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:**

Веб-приложение «Сервис для предоставления услуг»

Выполнил студент Хатченок Дмитрий Николаевич

(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

Минск 2025

# Содержание

[Содержание 2](#_Toc199320120)

[Введение 5](#_Toc199320121)

[1 Постановка задачи и обзор аналогичных решений 6](#_Toc199320122)

[1.1 Постановка задачи 6](#_Toc199320123)

[1.2 Обзор аналогичных решений 6](#_Toc199320124)

[1.2.1 Аналог «Onliner» 6](#_Toc199320125)

[1.2.2 Аналог «YouDo» 7](#_Toc199320126)

[1.2.2 Аналог «Профи.ру» 7](#_Toc199320127)

[1.3 Выводы по главе 8](#_Toc199320128)

[2 Проектирование веб-приложения 9](#_Toc199320129)

[2.1 Функциональность веб-приложения 9](#_Toc199320130)

[2.2 Логическая схема базы данных 10](#_Toc199320131)

[2.3 Архитектура веб-приложения 14](#_Toc199320132)

[2.4 Выводы по главе 15](#_Toc199320133)

[3 Реализация веб-приложения 16](#_Toc199320134)

[3.1 Программная платформа 16](#_Toc199320135)

[3.2 Система управления базами данных PostgreSQL 16](#_Toc199320136)

[3.3 Object-Relational Mapping Prisma 16](#_Toc199320137)

[3.4 Библиотеки и фреймворки 17](#_Toc199320138)

[3.5 Структура серверной части 18](#_Toc199320139)

[3.6 Реализация функций 21](#_Toc199320140)

[3.6.1 Регистрация 22](#_Toc199320141)

[3.6.2 Аутентификация 23](#_Toc199320142)

[3.6.3 Просмотр, поиск и фильтрация услуг 23](#_Toc199320143)

[3.6.4 Просмотр информации об услуге 24](#_Toc199320144)

[3.6.5 Добавление услуги в избранное 25](#_Toc199320145)

[3.6.6 Удаление услуги из избранного 26](#_Toc199320146)

[3.6.7 Добавление услуги 26](#_Toc199320147)

[3.6.8 Изменение услуги 27](#_Toc199320148)

[3.6.9 Удаление услуги 27](#_Toc199320149)

[3.6.10 Общение с представителем услуги в чате 28](#_Toc199320150)

[3.6.11 Бронирование услуги 29](#_Toc199320151)

[3.6.12 Просмотр истории бронирований 30](#_Toc199320152)

[3.6.13 Просмотр бронирований своей услуги 31](#_Toc199320153)

[3.6.14 Изменение статуса бронирования 32](#_Toc199320154)

[3.6.15 Оплата выполненной услуги 33](#_Toc199320155)

[3.6.16 Добавление отзыва на услугу 34](#_Toc199320156)

[3.6.17 Удаление отзыва на услугу 35](#_Toc199320157)

[3.6.18 Изменение информации о пользователе 35](#_Toc199320158)

[3.6.19 Просмотра списка пользователей 36](#_Toc199320159)

[3.6.20 Блокирование и разблокирование пользователя 37](#_Toc199320160)

[3.6.21 Удаление пользователя 37](#_Toc199320161)

[3.7 Структура клиентской части 38](#_Toc199320162)

[3.8 Использование Docker 39](#_Toc199320163)

[3.9 Использование Nginx 42](#_Toc199320164)

[3.12 Выводы по разделу 43](#_Toc199320165)

[4 Тестирование веб-приложения 44](#_Toc199320166)

[4.1 Функциональное тестирование 44](#_Toc199320167)

[4.2 Выводы по разделу 49](#_Toc199320168)

[5 Руководство пользователя 50](#_Toc199320169)

[5.1 Регистрация 50](#_Toc199320170)

[5.2 Аутентификация 50](#_Toc199320171)

[5.3 Просмотр, поиск и фильтрация услуг 51](#_Toc199320172)

[5.4 Просмотр информации об услуге 51](#_Toc199320173)

[5.5 Добавление услуги в избранное 52](#_Toc199320174)

[5.6 Удаление услуги из избранного 52](#_Toc199320175)

[5.7 Добавление услуги 53](#_Toc199320176)

[5.8 Изменение услуги 54](#_Toc199320177)

[5.9 Удаление услуги 55](#_Toc199320178)

[5.10 Общение с представителем услуги в чате 56](#_Toc199320179)

[5.11 Бронирование услуги 56](#_Toc199320180)

[5.12 Просмотр истории личных бронирований 57](#_Toc199320181)

[5.13 Просмотр бронирований своей услуги 57](#_Toc199320182)

[5.14 Изменение статуса бронирования 58](#_Toc199320183)

[5.15 Оплата выполненной услуги 59](#_Toc199320184)

[5.16 Добавление и удаление отзывов на услугу 59](#_Toc199320185)

[5.17 Профиль 60](#_Toc199320186)

[Заключение 62](#_Toc199320187)

[Приложение А 65](#_Toc199320188)

[Приложение Б 66](#_Toc199320189)

[Приложение В 67](#_Toc199320190)

[Приложение Г 68](#_Toc199320191)

[Приложение Д 70](#_Toc199320192)

# Введение

Сервисы для предоставления услуг — будь то ремонт, репетиторство или любая другая сфера — не только упрощают поиск квалифицированных специалистов, но и способствуют более эффективной организации повседневных задач. Также такие сервисы помогают лучше ориентироваться в ценовом сегменте рынка, что позволяет найти наиболее выгодное предложение от высококвалифицированного специалиста. Однако при самостоятельном поиске и выборе исполнителя можно упустить важные детали, влияющие на качество и надежность результата. Также существует необходимость обычному человеку иметь возможность заработать, занимаясь своим любимым делом.

