequelize. Этот инструмент позволяет работать как в парадигме реляционных баз данных, так и в объектно-ориентированной парадигме.

В проекте он используется для преобразования и хранения объектов в реляционной базе данных PostgreSQL. С помощью этого фреймворка описываются сущности системы с помощью классов, а методы реализующие запросы к базе данных, преобразуют эти экземпляры классов в записи в реляционном виде. Таким образом, стало возможно работать в объектно-ориентированной парадигме и сохранять данные в читаемом всеми модулями виде.

Для того, чтобы обеспечить качественное функционирование приложения, желательно, чтобы запросы и ответы между клиентской и серверной стороной как-то оптимизировались и обслуживались с применением мер защиты.

Для этого существуют решения обслуживающие HTTP-соединения и выступающие в качестве балансировщиков нагрузки. Представителем данного класса решений является NGINX [21], для реализации данного функционала его можно сделать посредником в клиент-серверной коммуникации.

Этот программный продукт при внедрении в систему становится веб-сервером и обратным прокси-сервером [22], что позволяет делать более комплексную логику HTTP-запросов и балансирует их потоки между клиентом и сервером, помимо этого ограничивая прямой доступ клиента к серверу, что в рамках сети интернет увеличивает надежность платформы.

В силу архитектурных особенностей и для обеспечения изолированности компонентов системы, все они были обернуты в контейнеры [11] с помощью Docker, этот инструмент ускоряет разработку и помогает легче масштабировать сервисы и модули системы, создавая в каждом контейнере среду, необходимую для функционирования и подгружая все нужные зависимости. Это упрощает процесс развертывания всей системы.

В серверной части для обработки запросов и отправки HTTP-запросов использовалась библиотека Express [18]. А, в свою очередь, для их формирования на клиентской стороне задействовалась библиотека Axios [20].

Проект разрабатывался в среде программирования Visual Studio Code, в гибком инструменте для разработки программных систем на различных языках программирования. В нем реализуется принцип модульной адаптации под нужды разработчика. То есть при необходимости задействовать новый язык программирования или внедрения инструмент работы с программными утилитами, можно подгрузить требуемые расширения из широкой библиотеки внутри Visual Studio Code.

Для того, чтобы создавать удаленные бэкапы проекта и отслеживать историю изменений кода использовалась платформа GitHub [23].

Для того, чтобы тестировать систему на тех платформах, где нужно соединение по протоколу https, например, в мини-приложениях Телеграм, создавались туннели с помощью ngrok [24]. Этот инструмент предоставляет временный публичный адрес в сети, а трафик с него перенаправляет на указанный порт.

### 4.2. Реализация компонентов системы

Платформа состоит из нескольких модулей, каждый из которых отвечает за свою часть приложения. Список основных компонентов, реализующих веб-приложение также отражена в организации файлов в проекте (рисунок 9).

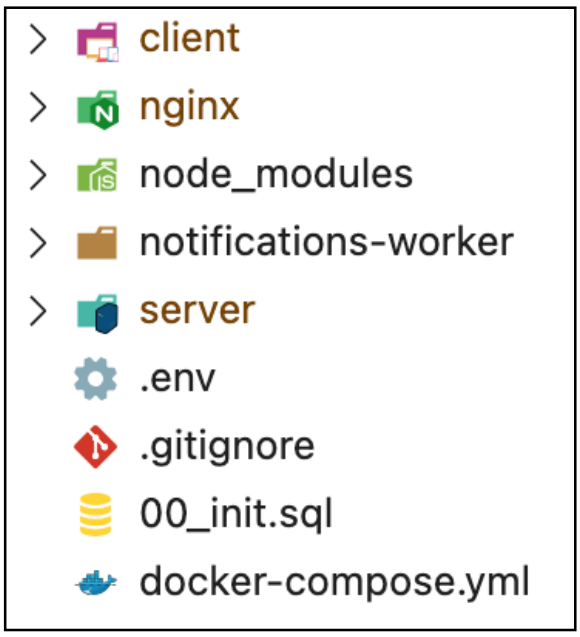


Рисунок 9 – Файловая структура проекта

Ключевыми директориями здесь являются «client», «nginx», «notifications-worker», «server». Под каждую из них настраивается Docker-контейнер [12]. Он определяет среду, в которой будет запускаться компоненты системы, подгружая зависимости и изолируя части системы для более стабильного функционирования. Это упростило процесс разработки и отладки приложений. Поэтому в каждой из перечисленных директорий в корне создается файл «Docker»  или «Docker.dev».

Код исполняемого файла «Docker.dev» для фронтенда, разработанного в директории сервиса представлен в листинге 1. В нем указывается какой образ необходимо использовать для контейнеризации, из какого файла подгружать зависимости, библиотеки, задействованные сервисом, а также команды, с помощью которых и будет происходить запуск кода в контейнере.

Листинг 1 – Содержание файла «Docker.dev» в директории «client»

```
FROM node:14.14.0-alpine
WORKDIR /app
COPY ./package.json ./
RUN npm i
COPY . .
CMD ["npm", "run", "start"]
```

Взаимодействие контейнеров и общий запуск обеспечивается инструкциями прописанными в «docker-compose.yaml» в папке составленной со всеми файлами веб-приложения, таким образом применяется подход мультиконтейнерного приложения. Под каждый контейнер определяется название его как сервиса, то, на запуск каких сервисов они полагаются, переменные сред под каждый сервис, открываемые и переназначаемые порты, условия запуска и перезапуска, а также структура отдаваемых пространств `volumes` внутри контейнеров.

В этом же файле описывается создание образов систем управления базами данных PostgreSQL, их образы описаны на официальном сайте Docker [12]. В качестве основной базы данных выступает `postgres_primary`, она открыта для чтения и записи серверному приложению. Ее репликой задается `postgres_replica`. К ней `notifications_worker` совершает запросы, они разрешены лишь в режиме `read-only`, чтобы избежать возможных конфликтов между основной базой данных и репликой. Код их реализации в «docker-compose.yaml» представлен в листинге 1 приложения А.

В общих для обеих баз данных параметрах `x-postgres-common` задается проверка их работы – `healthcheck`. Ее в качестве зависимости `depends_on` на свою СУБД указывают сервисы `api` и `notifications`, это, соответственно, серверное приложение и `notifications_worker`, так как именно эти компоненты производят запросы к хранилищам. Это позволяет сделать так, чтобы сервисы дожидались запуска базы данных перед тем, как попытаться установить с ними соединение.

### 4.2.2. Серверная часть

Модуль server написан на Node.js. Он также был обернут в контейнер версии node:14.14.0-alpine. Выбор alpine в качестве прототипа образа обусловлен его легковесностью и упором на безопасность, что критически важно для разработки устойчивых приложений.

Структура серверной части в проекте отображена на рисунке 10.


Рисунок 10 – Структура серверной части проекта

В проекте сервер выступает основным и связующим звеном между клиентской стороной и сервером. Поэтому его структура содержит составляющие, необходимые для коммуникации как с клиентом, так и с базой данных. Так как серверная логика, в силу специфики предметной области, манипулирует сущностями, и интерпретирует их для записи в базу данных PostgreSQL, то для общения с ней используется ORM Sequelize.

Для того, чтобы создавать записи в базе данных, сначала необходимо установить с ней соединение. Так как в проекте используется технология объектно-реляционного отображения, реализуемого с помощью Sequelize, то и, непосредственно, само подключение к базе данных происходит с помощью вызова, встроенной в эту библиотеку, функции `new Sequelize()`. В данную функцию необходимо отдать параметры базы данных, эти данные передаются в переменных окружения. А задаются они окружению внутри конфигурационного файла «docker-compose.yaml», описанного выше. Таким образом ключи не хранятся в исходном коде.

Чтобы задать объекты, которые затем будут размещаться в базе данных, необходимо определить их как Sequelize-модели. Для этой цели было выделена директория «models». В ней, каждая из своего JavaScript файла,

экспортируется переменная, хранящая результат выполнения метода `define`. В нем назначаются поля сущностей, а также типы их значений.

Таким образом, в директории «models» задаются все сущности сервиса, такие как клиент, мастер, администратор, запись, салон. Затем все экземпляры класса импортируются в файл «index.js» – главный файла этой директории, где описываются типы связей между моделями. Пример создания такого класса представлен в листинге 2.

Листинг 2 – Пример создания модели объекта салона

```
module.exports = (sequelize, DataTypes) => {
  const Salon = sequelize.define('salons',
      {
        UID:{
          type: DataTypes.UUID,
          defaultValue: DataTypes.UUIDV4,
          primaryKey: true,
          allowNull: false
        },
        main_image: { type: DataTypes.STRING },
        images: { type: DataTypes.ARRAY(DataTypes.STRING),},
        name: {
          type: DataTypes.STRING,
          allowNull: false,
        },
        slogan: { type: DataTypes.STRING },
        description: { type: DataTypes.STRING, },
        address: { type: DataTypes.STRING },
        latitude: { type: DataTypes.FLOAT },
        longitude: { type: DataTypes.FLOAT },
        phone: { type: DataTypes.STRING },
        email: { type: DataTypes.STRING },
        website: { type: DataTypes.STRING },
        telegram: { type: DataTypes.STRING },
        vk: { type: DataTypes.STRING },
        master_count: { type: DataTypes.INTEGER },
        client_count: { type: DataTypes.INTEGER },
      },{ paranoid: true, });
  return Salon;}
```

В корне серверного приложения находится файл «index.js», он является основным в данном компоненте. В нем собираются все прочие элементы и происходит их проверка и запуск. Здесь определяется объект `app`, который будет представлять собой серверное приложение, он создается с помощью фреймворка Express.js методом `express()`. В `app` передаются заголовки, обработчики различных частей запросов и ответов, в том числе маршруты, определяющие какие запросы возможны от клиента к серверу. В

листинге 3 представлен код, отвечающий за конфигурацию переменной `app`.