Целью данного курсового проектирования является создание удобного и функционального веб-приложения для предоставления и бронирования различных услуг. Сервис позволит пользователям не только находить исполнителей в интересующих сферах (ремонт, репетиторство и т.д.), но и самим выступать в роли специалистов, предлагая собственные услуги другим пользователям.

Для достижения заданной цели необходимо выполнить следующие задачи:

* проанализировать существующие решения и на их основе выделить ключевые моменты, которые следует реализовать (глава 1);
* определить структуру веб-приложения (глава 2);
* определить используемые технологии (глава 2);
* реализовать веб-приложение (глава 3);
* убедиться в работоспособности сервиса (глава 4);
* объяснить пользователю, как пользователя сервисом (глава 5).

Целевая аудитория сервиса — частные лица и специалисты, желающие предложить свои услуги, а также пользователи, заинтересованные в поиске и бронировании квалифицированных исполнителей. Платформа рассчитана как на профессионалов с опытом, так и на начинающих специалистов, а также на заказчиков, которым важно удобство, прозрачность и надёжность при взаимодействии с исполнителями.

Программной платформой веб-приложения является Node.js [1] v22.13.1, благодаря своей масштабируемости и активной экосистеме, поддерживаемой npm [2] v10.9.2.

# 1 Постановка задачи и обзор аналогичных решений

## 1.1 Постановка задачи

Основной задачей курсового проектирования является разработка веб-приложения для сервиса предоставления услуг.

Данное приложение будет иметь возможность различные услуги, добавление своих и их бронирование. Статусом бронирования сможет управлять как и обычный пользователь, так и специалист. Также будет доступна возможность оплаты.

Для поиска задач необходимо разделить их на различные категории и добавить фильтры. Для удобства их можно добавлять в избранные.

Для упрощения коммуникации пользователей также будет разработан чат, в котором они смогут общаться и уточнять детали бронирования.

## 1.2 Обзор аналогичных решений

### 1.2.1 Аналог «Onliner»

«Onliner» [3] – один из крупнейших белорусских интернет-порталов, включающий в себя маркетплейс, новостной раздел, форум и сервис для размещения объявлений. Среди множества функций особое место занимает раздел «Услуги», где пользователи могут как предлагать собственные услуги, так и искать исполнителей в различных сферах: ремонт, строительство, образование, красота, техника и др. Платформа отличается высокой посещаемостью и узнаваемостью в Беларуси, что обеспечивает широкий охват аудитории и доверие со стороны пользователей (рисунок 1.1).

Ключевыми особенностями являются:

* разнообразие категорий услуг, охватывающее бытовую, профессиональную и образовательную сферы;
* возможность зарегистрироваться как частному лицу, так и компании;
* наличие отзывов и рейтингов, повышающих прозрачность и доверие;

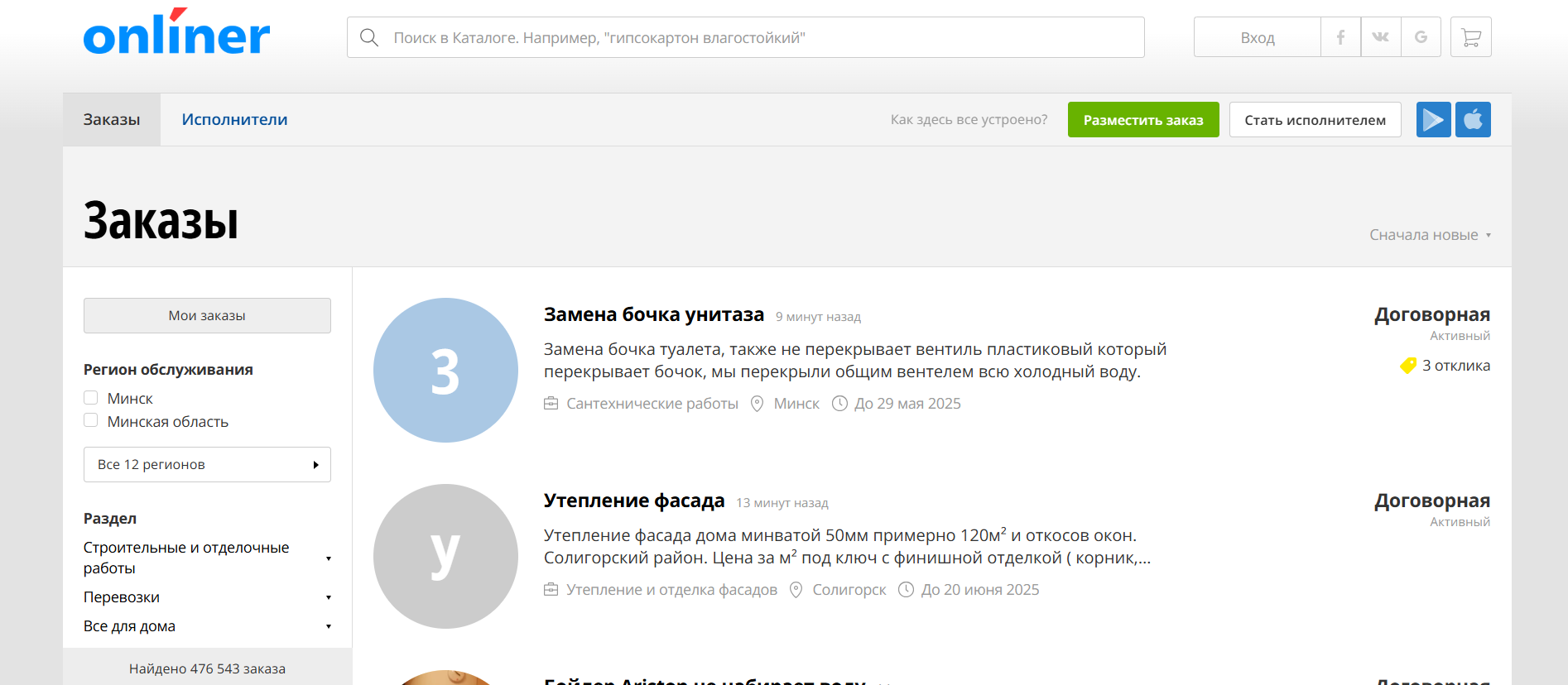


Рисунок 1.1 – Раздел услуг сайта «Onliner»

### 1.2.2 Аналог «YouDo»

«YouDo» [4] — российская онлайн-платформа, предназначенная для поиска и предложения услуг в различных сферах, таких как ремонт, уборка, курьерские доставки, репетиторство, дизайн, IT и другие. Сервис функционирует по модели маркетплейса, позволяя пользователям выступать как в роли заказчиков, размещая задания, так и в роли исполнителей, предлагая свои услуги.

Платформа обеспечивает удобный интерфейс для взаимодействия между пользователями, включая возможность фильтрации заданий по категориям, регионам и стоимости. Исполнители проходят процедуру верификации, что повышает уровень доверия со стороны заказчиков. Кроме того, система рейтингов и отзывов способствует формированию прозрачной и надежной среды для сотрудничества (рисунок 1.2).

Ключевыми особенностями являются:

* широкий спектр категорий услуг, охватывающий как бытовые, так и профессиональные направления;
* возможность для пользователей быть как заказчиками, так и исполнителями;
* система рейтингов и отзывов, обеспечивающая прозрачность и доверие;
* наличие отзывов и рейтингов, повышающих прозрачность и доверие;
* поддержка сделок с использованием системы «Сделка без риска».

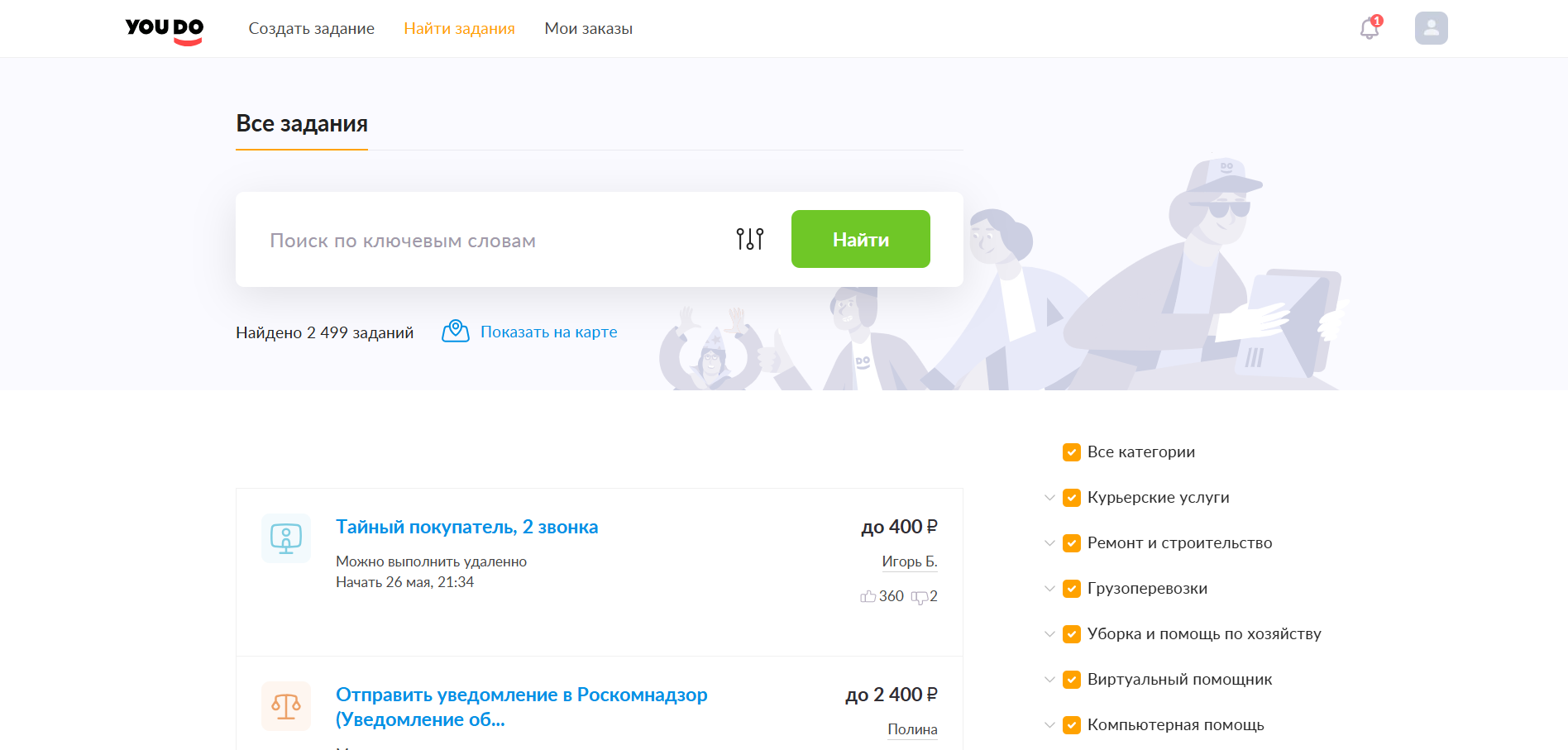


Рисунок 1.2 – главная страница сайта «YouDo»

### 1.2.2 Аналог «Профи.ру»

«Профи.ру» [5] – ведущая российская онлайн-платформа, функционирующая как маркетплейс услуг, который объединяет заказчиков и исполнителей в различных профессиональных сферах. Сервис охватывает широкий спектр направлений, включая строительство и ремонт, клининговые услуги, логистику, образование, дизайн, IT-услуги и многие другие.