Листинг 3 – Код задания присвоения обработчиков заголовков, тела и URL

```
const app = express();
app.use(cors());
app.use((req, res, next) => {
  res.setHeader("Access-Control-Allow-Origin", "*");
  res.setHeader("Access-Control-Allow-Methods", "GET, POST, PUT, DELETE");
  res.setHeader("Access-Control-Allow-Headers","Content-Type,
Authorization");
  res.setHeader("Access-Control-Allow-Credentials", true);
  next();
});
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));
app.use("./routes/main.routes")(app);
app.use("./routes/auth.routes")(app);
```

В данном фрагменте кода присваиваются два набора маршрутов `routes`: `main.routes` и `auth.routes`. Их задача задавать список URL-запросов, которые сервер сможет в дальнейшем обслужить и вернуть результат, тем самым предоставляя API клиентской стороне. В моем проекте они описаны в директиве `routes`.

В «auth.routes.js» задаются три пути, их описание представлено в таблице 1.

Таблица 1 – Список API авторизации и аутентификации

| Тип запроса | Маршрут | Описание |
|---|---|---|
| PUT | /auth/login | Вход пользователя в учетную запись с помощью логина и пароля |
| POST | /auth/signup/:userType | Регистрация пользователя типа, переданного в параметре `userType` |
| GET | /auth/verifyToken | Запрос на проверку актуальности и достоверности токена пользователя |

Для обеспечения аутентификации и регистрации пользователей была разработана система, состоящая из модели настроек, связанных с каждым типом пользователя, методов. При создании пользователя создаются его настройки, пароль хэшируется непосредственно при создании настроек методами `create()` и `bulkCreate()`. Такое поведение задается с помощью хуков `beforeCreate` и `beforeBulkCreate`, соответственно в них проверяется

наличие пароля, после чего он преобразуется в хэш. Хэш получается путем передачи строки пароля и числа итераций генерации salt – дополнительной меры защиты хэша от подбора – в функцию `hashSync()` JavaScript пакета `bcrypt`. В качестве токенов используются JSON Web Tokens или JWT [28], в них оборачивается идентификатор настроек менеджера, администратора, клиента или мастера. Это происходит с использованием заданного ключа, такой токен создается на сервере при успешном входе в систему с логином и паролем и передается клиенту. На клиентской стороне он хранится, а передается вместе с запросами, что дает возможность сохранять пользователя авторизованным на платформе до истечения срока действия токена, задаваемого на сервере. В тех маршрутах, где требуется удостовериться в наличии у пользователя доступа к данным.

В директории «controllers» задаются функции, которые определяют, как и в каком виде будут запрашиваться данные из PostgreSQL и как они будут туда записываться, модифицироваться и удаляться из базы данных. Пример метода реализующего обработку запроса представлен в листинге 2 приложения А.

Те же методы, которые часто использовались в различных контроллерах были вынесены в отдельную директорию «middleware». Эти методы в основном устанавливают процесс авторизации, и определения типа пользователя через токен JWT, к таким методам относятся `verifyJWT()`, `verifyAdmin()`, `verifyClient()`, `verifyMaster()`, `verifyManager()`, `checkUserType()`. Они принимают на вход объекты запроса и ответа, а также функцию, которую необходимо выполнить далее, в случае успешной проверки.

Они встраиваются в процесс раутинга и таким образом позволяют выяснить возможности и уровень доступа пользователя (листинг 4).

Листинг 4 – Пример пути с применением `middleware`

```
app.post("/api/salons",
  [verifyToken, verifyAdmin],
  mainController.createSalon);
```

В таблице 2 представлены объяснения по принципу функционирования путей в `main.routes`.

Таблица 2 – Список основных API путей

| Тип | Маршрут | Доступ | Описание |
|---|---|---|---|
| GET | /:userType/settings | Авторизованный пользователь | Получение пользователем своих настроек |
| PUT | /:userType/settings | Авторизованный пользователь | Изменение пользователем своих настроек |
| GET | /salons | Все пользователи | Получение списка всех салонов на сервисе с возможностью применения фильтров, переданных в теле запроса |
| GET | /salons/:salonId | Без ограничений | Получение информации о салоне по его ID |
| POST | /salons/ | Администратор | Создание салона на платформе |
| PUT | /salons/:salonId | Менеджер или администратор салона | Редактирование данных о салоне |
| GET | /masters | Без ограничений | Получение списка мастеров |
| GET | /masters/:masterId | Без ограничений | Просмотр информации о мастере |
| GET | /masters/: masterId/schedule | Все пользователи | Посетителям возвращаются только периоды, мастерам все свое расписание, администраторам и менеджерам только подчиненных |
| GET | /client/schedule | Клиент | Просмотр клиентом собственного расписания |
| POST | /:userType/periods | Авторизованный пользователь | Добавление слота в расписание, клиентам и мастерам только в своем, менеджерам и администраторам у подчиненных |
| PUT | /:userType/periods | Авторизованный пользователь | Изменение слота в расписании, клиентам и мастерам только в свое, менеджерам и администраторам – подчиненным |
| POST | /:client/appointemt | Клиент | Создание новой записи |
| PUT | /:userType/ appointments | Авторизованный пользователь | Изменение записи.<br>Мастер и пользователь только к своей, администратор и менеджер только к подчиненным мастерам |

В данной таблице вместо «:userType» запрашивающая сторона подставляет тип пользователя совершающего запрос.

Также следует выделить, что логика создания записи и редактирования расписания были реализованы на основе транзакций, так как процесс

создания записи и проверки пересечения периодов многоэтапен, то при возникновении ошибки, процесс может быть нарушен, что может вести за собой некорректные ситуации, нарушающие бизнес-логику. Поэтому, когда процесс нарушается, благодаря sequelize-транзакциям может быть свернуты модификации базы данных, которые составляли транзакцию. После чего можно предпринять меры предупреждающие о возникновении ошибки.

Связи между моделями производятся с помощью встроенных в Sequelize методов. Для каждой пары моделей задавалось по два соединения, чтобы предоставить обработчикам возможность делать запросы по связям объектов в обе стороны. Это происходит в файле «associations.js» (листинг 5).

Листинг 5 – Пример создания связи типа многие-ко-многим

```
// client to notification one-to-many
Client.Notifications = Client.hasMany(Notification);
Notification.Clients = Notification.belongsTo(Client);
```

### 4.2.1. Клиентская часть

Компонент `client` реализован на React и также обернут в Docker-контейнер. Он отвечает за предоставление интерфейса веб-приложения для записи пользователям. Компонентная структура фреймворка React позволила гибко отображать данные под каждый тип пользователя. Для этого специфичные для клиентов, мастеров, администраторов, и менеджеров компоненты были описаны в соответствующих директориях и отображаются лишь если статус пользователя соответствует.

Для того, чтобы делать единообразные запросы к серверу была создана функция `useApi` она расположена в директории «/client/src/hook» и при ее вызове в переменные `data`, `loading`, `error` передаются состояния [26], которые, соответственно, хранят полученные данные, статус загрузки, описание и статус ошибки, если она возникнет при запросе. Эти со-

стояния задействуются каждый раз, когда функция `fetchData()` вызывается и делает запрос к серверу, эта функция также описана в `useApi`. Код метода `useApi` представлен в листинге 3 приложения А.

Для того, чтобы пользователям предоставлялась возможность подбирать услуги, на сайте предусмотрена возможность фильтрации по типу. Она осуществляется за счет обновления состояний, хранящих списки услуг, в `useEffect()`. Домашняя страница клиента представлена не рисунке 11.