Платформа реализует двустороннюю модель взаимодействия. Для заказчиков предусмотрена возможность публикации заданий с детальным описанием требований, а для исполнителей – размещение профилей с портфолио и предложением своих услуг.

Ключевыми особенностями являются:

* система верификации документов и блокировка нарушителей;
* огромная база проверенных специалистов;
* клиенты публикуют задачи, а мастера сами отправляют предложения;



Рисунок 1.3 – главная страница сайта «Профи.ру»

# 1.3 Выводы по главе

* основными задачами разработки веб-приложения являются: создание сервиса с поддержкой ролей администратора, пользователя и гостя, реализация функционала поиска и просмотра всех услуг и услуг клиента, просмотра избранного и бронирований, управления услугами и бронированиями, оплаты бронирований и модерации клиентов;
* в процессе анализа веб-приложений были выявлены как преимущества, так и недостатки каждого из них, которые могут быть учтены при разработке нового решения для предоставления услуг;

# 2 Проектирование веб-приложения

## 2.1 Функциональность веб-приложения

Функциональные возможности веб-приложения представлены в диаграмме вариантов использования, представленной в приложении А.

В веб-приложении предусмотрены 3 роли: «Гость», «Клиент» и «Администратор». Описание ролей представлено в таблице 2.1.

Таблица 2.1 – Описание ролей

|  |  |
| --- | --- |
| Название роли | Описание |
| Гость | Роль «Гость» присваивается неаутентифицированным пользователям. Она предусматривает доступ к функционалу, не требующему авторизации, в том числе просмотр доступных услуг, регистрация и вход в систему. |
| Клиент | Роль «Клиент» присваивается после успешной аутентификации. Пользователь с данной ролью получает доступ к основному функционалу веб-приложения. |
| Администратор | Роль «Администратор» выдается пользователю, который создаётся при инициализации веб-приложения. Данная роль предусматривает функционал, связанный с управлением веб-приложением. |

Описание функциональных возможностей ролей представлено в таблице 2.2.

Таблица 2.2 – Описание функциональных возможностей

|  |  |  |
| --- | --- | --- |
| Роль | Функция | Описание |
| Гость | Регистрация | Создание учетной записи |
| Гость | Аутентификация | Получение доступа к основному функционалу веб-приложения |
| Гость,  Клиент, Администратор, | Просмотр, поиск, и фильтрация услуг | Получение списка услуг, подходящих под заданные критерии |
| Гость,  Клиент, Администратор, | Просмотр информации об услуге | Ознакомление с подробной информацией об услуге |
| Клиент | Добавление услуги в избранное | Добавление услуги в свой список избранного |
| Клиент | Удаление услуги из избранного | Удаление услуги из своего списка избранного |
| Клиент | Добавление услуги | Создание новой услуги |
| Клиент | Изменение услуги | Изменение услуги |
| Клиент,  Администратор | Удаление услуги | Удаление услуги |

Окончание таблицы 2.2

|  |  |  |
| --- | --- | --- |
| Роль | Функция | Описание |
| Клиент | Общение с представителем услуги в чате | Обмен сообщениями с другим пользователем с использованием чата |
| Клиент | Бронирование услуги | Бронирование услуги с определенной датой и временем исполнения |
| Клиент | Просмотр истории личных бронирований | Получение своего списка бронирований за все время |
| Клиент | Просмотр бронирований своей услуги | Получение списка бронирований, оформленных другими пользователями на опубликованную им услугу |
| Клиент | Изменение статуса бронирования услуги | Возможность изменить статус бронирования в зависимости от своей роли в сделке |
| Клиент | Оплата выполненной услуги | Оплата по факту выполнения услуги – возможность оплатить бронирование только после его успешного завершения исполнителем. |
| Клиент | Добавление отзыва на услугу | Оставить свой отзыв об услуге после выполнения оплаты |
| Клиент | Удаление отзыва на услугу | Удалить свой отзыв |
| Клиент | Изменение информации о пользователе | Возможность обновления персональных данных пользователя |
| Администратор | Просмотр списка пользователей | Просмотр списка всех пользователей приложения |
| Администратор | Блокирование пользователя | Ограничение доступа пользователя к функционалу сервиса |
| Администратор | Разблокирование пользователя | Возвращение пользователю доступа к функционалу сервиса |
| Администратор | Удаление пользователя | Удаление пользователя, всех его услуг, отзывов и бронирований |

## 2.2 Логическая схема базы данных

Для обеспечения функциональности веб-приложения необходимо разработать таблицы базы данных. Логическая схема базы данных представлена в приложении Б.

Для поддержания работы веб-приложения необходимо 7 таблиц. Между таблицами используется связь «многие ко многим». В таблице 2.3 представлен список таблиц с их описанием.

Таблица 2.3 – Описание таблиц базы данных

|  |  |
| --- | --- |
| Таблица | Описание |
| Users | Таблица содержит информацию о пользователях, данные для аутентификации |
| RefreshTokens | Таблица содержит информацию о refresh-токенах [6], необходимых для поддержки механизма обновления доступа пользователей без повторной аутентификации. |
| Services | Таблица содержит информацию об услугах, предоставляемых пользователями |
| Reviews | Таблица содержит информацию об отзывах на услуги, которые оставляют пользователи |
| Bookings | Таблица содержит информацию об бронированиях услуг пользователей и их статусах |
| Favorites | Таблица содержит информацию о связи пользователей услуг, необходимо для реализации функционала списка избранного |
| Messages | Таблица содержит информацию о сообщениях, которые отправляют пользователи друг другу в чатах |

В таблице «Users» хранится информация о пользователях, их email, имена, фамилии пароли, номера телефона, аватары, роли, статусы блокировки и дата регистрации, описание таблицы приведено в таблице 2.4.

Таблица 2.4 – Описание таблицы «Users»

|  |  |  |
| --- | --- | --- |
| Атрибут | Описание | Тип данных |
| id | Уникальный идентификатор пользователя, первичный ключ | string |
| email | Уникальный почтовый ящик пользователя | string |
| password | Захешированный пароль | string |
| firstName | Имя пользователя | string |
| lastName | Фамилия пользователя | string |
| phone | Номер телефона пользователя | string |
| profilePhotoPath | Путь к файлу с аватаром | string |
| userType | Роль, принимает значения «USER» и «ADMIN» | string |
| isBlocked | Статус блокировки пользователя | boolean |
| createdAt | Дата регистрации пользователя | datetime |

Таблица «RefreshTokens» содержит о refresh-токенах пользователей, их дату создания и дату окончания времени действия. Описание таблицы представлено в таблице 2.5.

Таблица 2.5 – Описание таблицы «RefreshTokens»

|  |  |  |
| --- | --- | --- |
| Атрибут | Описание | Тип данных |
| id | Уникальный идентификатор задачи, первичный ключ | string |
| userId | Уникальный идентификатор пользователя, внешний ключ | string |
| token | Уникальное значение токена | string |
| expiresAt | Время истечения срока действия токена | datetime |
| createdAt | Дата создания токена | datetime |

В таблице «Services» хранится информация об услугах: название, описание, цена, категория, длительность выполнения, рейтинг, изображение. Описание таблицы приведено в таблице 2.6.

Таблица 2.6 – Описание таблицы «Services»

|  |  |  |
| --- | --- | --- |
| Атрибут | Описание | Тип данных |
| id | Уникальный идентификатор теста, первичный ключ | string |
| providerId | Уникальный идентификатор пользователя, создавшего услугу, внешний ключ | string |
| title | Выходные данные теста | string |
| description | Виден ли тест клиентам | boolean |
| price | Уникальный идентификатор задачи, внешний ключ | integer |
| category | Название категории, к которой относится услуга | string |
| duration | Длительность выполнения услуги в минутах | integer |
| rating | Средний рейтинг услуги, основанный на отзывах | float |
| photoPath | Путь к файлу изображения услуги | string |
| createdAt | Дата создания услуги | datetime |

Данные об отзывах хранятся в таблице «Reviews». Описание таблицы приведено в таблице 2.7.

Таблица 2.7 – Описание таблицы «Reviews»

|  |  |  |
| --- | --- | --- |
| Атрибут | Описание | Тип данных |
| id | Уникальный идентификатор теста, первичный ключ | string |
| userId | Уникальный идентификатор пользователя, внешний ключ | string |

Окончание таблицы 2.7

|  |  |  |
| --- | --- | --- |
| Атрибут | Описание | Тип данных |
| serviceId | Уникальный идентификатор услуги, внешний ключ | string |
| rating | Оценка клиента, принимает значения от 1 до 5 | float |
| comment | Комментарий к отзыву | string |
| createdAt | Дата написания отзыва | datetime |

В таблице «Bookings» содержится информация о бронировании: связь с пользователем и услугой, датой выполнения, статусе. Описание таблицы представлено в таблице 2.8.

Таблица 2.8 – Описание таблицы «Bookings»

|  |  |  |
| --- | --- | --- |
| Атрибут | Описание | Тип данных |
| id | Уникальный идентификатор языка, первичный ключ | string |
| userId | Уникальный идентификатор пользователя, внешний ключ | string |
| serviceId | Уникальный идентификатор услуги, внешний ключ | string |
| date | Дата выполнения услуги | string |
| status | Статус услуги, принимает значения «PENDING», «CANCELLED», «CONFIRMED», «COMPLETED» и «PAID» | string |
| createdAt | Дата создания бронирования | datetime |

Данные об избранных услугах пользователей содержатся в таблице «Favorites». Описание таблицы приведено в таблице 2.9.

Таблица 2.9 – Описание таблицы «Favorites»

|  |  |  |
| --- | --- | --- |
| Атрибут | Описание | Тип данных |
| id | Уникальный идентификатор избранной услуги, первичный ключ | string |
| clientId | Уникальный идентификатор пользователя, внешний ключ | string |
| serviceId | Уникальный идентификатор услуги, внешний ключ | string |
| createdAt | Дата добавления в избранное | datetime |

Информация о сообщениях пользователей из чатов содержится в таблице «Messages». Описание таблицы представлено в таблице 2.10.

Таблица 2.10 – Описание таблицы «Messages»

|  |  |  |
| --- | --- | --- |
| Атрибут | Описание | Тип данных |
| id | Уникальный идентификатор сообщения пользователя, первичный ключ | string |

Окончание таблицы 2.10

|  |  |  |
| --- | --- | --- |
| Атрибут | Описание | Тип данных |
| senderId | Уникальный идентификатор пользователя, отправившего сообщение, внешний ключ | string |
| receiverId | Уникальный идентификатор задачи, получившего сообщение, внешний ключ | string |
| content | Текст сообщения | string |
| sentAt | Время отправления сообщения | datetime |

Между таблицами имеются различные виды связей, описание связей между таблицами представлено в таблице 2.11.