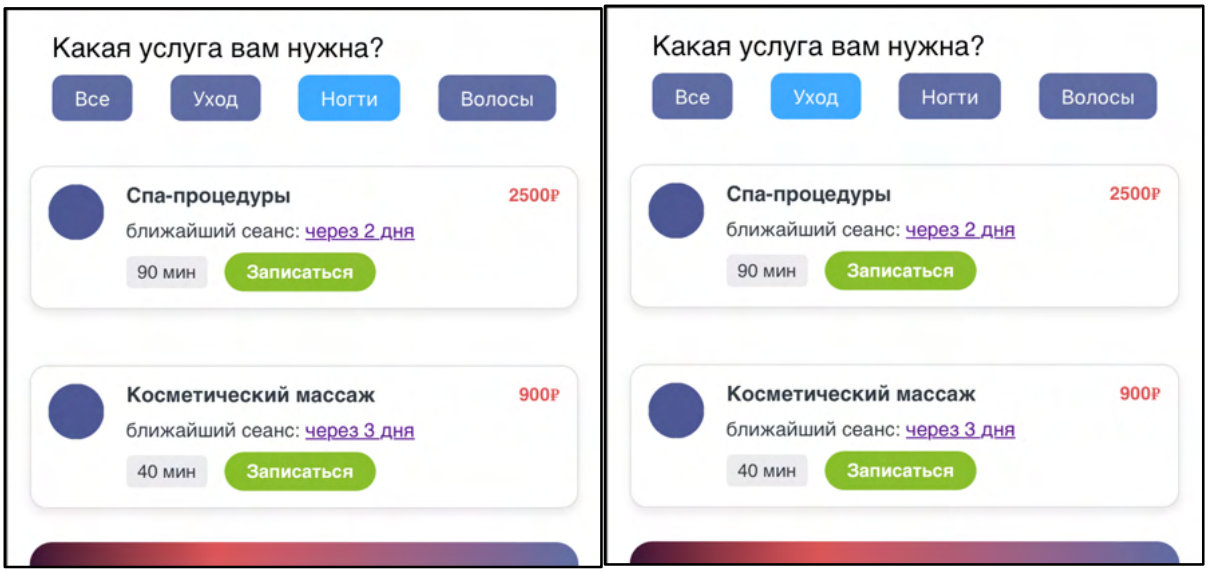


Рисунок 11 – Демонстрация интерфейса фильтрации услуг

Для дизайна домашних страницы клиента и мастера использовались самостоятельно разработанные компоненты, однако, для отображения стандартных форм применялись компоненты библиотеки Ant Design [27]. В них же и задаются правила валидация полей, в листинге 6 приведен фрагмент кода, описывающий поле ввода email на странице регистрации.

Листинг 6 – Код с полем с валидацией email

```
<Form.Item name="email"
    rules={[{
        required: true,
        message: "Пожалуйста, введите вашу почту!",
        type: "email",
      }]}>
  <Input
        prefix={<MailOutlined className="site-form-item-icon" />}
        placeholder="Ваша почта"/>
</Form.Item>
```

На рисунке 12 показан пример валидации поля email в форме регистрации на платформе.

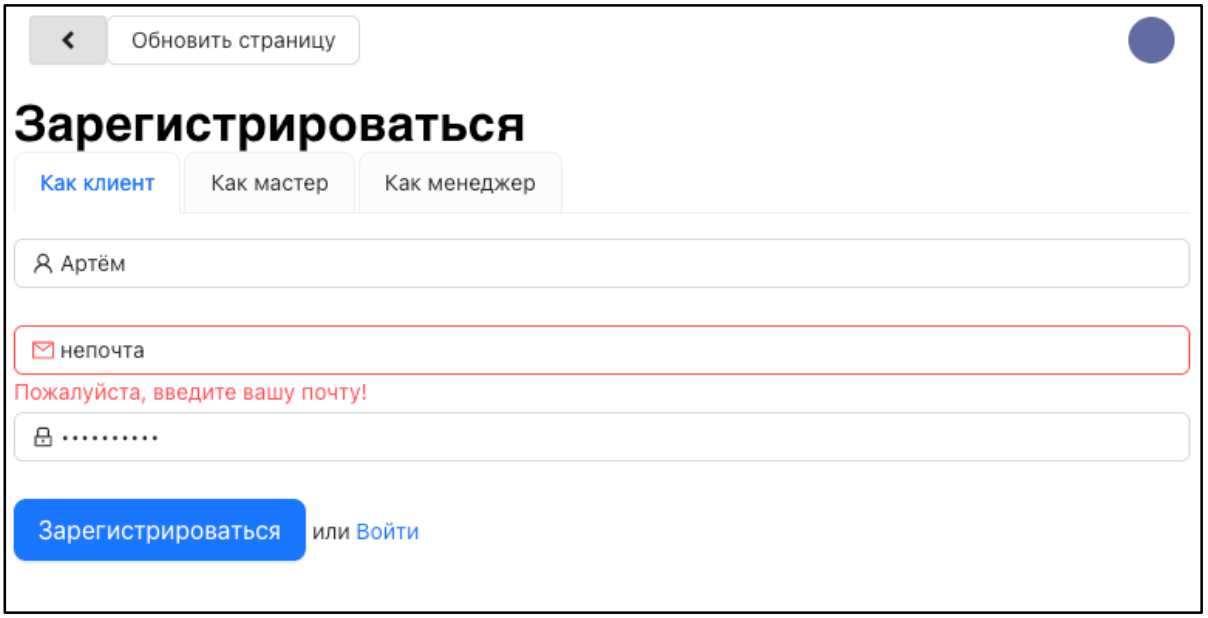


Рисунок 12 – Скриншот формы регистрации

### 4.2.3. Реализация дополнительных сервисов

Для общения между API мессенджера и мини-приложением был создан notifications-worker, в его структуре можно выделить Телеграм бота, который подключается к ранее созданному в мессенджере через BotFather боту, который будет обрабатывать сообщения пользователей, события в мини-приложении, а также высылать сообщения-напоминания.

Бот, как и сервер, написан на Node.js и обернут в контейнер и запускается в общей конфигурации. Подключение к базе данных у бота возможно только в режиме чтения.

В качестве альтернативного способа рассылки уведомлений была выбрана электронная почта клиентов. Для этого в «index.js» сервиса `notification_worker` конфигурируется, в тестовых целях, подключение к внешнему, относительно разрабатываемой системы, API почтового провайдера «smtp.mail.ru». У него существуют ограничения на отправку уведомлений по количеству в промежуток времени.

Чтобы производить проверку и рассылку уведомлений с заданной частотой, необходимо создание планировщика. С этой целью используется метод `scheduleJob()` пакета `node-schedule`. В него необходимо передать два параметра:

1) строка символов, описывающая паттерн, по которому формируется расписание;

2) метод, вызываемый по заданному расписанию.

Его логика реализована по следующему принципу: через заданный интервал бот запрашивает уведомления, со статусом «pending», срок срабатывания которых, назначен до настоящего момента времени, сохраняет их идентификаторы и высылает адресатам. Затем notification-worker формирует запрос к серверному приложению, чтобы изменить статус отправленных напоминаний.

### 4.2.4. Тестирование системы

Валидация в сервисе данных происходила по следующим правилам.

1. Описания полей объектов в базе данных содержат соответствующий тип данных, не позволяющий записывать значения иного типа.

2. Проверка выбираемых дат на актуальность и учетных данных на соответствие регулярным выражениям происходит на фронтенде. Когда пользователь вводит некорректные информацию, система уведомляет его об этом, а запрос на сервер не производится. Таким образом уменьшается количество запросов, и, соответственно, снижается нагрузка.

3. Перед тем, как приступить к дальнейшей обработке, сервер проверяет наличие всех нужных параметров в запросе.

Чтобы убедиться, что данные правила выполняются в веб-приложении, были проведены тесты, описанные в таблице 3.

Таблица 3 – Результаты тестирования

| № | Описание теста | Ожидаемый результат | Тест пройден |
|---|---|---|---|
| 1 | Поля аутентификации и авторизации заполняются некорректными данными | Пользователю сообщается об ошибке и выделяется поле с некорректными значениями | Да |
| 2 | Ввод иных типов значения в поле ввода чисел | Поля не принимают ввод, поля остаются пустыми | Да |
| 3 | Формирование в серверном приложении сущностей с отсутствующим обязательными полями | Отказ базы данных в записи переданных данных, объекты в базе не сохраняются | Да |
| 4 | С клиентского приложения к серверу совершается запрос с некорректными значениями обязательных параметров | Сервер не высылает клиентскому приложение сообщение об ошибке, клиенту выдается ошибка о некорректности запроса | Да |

Также необходимо было убедиться, что данные, хранящиеся в основную базу данных, реплицируются. Для этого были созданы записи, которые затем запрашивались у реплицируемого хранилища. Количество таблиц, значений и их содержимое совпадают – тест пройден.

Для локального запуска проекта и его тестирования в интерфейсе мессенджера Телеграм, с помощью утилиты `mkcert` создавались самоподписные SSL-сертификаты.

Все страницы веб-приложения были открыты в различных браузерах на мобильном устройстве и в мини-приложении Телеграм, а также десктопном компьютере и в браузерах Safari версии 16.6, Google Chrome версии 125.0.6422.142, Yandex версии 24.1.1.915.

**Вывод по четвертой главе**

Таким образом были разработаны, интегрированы и протестированы компоненты системы, предназначенной для работы сервиса онлайн-записи на услуги салонов красоты.

# 5. БИЗНЕС-ПРОЕКТИРОВАНИЕ

## 5.1. Описание продукта

«PlanzUp» – стартап-проект по разработке веб-продукта для онлайн-записи на услуги красоты, который предоставит мастерам салонов и их клиентам инструменты планирования, автоматизации записи.

Цель проекта – создание коммерческого программного продукта, с помощью которого можно составлять расписание на удобной как для клиентов, так и мастеров площадке, такие как телеграмм и PWA. Дополнительным преимуществом создания продукта будет возможность расширять свою клиентскую базу представителям бьюти-услуг, а также повышать лояльность уже имеющихся клиентов.

Интернет-сервис поможет найти нужный сеанс удобно и быстро, чтобы клиенты могли сэкономить время и получить качественное обслуживание. Также интернет-сервис облегчит размещение услуг и поможет соединить между собой клиентов и представителей сферы красоты.