Таблица 2.11 – Описание связей между таблицами

|  |  |
| --- | --- |
| Таблицы | Вид связи |
| «Users» и «RefreshTokens» | Один ко многим |
| «Users» и «Reviews» | Один ко многим |
| «Users» и «Favorites» | Один ко многим |
| «Users» и «Messages» | Один ко многим |
| «Users» и «Bookings» | Один ко многим |
| «Users» и «Services» | Один ко многим |
| «Services» и «Bookings» | Один ко многим |
| «Services» и «Reviews» | Один ко многим |

## 2.3 Архитектура веб-приложения

Диаграмма развертывания, описывающая архитектуру веб-приложения, представлена в приложении В.

В качестве серверной операционной системы, на котором запускаются серверные компоненты системы, выступает Ubuntu Server 24.04.2 LTS. [7].

Для удобства управления компонентами системы, обеспечения их работоспособности на любой операционной системе, поддерживающей виртуализацию, использовались Docker [8] и Docker Compose [9].

В таблице 2.12 представлены основные компоненты архитектуры, их описание и назначение.

Таблица 2.12 – Основные компоненты архитектуры

|  |  |
| --- | --- |
| Компонент | Описание |
| Web Browser | Используется клиентом для отображения фронтенд-части веб-приложения, отправки запросов на бэкенд-часть, обработки полученных результатов |
| Web Server | Является точкой доступа клиента к веб-приложению, предоставляет статические файлы с фронтендом, а также позволяет получить доступ к бэкенду |

Окончание таблицы 2.12

|  |  |
| --- | --- |
| Компонент | Описание |
| Application Server | Обрабатывает запросы пользователей, обращается за данными к базе данных |
| Database Server | Хранит пользовательские данные, предоставляет доступ к ним, а также выполняет базовые операции с ними, такие как добавление, удаление, изменение |
| SMTP [10] Server | Позволяет выполнять почтовую рассылку пользователям |
| Stripe [11] API | Позволяет совершать безопасные платежные операции, такие как создание и управление платежными сессиями. |

Сетевые протоколы, используемые в веб-приложение, их назначение, а также варианты использования представлены в таблице 2.13.

Таблица 2.13 – Сетевые протоколы, используемые в веб-приложении

|  |  |
| --- | --- |
| Протокол | Описание |
| HTTP 1.1 [12] | Используется для обмена данными между Application Server и Web Server, Web Server и WebBrowser |
| TCP [13] | Используется для обмена данными между Application Server и Database Server, гарантирует надежность передачи данных между ними |
| WebSocket [14] | Используется для двустороннего обмена данными в реальном времени между Application Server и Web Server. Обеспечивает установление постоянного соединения, позволяющего серверу инициировать передачу данных к клиенту без необходимости повторных HTTP-запросов |

## 2.4 Выводы по главе

В данной главе были описаны следующие роли:

* гость, который имеет возможности аутентификации и регистрации, просмотра, поиска и фильтрации услуг и просмотра информации об услуге;
* клиент, который может управлять и взаимодействовать с услугами, бронированиями, отзывами, избранными услугами и обмениваться сообщениями в чатах;
* администратор, который модерирует услуги и пользователей с возможностью их блокировки.

Структура базы данных состоит из 7 таблиц: «Users», «RefreshTokens», «Services», «Favorites», «Bookings», «Reviews», «Messages».

Определены следующие протоколы: HTTP, TCP, WebSocket.

Архитектура веб-приложения состоит из 6 компонентов: Web Browser, Web Server, Application Server, Database Server, SMTP Server, Stripe API.

# 3 Реализация веб-приложения

## 3.1 Программная платформа

Для разработки серверной части веб-приложения использовалась платформа Node.js, которая способна обрабатывать запросы одновременно благодаря событийно-ориентированной архитектуре. Для работы с библиотеками использовался пакетный менеджер npm, в процессе разработки серверной части использовался фреймворк Nest.js [15], предоставляющие набор готовых реализаций стандартных решений, при разработке веб-приложений.

Вместе с Nest.js использовались паттерны Dependency Injection [16], DTO [17].

Паттерн Dependency Injection позволяет упростить работу с зависимостями, предоставляя механизм, который сам управляет ими и их жизненным циклом.

Паттерн DTO используется для передачи данных между различными компонентами системы, позволяя скрыть приватные поля или представить данные в более удобном формате.

## 3.2 Система управления базами данных PostgreSQL

В качестве СУБД, используемой в проекте, была выбрана PostgreSQL [18]. Это реляционная база данных, поддерживающая транзакции, имеет открытый исходный код и распространяется бесплатно. Скрипт создания базы данных представлен в приложении Г.

## 3.3 Object-Relational Mapping Prisma

Для удобства работы с базой данных используется Prisma ORM [19], которая позволяет работать с данными так, словно это объекты. Prisma автоматически генерирует типы для генерируемых таблиц, имеет собственный формат описания данных.

Для работы с Prisma необходимо определить схему. Схема, используемая в веб-приложении, представлена в приложении Д.

Следующим шагом является создание миграции и генерация PrismaClient, который используется в веб-приложении для работы с базой данных. Команды, необходимые для выполнения данных действий представлены в листинге 3.1.

npx prisma migrate dev -name init

npx prisma generate

Листинг 3.1 – Команды для генерации PrismaClient

После генерации PrismaClient, приложение способно выполнять запросы к базе данных.

## 3.4 Библиотеки и фреймворки

Набор библиотек и фреймворков, использованных для работы серверной части веб-приложения представлен в таблице 3.1.

Таблица 3.1 – Набор библиотек и фреймворков серверной части

|  |  |  |
| --- | --- | --- |
| Библиотека | Версия | Описание |
| @nestjs/common | 11.0.1 | основные утилиты и декораторы NestJS (контроллеры, сервисы, middleware) |
| @nestjs/config | 4.0.0 | Управление конфигурацией веб-приложения через .env файл |
| @nestjs/core | 11.0.17 | Ядро NestJS, реализует модульную архитектуру и механизм внедрения зависимостей (DI) |
| @nestjs/jwt | 11.0.0 | Модуль для работы с JWT[23] (аутентификация и авторизация) |
| @nestjs/passport | 11.0.0 | Интеграция Passport.js в NestJS для реализации стратегий аутентификации |
| @nestjs/platform-express | 11.0.17 | Адаптер NestJS для запуска на платформе Express [24] |
| @nestjs/platform-socket.io | 11.1.0 | Поддержка WebSocket через Socket.IO |
| @nestjs/serve-static | 5.0.3 | Служит для раздачи статических файлов, например, frontend-части |
| @nestjs/swagger | 11.1.3 | Генерация OpenAPI [25] (Swagger) документации для REST API |
| @nestjs/websockets | 11.1.0 | Поддержка WebSocket-соединений в NestJS |
| @prisma/client | 5.0.0 | Клиент для работы с базой данных через Prisma ORM |
| bcrypt | 5.1.0 | Преобразование объектов между различными слоями (например, DTO и Entity) |
| class-transformer | 0.5.1 | генерация уникальных идентификаторов |
| class-validator | 0.14.0 | Валидация данных через аннотации или декораторы |
| multer | 1.4.5-lts.2 | Middleware для обработки multipart/form-data, загрузка файлов |
| nodemailer | 7.0.3 | Отправка писем через SMTP (например, уведомления) |
| passport | 0.6.0 | Основной модуль аутентификации, используется с различными стратегиями |
| passport-jwt | 4.0.1 | JWT-стратегия для Passport |
| prisma | 5.0.0 | CLI-инструмент для генерации Prisma-клиента и управления миграциями |
| reflect-metadata | 0.1.13 | Поддержка метаданных, необходима для работы декораторов |
| sharp | 0.34.1 | Обработка изображений (сжатие, обрезка и т.д.) |

Окончание таблицы 3.1

|  |  |  |
| --- | --- | --- |
| Библиотека | Версия | Описание |
| socket.io | 4.8.1 | Библиотека для работы с WebSocket (серверная часть) |
| stripe | 18.1.1 | Интеграция с платежной системой Stripe для приема и обработки платежей |
| swagger-ui-express | 5.0.1 | Визуализация документации API на основе Swagger/OpenAPI |

Для разработки клиентской части в качестве основной библиотеке использовался React. Полный набор библиотек, использованных при разработке клиентской части представлен в таблице 3.2.

Таблица 3.2 – Набор библиотек и фреймворков клиентской части

|  |  |  |
| --- | --- | --- |
| Библиотека | Версия | Описание |
| @emotion/react | 11.14.0 | Библиотека для стилизации компонентов с использованием Emotion (CSS-in-JS) |
| @emotion/styled | 11.14.0 | API для создания стилизованных компонентов на основе Emotion |
| @mui/icons-material | 7.0.2 | Иконки Material Design для использования с MUI |
| @mui/material | 7.0.2 | Компонентная библиотека Material UI для React |
| @mui/types | 7.4.1 | Типы TypeScript для библиотеки MUI |
| @mui/x-date-pickers | 8.2.0 | Компоненты выбора даты и времени на базе MUI |
| axios | 1.9.0 | HTTP-клиент для отправки запросов на сервер |
| dayjs | 1.11.13 | Легкая библиотека для работы с датами и временем |
| react | 19.0.0 | библиотека для создания пользовательских интерфейсов |
| react-dom | 19.0.0 | рендеринг React-компонентов в браузере |
| react-router-dom | 7.4.0 | маршрутизация в React-приложениях |
| react-toastify | 11.0.5 | удобные уведомления в React |
| socket.io-client | 4.8.1 | Клиент для подключения к серверу WebSocket через Socket.IO |

## 3.5 Структура серверной части

Серверная часть веб-приложения реализована с использованием фреймворка NestJS, который построен на модульной архитектуре и ориентирован на разделение ответственности. В структуре серверного проекта можно выделить ключевые директории, каждая из которых соответствует определённому слою или компоненту архитектуры. В таблице 3.3 приведено описание основных директорий и их назначение.

Таблица 3.3 – Описание основных директорий и их назначение

|  |  |
| --- | --- |
| Название директория | Назначение |
| controllers | Содержит контроллеры, обрабатывающие HTTP-запросы и маршруты. Отвечают за приём входящих запросов и возврат ответов клиенту |
| decorators | Пользовательские декораторы, применяемые к классам, методам или параметрам, для расширения функциональности |
| dto | Классы объектов передачи данных, определяющие структуру данных, передаваемых между клиентом и сервером |
| enums | Перечисления, содержащие набор фиксированных значений, используемых в приложении |
| filters | Глобальные и локальные фильтры исключений, используемые для обработки и форматирования ошибок |
| gateways | Веб-сокет-шлюзы, реализующие двустороннюю связь между сервером и клиентом в реальном времени |
| guards | Стражи, определяющие, имеет ли пользователь доступ к определённым маршрутам или функциям (например, авторизация, роли) |
| interfaces | Интерфейсы TypeScript, определяющие контракт структуры данных и обеспечивающие строгую типизацию |
| modules | Модули NestJS, объединяющие связанную функциональность, контроллеры, сервисы и провайдеры, в логические блоки |
| prisma | Конфигурации и клиент Prisma ORM, используемые для работы с базой данных |
| services | Сервисы, реализующие бизнес-логику приложения. Вызываются из контроллеров или других сервисов |
| strategies | Стратегии авторизации с использованием библиотеки passport |
| uploads | Хранение статических файлов, загружаемых пользователем |

Соответствие маршрутов контроллерам и функциям серверной части представлено в таблице 3.3.