Программное обеспечение также предоставит администраторам салонов инструменты по удаленному управлению процессом работой салона и наглядное представление клиентского потока. Платформа будет состоять из веб-сервиса и мини-приложения в Телеграм. Это позволит каждому заинтересованному пользователю управлять записями. Стратегия стартапа концептуально может быть описана в следующих пунктах:

1) интернет-сервис с удобным интерфейсом, который облегчит размещение и поиск услуг;

2) постоянная плата для представителей услуг;

3) представители услуг могут размещать собственный контент на сервисе для привлечения клиентов;

4) конечные клиенты могут оплачивать эксклюзивные услуги на сервисе.

Благодаря «PlanzUp» будет решена проблема фиксирования расписания клиентов и их особенностей. У представителей услуг будет точное расписание записей клиентов. Клиент не будет ждать долго ответа от мастера, так как через календарь можно записаться на услугу напрямую. С помощью интегрируемого календаря клиенты и представители услуг смогут отслеживать и планировать свои записи. В следующем релизе в приложении будет встроенная карта, человек сможет найти ближайшего к нему территориально мастера.

### 5.2. Анализ проблем целевой аудитории

В ходе анализа целевой аудитории были выявлены проблемы, с которыми сталкиваются потенциальные пользователи и которые может помочь решить приложение.

Представители услуг могут забыть записать кого-то или же записать, но забыть про него, что может привести к плохим отзывам и потере лояльности. Для небольших предприятий наем отдельных сотрудников для организации записи приводит к крупным затратам. Также представители услуг могут быть заинтересованы в повышении степени удобства ведения бизнес-процесса по записи клиентов, экономии времени, управляемости своего расписания.

Клиентам важно быстро узнавать каковы цены на услуги, а также когда возможно на них попасть. Это сложно обеспечить при большом потоке клиентов. Проблема накладок в расписании также возможна, она актуальна для занятых людей, особенно, когда нет единого места, где все это учитывается.

Сложности с менеджментом большого объема клиентов, проблема оттока клиентов из-за ясной наглядности при подборе дат для следующих сеансов клиента.

При росте клиентской базы становится сложнее подбирать для них удобные обеим сторонам часы сеансов, ведь у клиентов может быть разный образ жизни.

Отмечается, что у мастеров поток клиентов может быть неравномерным. Приложение может помочь решить и эту проблему.

У мастеров много времени уходит на коммуникации, это неотъемлемая часть работы, однако гораздо удобнее, когда такое общение оптимизировано с помощью автоматизации. А прочие уже рутинные процессы могут отвлекать от работы: на это уходят силы и время, а значит их остается меньше на основную работу.

Эти проблемы актуальны для салонов красоты и мастеров, и они готовы внедрять цифровые продукты для их решения в свои бизнесы. Некоторые из этих сложностей также влияют и на клиентов.

Эти проблемы могут решаться с помощью разработанной платформы следующим образом.

1. Клиентам будут приходить напоминания о сеансах, это снизит возможность того, что они не придут вовремя.
2. Будет решена проблема фиксирования клиентов – теперь у представителей услуг будет точное расписание записей клиентов.
3. Клиент не будет ждать долго ответа от мастера, так как через календарь можно записаться на услугу напрямую.
4. С помощью календаря записей клиенты и представители услуг смогут отслеживать и планировать свои записи.
5. Приложение будет доступно в мини-приложении Телеграм, а значит объединит платформу для коммуникации с сервисом онлайн-записи.

Целевая аудитория проекта – конечные клиенты и представители услуг. Основная проблема двух сегментов – сложность связи спроса и предложения.

Далее обозначены представители услуг, которые могут быть потенциальными пользователями приложения, то есть, те кому необходимо составлять расписания и предлагать их своим клиентам:

1) репетиторы;
2) спортивные клубы;
3) фотостудии;
4) психологи;
5) представители бьюти-услуг.

В качестве первой ниши проекта были выбраны бизнесы в сфере бьюти-услуг, она является одной из самых крупных на рынке, а также в ней очень важны ясная демонстрация перечня услуг, расценок и быстрота коммуникации с клиентами, индивидуальный подход. Работа в мини-приложении в Телеграм, будет преимуществом в данной нише, так как улучшит этот канал связи.

В ходе работы над проектом были проведены опросы обеих целевых аудиторий: как представителей услуг, так и их клиентов. Среди представителей выбранной ниши был проведен опрос 10 респондентов, чтобы выяснить, насколько продукт будет для них полезным. На рисунке 13 представлено соотношение услуг, предоставляемых опрошенными.

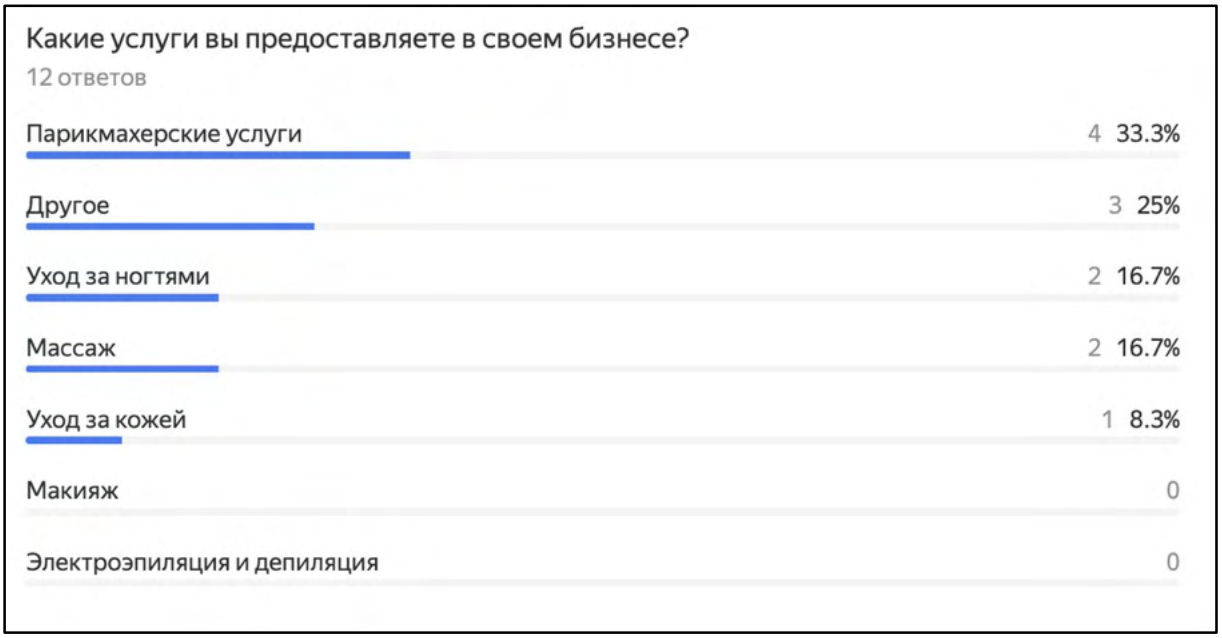


Рисунок 13 – Скриншот соотношения представителей услуг опроса

Выяснилось, что среди них наблюдается большая дифференциация, бизнесы могут оказывать широкий спектр услуг. И, соответственно, есть свои потребности (рисунок 14).

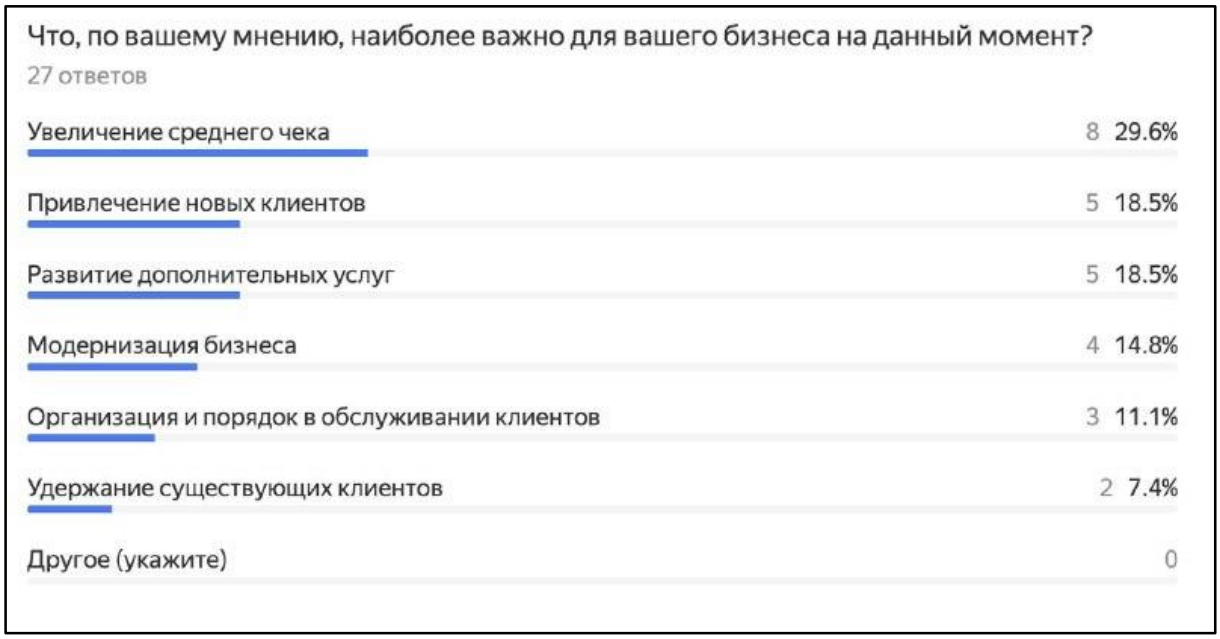


Рисунок 14 – Скриншот результатов вопроса о потребностях

Приложение будет полезно тем, кто хочет привлечь новых клиентов (50% опрошенных) и организовать порядок в обслуживании (30%), применяя цифровые технологии в своей работе (рисунок 15), также такое бизнес-решение сможет поспособствовать увеличению среднего чека, так пользователи смогут ознакомится со всеми услугами на платформе.

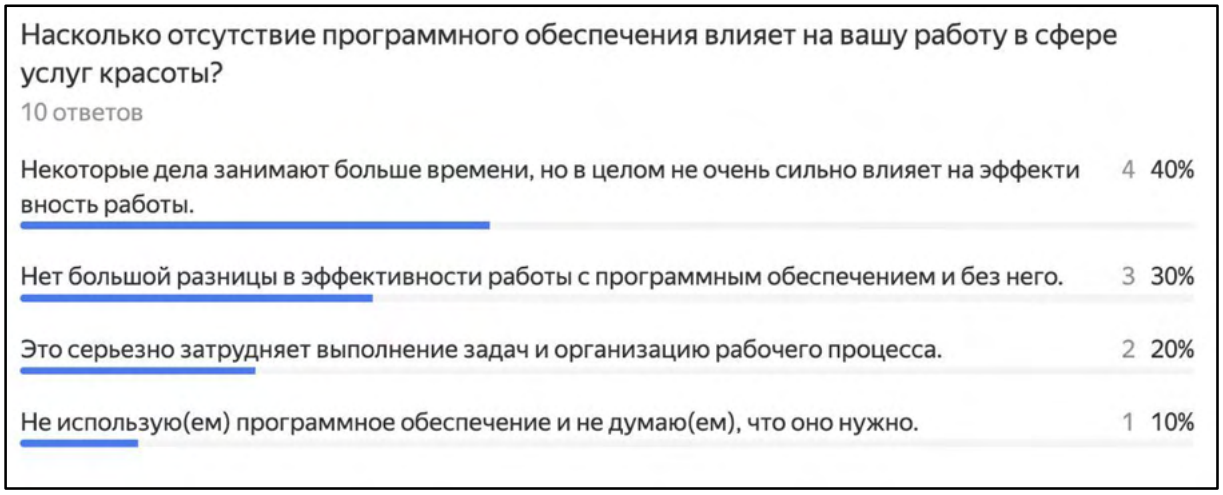


Рисунок 15 – Скриншот результатов анкетирования

Таким образом, второй целевой аудиторией проекта являются, преимущественно, девушки и женщины, как более активные посетители салонов красоты.

Таким образом, программный продукт поможет преодолеть некоторые проблемы представителей индустрии, например, отток посетителей, вызванный организационными ошибками в общении с клиентами. Мастера нередко теряют время на коммуникации с клиентами перед сеансами.

Начинающие предприятия испытывают трудности при поиске своих первых клиентов. Предлагаемое решение также может помочь решить и эту проблему.

### 5.3. Анализ рынка и конкурентов

По данным росстата в России более 11 тысяч организаций с ОКВЭД 96.02 «Предоставление услуг парикмахерскими и салонами красоты» [24]. По данным карт 2ГИС на территории Челябинска расположено 3623 места по категории красота, лишь у 891 есть сайт, а расценки указаны у 1751.

Значит, у продукта на домашнем рынке уже есть возможность найти потенциальных клиентов. По данным опроса более 30 процентов ищут цифровое решение для организации онлайн-записи и упорядочивания клиентского потока (рисунок 16).

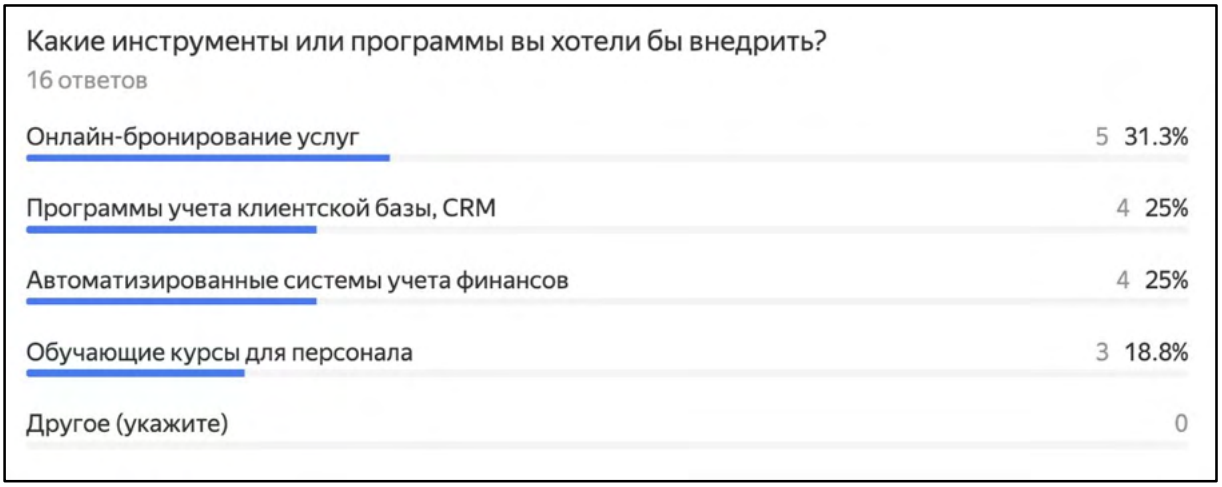


Рисунок 16 – Предпочтения опрошенных по внедрению

Если брать нишу заведений, не указавших сайт на картах, то 30 процентов от нее составит 819 заведений, на этом и будет основываться оценка реально достижимого рынка.

На российском рынке существуют решения, предоставляющие онлайн-запись и, поэтому для выделения на их фоне необходимо определить ключевые отличительные черты и преимущества продукта. К таким сервисам относятся Avito, Yclients, Dikidi, Profi.ru. Их сравнительная характеристика с разрабатываемым продуктом PlanzUp представлена в таблице 3.

Таблица 3 – Сравнительная характеристика конкурентов

| **Категория сравнения** | **PlanzUp** | **Yclients** | **Avito** | **Profi.ru** | **Dikidi** |
|---|---|---|---|---|---|
| Чат-бот | + | + | + | – | – |
| Поиск ближайшего мастера | В разработке | + | – | – | + |
| Мини-приложение | Телеграм | ВК | – | – | – |
| Календарь | + | – | – | – | – |

Сравнительный анализ показал, что «идеальное» приложение должно обладать, всеми указанными в таблице 3 характеристиками, они были учтены при разработке продукта.

### 5.4. Бизнес-модель

В качестве бизнес-модели выбран маркетплейс на платформе будут размещаться различные представители сферы бьюти-услуг, а также они смогут размещать на платформе статьи с информацией о салоне, его основных преимуществах.

Была выбрана смешанная бизнес-модель. Доход планируется от постоянной подписки, фиксированной цены за приведенного клиента, дополнительных услуг, рекламы внутри сервиса.

Каналы продвижения: карточки организации в браузере, социальные сети.

Каналы сбыта: прямые продажи, продажа своих услуг компаниям, сарафанное радио, цепочка рекомендаций и отзывов.

В качестве стратегии распространения продукта нередко используется бесплатный пробный период, однако, можно выделить несколько альтернатив такой стратегии.

Предоставление бесплатных токенов на использование, то есть клиенту предоставляется ограниченной количество пробных попыток на использование сервиса. Так, например, в случае онлайн-записи можно ограничить для салона количество самих записей, это и будут токены сервиса. Представители услуг смогут наглядно ознакомиться с сервисом, протестировать его для своего бизнеса.

Также можно предоставлять в бесплатной версии сервиса можно дать лишь один элемент функционала, а остальные распространять по подписке. Такими услугами могут быть размещение сервиса на платформе, предоставление им возможности публиковать статьи, но без онлайн-записи и помощи в создании контента.

Также стоит обратить внимание на вариант пробного периода с предоплатой или с указанием счета или платы за каждое использование привилегированных сервисов как для конечных клиентов, так и для салонов.

После анализа целевой аудитории, опросов и интервью были выявлено несколько потенциальных каналов связи с клиентами.

Прямые продажи, могут оказаться хоть и затратным, по причине необходимости найма специальных сотрудников, но эффективным каналом продаж так, как в данной нише большое значение имеет сарафанное радио. Согласно опросам, этот канал считается одним из самых эффективных и надежных в данной нише – 40 процентов мастеров считают его наиболее полезным (рисунок 17). Таким образом прямые продажи могут запустить цепочку рекомендаций и отзывов.


Рисунок 17 – Предпочтения опрошенных по каналам продвижения

Также стоит обратить внимание на тематические форумы, на которых концентрируется множество мастеров и представителей салонов красоты. И на них бизнесы могут представить свой продукты целевой аудитории.

Коммуникации через социальные сети в том числе являются возможным источников новых клиентов, более того специфика продукта и поддержка мини-приложений может стать преимуществом при таком способе продаж.

Продажа своих услуг компаниям – можно на платформе размещать информационные статьи от представителей услуг, что позволит рекламировать их у нас в интернет-сервисе.

### 5.5. Экономика стартапа

Для определения реализуемости проекта рассчитаем показатели эффективности. Проект предполагает две стадии инвестиционную и реализационную. Инвестиционный этап включает в себя разработку приложения и проведение НИОКР. На данном этапе требуется один специалист разработчик, который выполнит все эти предварительные работы. Таким образом инвестиционные затраты включают в себя заработную плату и страховые взносы, которые составляют 127 тысяч рублей.

Исходя из бизнес-модели проекта, в качестве элемента расчета финансовой стратегии были выбраны салоны красоты и индивидуальные мастера, именно они будут платить подписку за сервис, следовательно, подсчет дохода необходимо производить исходя из количества таких клиентов и их LTV, метрики отражающей пожизненную ценность клиента, а расходы в терминах стоимости привлечения клиента (CAC).

Выручка планируется за счет подписки в размере 2000 рублей, в среднем, в зависимости от размера предприятия. Дополнительную прибыль можно получить от подписок, которые будут оплачивать сами салоны для своих специальных клиентов. Реклама внутри сервиса в виде статей также принесет дополнительный доход. Финансовая модель стартапа учитывает затраты на облачный хостинг и фонд оплаты труда. Маркетинговые расходы включают рекламу в Telegram и Яндексе. Оборудование будет учитываться для предотвращения проблем, связанных с неожиданной поломкой устройств.

Этап реализации проекта начинается с момента запуска приложения и включает маркетинговые предприятия для продвижения платформы. Предполагаемые расходы на них представлены в таблице 4.

Таблица 4 – Маркетинговые мероприятия

| **Маркетинговое мероприятие** | **Бюджет на рекламу в рублях** |
|---|---|
| Прямые продажи | 200 000 |
| Участие в форуме | 150 000 |
| Рассылки по данным с открытых источников | 100 000 |
| Лендинг | 100 000 |
| Контекстная реклама | 100 000 |

Текущие затраты за месяц на этапе реализации проекта представлены в таблице 5.

Таблица 5 – Текущие затраты по прямым продажам

| **Статьи затрат** | **Сумма в рублях** |
|---|---|
| Хостинг приложения | 7 000 |
| Расходы на команду | 100 000 |
| Маркетинговые расходы | 200 000 |
| Итого | 307 000 |

Планируемые расходы и доходы по проекту представлены помесячно на рисунке 18.

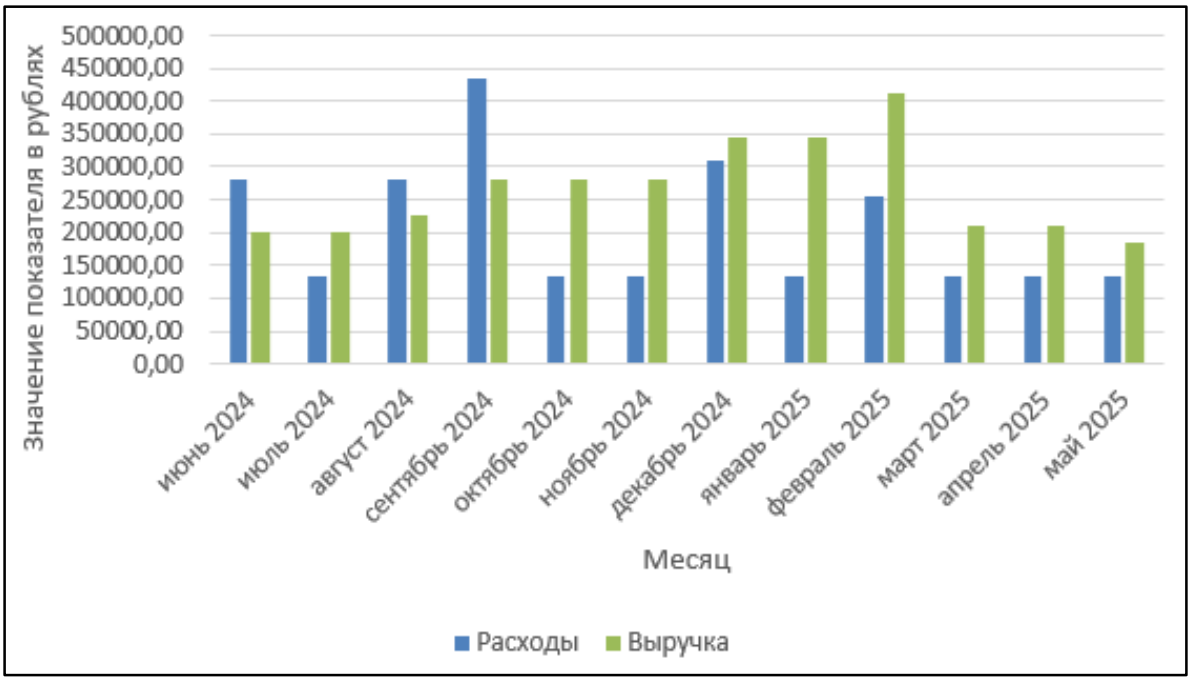


Рисунок 18 – Предпочтения опрошенных по каналам продвижения

По мере реализации проекта видно, что выручка начинает превышать затраты. Итоговый расчет показателей реализации проекта приведен в таблицах 1–3 приложения Б. На его основе рассчитаны показатели эффективности (таблица 6).

Таблица 6 – Расчетные показатели эффективности проекта

| **Наименование показателя** | **Значение показателя** |
|---|---|
| Ставка дисконтирования | 24% |
| NPV (чистый дисконтированный доход) | 293 438,80 рублей |
| Срок проекта | 12 месяцев |
| Дисконтированный срок окупаемости | Около 10 месяцев |
| IRR (внутренняя норма доходности) | 26% |

**Вывод по пятой главе**

Чистый дисконтированный доход положительный, внутренняя норма доходности меньше ставки дисконтирования, дисконтированный срок окупаемости меньше срока реализации проекта. Это говорит об эффективности проекта на стадии его планирования. Таким образом, проведенные расчеты показали целесообразность реализации проекта.

## ЗАКЛЮЧЕНИЕ

В рамках данной работы был разработан веб-сервис, который через веб-интерфейс позволяет записываться онлайн.

1. Произведен обзор и анализ предметной области и существующих аналогов.
2. Спроектировано веб-приложения.
3. Разработана архитектура сервиса онлайн-записи.
4. Реализовано веб-приложение и произведены тесты системы.
5. Осуществлено бизнес-проектирование сервиса онлайн-записи.

В ходе работы были освоены принципы разработки приложений на React и Node.js, технологии построения мультиконтейнерных приложений, упрощающие процесс развертывания. Были задействован модульный подход, позволяющий в дальнейшем масштабировать и модернизировать компоненты, как в связке, так и по отдельности.

В процессе выполнения выпускной квалификационной работы в формате «Стартап как диплом» был проведен анализ рынка и целевой аудитории, определены основные конкурентные достоинства будущего продукта, спланированы доходы и расходы проекта. Проведенный анализ эффективности проекта показал целесообразность внедрения программного продукта для целевой аудитории.

**ЛИТЕРАТУРА**

1. Yclients. [Электронный ресурс] URL: https://www.yclients.com (дата обращения: 09.02.2024 г.).

2. Sonline. [Электронный ресурс] URL: https://sonline.su (дата обращения: 11.02.2024 г.).

3. Руководство по языку программирования JavaScript. [Электронный ресурс] URL: https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/Introduction (дата обращения: 16.02.2024 г.).

4. Desktop Browser Market Share Worldwide | StatCounter Global Stats. [Электронный ресурс] URL: http://gs.statcounter.com/browser-market-share/desktop/worldwide (дата обращения: 16.02.2024 г.).

5. Изучаем PostgreSQL. [Электронный ресурс] URL: https://habr.com/ru/companies/otus/articles/706346/ (дата обращения: 17.02.2024 г.).

6. Учебное пособие «Диаграмма компонентов» | Полное руководство с примерами. [Электронный ресурс] URL: https://creately.com/blog/ru/uncategorized-ru/учебное-пособие-по-компонентной-диаг/ (дата обращения: 20.02.2024 г.).

7. Библиотека для создания Телеграм-ботов. [Электронный ресурс] URL: https://github.com/rubenlagus/TelegramBots (дата обращения: 27.02.2024 г.).

8. Руководство по React. [Электронный ресурс] URL: https://metanit.com/web/react/ (дата обращения: 28.02.2024 г.).

9. Маликов А.В. Ориентированные графы в реляционных базах данных. // Доклады ТУСУР, 2008. – № 2(18). – Ч. 2. – С. 100–104.

10. Введение в UML от создателей языка. Гради Буч, Джеймс Рамбо, Ивар Якобсон. [Электронный ресурс] URL: https://www.labirint.ru/books/274454/ (дата обращения: 01.03.2024 г.).

11. Документация к Docker. [Электронный ресурс] URL: https://docs.docker.com/get-started/overview/ (дата обращения: 09.03.2024 г.).

12. Документация к образу базы данных PostgreSQL в Docker. [Электронный ресурс] URL: https://hub.docker.com/_/postgres (дата обращения: 12.03.2024 г.).

13. Руковдоство по Node.js. [Электронный ресурс] URL: https://nodejs.org/en/learn/getting-started/introduction-to-nodejs (дата обращения: 12.03.2024 г.).

14. Официальный сайт Ruby on Rails. [Электронный ресурс] URL: https://rubyonrails.org (дата обращения: 12.03.2024 г.).

15. Официальный сайт Django. [Электронный ресурс] URL: https://www.djangoproject.com/ (дата обращения: 12.03.2024 г.).

16. Официальный сайт Flask. [Электронный ресурс] URL: https://flask.palletsprojects.com/ (дата обращения: 12.03.2024 г.).

17. Официальный сайт Spring. [Электронный ресурс] URL: https://spring.io/ (дата обращения: 12.03.2024 г.).

18. Руководство по Express.js. [Электронный ресурс] URL: https://expressjs.com/ (дата обращения: 12.03.2024 г.).

19. Express Tutorial Part 4: Routes and controllers. [Электронный ресурс] URL: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes/ (дата обращения: 10.04.2024 г.).

20. Getting Started Axios Docs. [Электронный ресурс] URL: https://axios-http.com/docs/intro/ (дата обращения: 11.04.2024 г.).

21. Документация к NGINX. [Электронный ресурс] URL: https://nginx.org/ru/docs/ (дата обращения: 12.04.2024 г.).

22. Описание технологии Reverse Proxy. [Электронный ресурс] URL: https://ddos-guard.net/ru/technologies/reverse-proxy (дата обращения: 12.04.2024 г.).

23. Документация по GitHub. [Электронный ресурс] URL: https://docs.github.com/en (дата обращения: 01.05.2024 г.).

24. Что такое коды ОКВЭД и как их выбрать. [Электронный ресурс] URL: https://www.sberbank.ru/ru/s_m_business/pro_business/chto-takoe-kody-okved-i-kak-ih-vybrat/ (дата обращения: 02.05.2024 г.).

25. Docker Compose overview. [Электронный ресурс] URL: https://docs.docker.com/compose/ (дата обращения: 03.05.2024 г.).

26. Руководство Built-in React Hooks. [Электронный ресурс] URL: https://react.dev/reference/react/hooks (дата обращения: 03.05.2024 г.).

27. Ant Design – Components Overview. [Электронный ресурс] URL: https://ant.design/components/overview (дата обращения: 05.05.2024 г.).

28. Нотации модели сущность-связь (ER диаграммы). [Электронный ресурс] URL: https://pro-prof.com/archives/8126 (дата обращения: 05.05.2024 г.).

# ПРИЛОЖЕНИЯ

## Приложение А. Исходные коды программ

Листинг 1 – Код конфигурационного compose-файла Docker

```
version: "3.7"
x-postgres-common:
  &postgres-common
  image: postgres:14-alpine
  user: postgres
  restart: always
  healthcheck:
    test: 'pg_isready -U user --dbname=postgres'
    interval: 10s
    timeout: 5s
    retries: 5

services:
  postgres_primary:
    <<: *postgres-common
    ports:
      - 5432:5432
    environment:
      POSTGRES_USER: user
      POSTGRES_DB: postgres
      POSTGRES_PASSWORD: postgres_password
      POSTGRES_HOST_AUTH_METHOD: "scram-sha-256\nhost replication all
0.0.0.0/0 md5"
      POSTGRES_INITDB_ARGS: "--auth-host=scram-sha-256"
    command: |
      postgres
      -c wal_level=replica
      -c hot_standby=on
      -c max_wal_senders=10
      -c max_replication_slots=10
      -c hot_standby_feedback=on
    volumes:
      - ./00_init.sql:/docker-entrypoint-initdb.d/00_init.sql
      - postgres_primary_data:/var/lib/postgresql/data

  postgres_replica:
    <<: *postgres-common
    ports:
      - 5433:5432
    environment:
      PGDATA: /var/lib/postgresql/data/dbfiles_
      PGUSER: replicator
      PGPASSWORD: replicator_password
    command: |
      bash -c "
      if [ ! -s /var/lib/postgresql/data/dbfiles_/PG_VERSION ]; then
        rm -rf /var/lib/postgresql/data/dbfiles_/*
        until pg_basebackup --pgdata=/var/lib/postgresql/data/dbfiles_ -R -
-slot=replication_slot --host=postgres_primary --port=5432
        do
        echo 'Waiting for primary to connect...'
        sleep 1s
        done
        echo 'Backup done, starting replica...'
        chmod 0700 /var/lib/postgresql/data/dbfiles_
      fi
      postgres
```

```
    "
  depends_on:
    - postgres_primary
  volumes:
    - postgres_replica_data:/var/lib/postgresql/data

nginx:
  depends_on:
    - api
    - client
    - notifications
  restart: always
  build:
    dockerfile: Dockerfile.dev
    context: ./nginx
  ports:
    - "3052:80"
api:
  build:
    dockerfile: Dockerfile.dev
    context: "./server"
  volumes:
    - /app/node_modules
    - ./server:/app
  environment:
    - PGUSER=user
    - PGHOST=postgres_primary
    - PGDATABASE=postgres
    - PGPASSWORD=postgres_password
    - PGPORT=5432
    - TGBOTTOKEN=1234567
  depends_on:
    postgres_primary:
      condition: service_healthy
  ports:
    - "5007:5007" # Expose the API port

notifications:
  build:
    dockerfile: Dockerfile.dev
    context: "./notifications-worker"
  volumes:
    - /app/node_modules
    - ./notifications-worker:/app
  environment:
    - PGUSER=replicator
    - PGHOST=postgres_replica
    - PGDATABASE=postgres
    - PGPASSWORD=replicator_password
    - PGPORT=5433
    - TGBOTTOKEN=1234567
  depends_on:
    postgres_replica:
      condition: service_healthy
  ports:
    - "3002:3002"

bot:
  build:
    dockerfile: Dockerfile.dev
    context: "./bot"
  volumes:
```

```
      - /app/node_modules
      - ./bot:/app
    environment:
      - PGUSER=replicator
      - PGHOST=postgres_replica
      - PGDATABASE=postgres
      - PGPASSWORD=replicator_password
      - PGPORT=5433
      - TGBOTTOKEN=1234567

  client:
    stdin_open: true
    environment:
      - CHOKIDAR_USEPOLLING=true
    build:
      dockerfile: Dockerfile.dev
      context: ./client
    volumes:
      - /app/node_modules
      - ./client:/app

volumes:
  postgres_primary_data:
  postgres_replica_data:
```

Листинг 2 – Код метода обработки запроса получения расписания

```
exports.getSchedule = (req, res) => {
  if (!req.body || !req.body.userId) {
    return res.status(401).redirect("/login").send({
      message: "Идентификатор пользователя не предотавлен",
    });
  }
  const isOldSchedule = req?.body?.old;
  const userType = req.params.userType;
  const dateOperator = Op.gte;
  if (isOldSchedule) {
    dateOperator = Op.lt;
  }
  if (!userType) {
    return res.status(400).send({
      message: "Тип пользователя не предоставлен",
    });
  }
  const model =
    userType === "client"
      ? Client
      : userType === "master"
      ? Master
      : userType === "manager"
      ? Manager
      : userType === "admin"
      ? Admin
      : null;

  if (!model) {
    return res.status(404).send({
      message: "Некорректный запрос к серверу",
    });
  }
```

```
  if (userType === "client" || userType === "master") {
    Settings.findOne({
      where: { UID: req.body.userId },
      include: {
        model: model,
        as: "user",
        include: {
          model: Schedule,
          as: "schedule",
          include: {
            model: Appointment,
            as: "appointments",
            include: {
              model: Service,
              as: "service",
            },
            where: {
              start: { [dateOperator]: new Date() },
            },
          },
          include: {
            model: Period,
            as: "periods",
            where: {
              start: { [dateOperator]: new Date() },
            },
          },
        },
      },
    })
      .then((settings) => {
        return res.status(200).send(settings.user);
      })
      .catch((err) => {
        console.log(
          "getAppointments - ошибка при получении записей ${userType}",
          err
        );
        return res
          .status(500)
          .send({
            message: "При получении записей произошла ошибка, попробуйте
позже",
          });
      });
  } else {
    Settings.findOne({
      where: { UID: req.body.userId },
      include: {
        model: model,
        as: "user",
        include: {
          model: Salon,
          as: "salon",
          include: {
            model: Master,
            as: "masters",
            include: {
              model: Schedule,
              as: "schedule",
              include: {
                model: Appointment,
```

Окончание листинга 2 приложения А

```
                as: "appointments",
                where: {
                  start: { [dateOperator]: new Date() },
                },
                iclude: {
                  model: Client,
                  as: "client",
                  include: {
                    model: Clientcard,
                    as: "clientcard",
                  },
                },
              },

              include: {
                model: Period,
                as: "periods",
                where: {
                  start: { [dateOperator]: new Date() },
                },
              },
            },
          },
        },
      },
    })
      .then((settings) => {
        return res.status(200).send(settings.user);
      })
      .catch((err) => {
        console.log(
          "getAppointments - ошибка при получении записей ${userType}",
          err
        );
        return res
          .status(500)
          .send({
            message: "При получении записей произошла ошибка, попробуйте
позже",
          });
      });
  }
};
```

Листинг 3 – Код метода обращения к серверу в клиентской части

```
import { useState } from "react";
import axios from "axios";
import { message } from "antd";
import authHeader from "../services/auth-header";
const useApi = () => {
  const [data, setData] = useState([]);
  const [error, setError] = useState(null);
  const [loading, setLoading] = useState(false);
  const [messageApi, contextHolder] = message.useMessage();
  const fetchData = async (endpoint, options = {}) => {
    setLoading(true);
    try {
      const userId = JSON.parse(localStorage.getItem("user")).UID;
      console.log("userId: " + userId);
```

```
      const response = await axios({
        ...options,
        url: endpoint,
        headers: {
          // передача JWT-токена
          ...authHeader(),
          "Content-Type": "application/json",
        },
        params: {
          ...options.params,
        },
      });
      setData(response.data);
      setError(null);
    } catch (err) {
      setData(null);
      if (err.response?.status === 401 && !error) {
        messageApi.open({
          type: "error",
          content: "Сессия истекла. Войдите заново.",
          duration: 10,
          onClose: () => {
            window.location.reload();
          },
        });
      }
      if (err.response?.status === 403 && !error) {
        messageApi.open({
          type: "error",
          content: err?.message || "Доступ запрещен.",
          duration: 10,
        });
      }
      if (err.response?.status === 500 && !error) {
        messageApi.open({
          type: "error",
          content: err?.message || "Сервер не отвечает. Попробуйте позже.",
          duration: 10,
        });
      }
      if (err.response?.status === 404 && !error) {
        messageApi.open({
          type: "error",
          content: "404 - некоректное обращение к серверу",
          duration: 10,
        });
      }
      setError(err);
    } finally {
      setLoading(false);
    }
  };

  return { data, error, loading, fetchData, contextHolder };
};

export default useApi
```

## Приложение Б. Расчеты показателей эффективности проекта

В таблицах 1–3 представлены расчеты показателей реализации проекта периодами по 4 месяца в рублях.

Таблица 1 – Расчеты показателей реализации на 1 период

| Наименование показателя | Июнь 2024 | Июль 2024 | Август 2024 | Сентябрь 2024 |
|---|---|---|---|---|
| Расходы | 279400,00 | 134400,00 | 279400,00 | 434400,00 |
| Выручка | 200000,00 | 200000,00 | 225000,00 | 281000,00 |
| Прибыль | -79400,00 | 65600,00 | -54400,00 | -153400,00 |
| Чистая прибыль | -79400,00 | 52480,00 | -54400,00 | -153400,00 |
| Чистая прибыль с нарастающим итогом | -79400,00 | -26920,00 | -81320,00 | -234720,00 |
| Дисконтированный денежный поток | -77 843,14 | 50 442,14 | -51 262,33 | -141 717,89 |
| Дисконтированный денежный поток с нарастающим итогом | -177 843,14 | -127 401,00 | -178 663,33 | -320 381,22 |

Таблица 2 – Расчеты показателей реализации на 2 период

| Наименование показателя | Октябрь 2024 | Ноябрь 2024 | Декабрь 2024 | Январь 2025 |
|---|---|---|---|---|
| Расходы | 134400,00 | 134400,00 | 309400,00 | 134400,00 |
| Выручка | 281000,00 | 281000,00 | 343500,00 | 343500,00 |
| Прибыль | 146600,00 | 146600,00 | 34100,00 | 209100,00 |
| Чистая прибыль | 117280,00 | 117280,00 | 27280,00 | 167280,00 |
| Чистая прибыль с нарастающим итогом | -117440,00 | -160,00 | 27120,00 | 194400,00 |
| Дисконтированный денежный поток | 106 224,11 | 104 141,28 | 23 748,88 | 142 771,87 |
| Дисконтированный денежный поток с нарастающим итогом | -214 157,11 | -110 015,83 | -86 266,95 | 56 504,92 |

Таблица 3 – Расчеты показателей реализации на 3 период

| Наименование показателя | Февраль 2025 | Март 2025 | Апрель 2025 | Май 2025 |
|---|---|---|---|---|
| Расходы | 279400,00 | 254400,00 | 134400,00 | 134400,00 |
| Выручка | 200000,00 | 411000,00 | 211000,00 | 211000,00 |
| Прибыль | -79400,00 | 156600,00 | 76600,00 | 76600,00 |
| Чистая прибыль | -79400,00 | 125280,00 | 61280,00 | 61280,00 |
| Чистая прибыль с нарастающим итогом | -79400,00 | 319680,00 | 380960,00 | 442240,00 |

Окончание таблицы 3 приложения Б

| **Наименование показателя** | **Февраль 2025** | **Март 2025** | **Апрель 2025** | **Май 2025** |
|---|---|---|---|---|
| Дисконтированный денежный поток | -77 843,14 | 104 828,70 | 50 270,94 | 49 285,24 |
| Дисконтированный денежный поток с нарастающим итогом | -177 843,14 | 161 333,62 | 211 604,56 | 260 889,80 |

## Приложение В. Схема базы данных

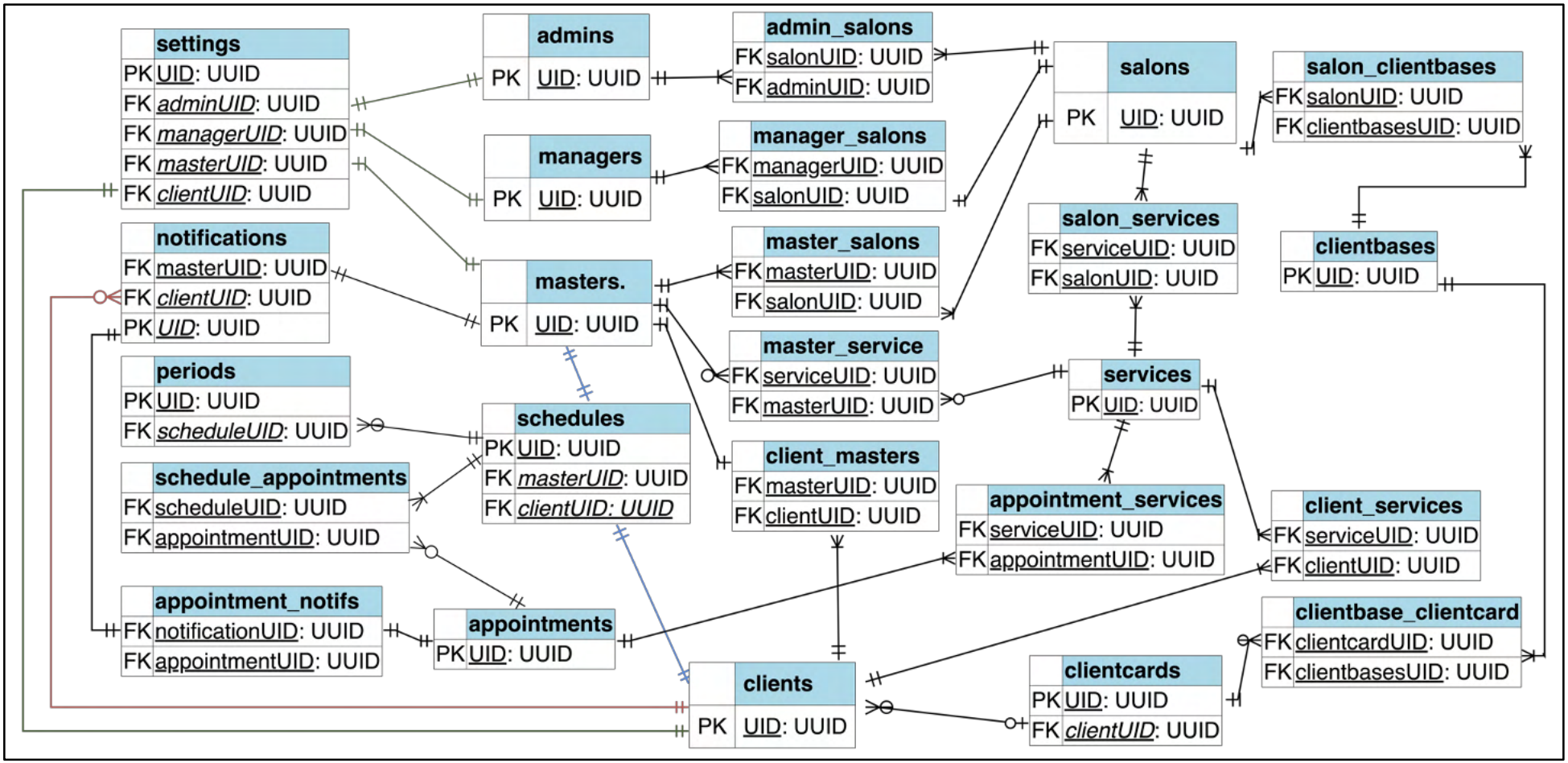


Рисунок 1 – Схема базы данных