Таблица 3.3 – Соответствие маршрутов контроллерам и функциям

|  |  |  |  |
| --- | --- | --- | --- |
| Метод | Маршрут | Контроллер | Метод контроллера |
| POST | /api/auth/register | AuthController | register |
| POST | /api/auth/login | AuthController | login |
| POST | /api/auth/refresh | AuthController | refreshTokens |
| POST | /api/auth/logout | AuthController | logout |
| GET | /api/users | UsersController | findAllUsers |
| GET | /api/users/:id | UsersController | findUser |
| PUT | /api/users/id | UsersController | updateUser |
| DELETE | /api/users/:id | UsersController | removeUser |

Окончание таблицы 3.3

|  |  |  |  |
| --- | --- | --- | --- |
| Метод | Маршрут | Контроллер | Метод контроллера |
| GET | /api/users/profile/me | UsersController | findMyProfile |
| PUT | /api/users/:id/password | UsersController | changeUserPassword |
| PUT | /api/users/:id/block | UsersController | blockUser |
| PUT | /api/users/:id/unblock | UsersController | unblockUser |
| POST | /api/services | ServicesController | createService |
| GET | /api/services | ServicesController | findAllServices |
| GET | /api/services/:id | ServicesController | findService |
| PUT | /api/services/:id | ServicesController | updateService |
| DELETE | /api/services/:id | ServicesController | removeService |
| GET | /api/services/provider/me | ServicesController | findMyServices |
| POST | /api/bookings | BookingsController | create |
| PUT | /api/bookings/:id | BookingsController | updateBooking |
| DELETE | /api/bookings/:id | BookingsController | removeBooking |
| GET | /api/bookings/user/me | BookingsController | findMyBookings |
| GET | /api/bookings/service/:serviceId | BookingsController | findByService |
| POST | /api/bookings/pay | BookingsController | payForBooking |
| POST | /api/stripe/webhook | StripeController | webhook |
| POST | /api/favorites | FavoritesController | addToFavorites |
| DELETE | /api/favorites/:id | FavoritesController | removeFromFavorites |
| GET | /api/favorites/:userId | FavoritesController | getUserFavorites |
| POST | /api/reviews | ReviewsController | createReview |
| DELETE | /api/reviews/:id | ReviewsController | removeReview |
| GET | /api/messages/chat/:userId | MessagesController | findChatMessages |

Соответствие событий WebSocket обработчикам и методам представлены в таблице 3.4.

Таблица 3.4 – Соответствие событий WebSocket обработчикам и методам

|  |  |  |
| --- | --- | --- |
| Событие WebSocket | Gateway | Метод обработчика |
| register | ChatGateway | handleRegister |
| sendMessage | ChatGateway | handleMessage |

Для передачи данных в теле используется формат JSON [20]. Для валидации данных используются DTO, где, используя декораторы, описаны правила валидации входных данных.

Для передачи медиа со стороны клиента использовался FormData [21], а со стороны сервера multer, который записывает получаемые данные на диск.

Серверная часть реализована с соблюдением архитектурных принципов REST [22]. Для каждой сущности определены маршруты с использованием стандартных HTTP-методов (GET, POST, PUT, DELETE), обеспечивающих единообразие и предсказуемость взаимодействия клиента с API.

## 3.6 Реализация функций

В соответствии с диаграммой вариантов использования, в коде реализованы функции на уровне бизнес-логики. Соответствие реализаций функциям представлено в таблице 3.5.

Таблица 3.5 – Описание функций

|  |  |  |
| --- | --- | --- |
| Название функции | Название действия | Описание |
| register | Регистрация | Регистрирует пользователя |
| login | Аутентификация | Проверяет учётные данные пользователя, возвращает JWT-токен и refresh-токен |
| getAllServices | Просмотр, поиск, и фильтрация услуг | Получение списка услуг, подходящих под заданные критерии |
| getService | Просмотр информации об услуге | Ознакомление с подробной информацией об услуге |
| createFavorite | Добавление услуги в избранное | Добавление услуги в свой список избранного |
| removeFromFavorite | Удаление услуги из избранного | Удаление услуги из своего списка избранного |
| addToFavorite | Добавление услуги | Создание новой услуги |
| updateService | Изменение услуги | Изменение услуги |
| removeService | Удаление услуги | Удаление услуги |
| handleMessage | Общение с представителем услуги в чате | Обмен сообщениями с другим пользователем с использованием чата |
| createBooking | Бронирование услуги | Бронирование услуги с определенной датой и временем исполнения |
| findBookingsByUser | Просмотр истории личных бронирований | Получение своего списка бронирований за все время |
| findBookingsByService | Просмотр бронирований своей услуги | Получение списка бронирований, оформленных другими пользователями на опубликованную им услугу |
| updateBooking | Изменение статуса бронирования услуги | Возможность изменить статус бронирования в зависимости от своей роли в сделке |

Окончание таблицы 3.5

|  |  |  |
| --- | --- | --- |
| Название функции | Название действия | Описание |
| createCheckoutSession | Оплата выполненной услуги | Оплата по факту выполнения услуги – возможность оплатить бронирование только после его успешного завершения исполнителем. |
| createReview | Добавление отзыва на услугу | Оставить свой отзыв об услуге после выполнения оплаты |
| removeReview | Удаление отзыва на услугу | Удалить свой отзыв |
| updateUser | Изменение информации о пользователе | Возможность обновления персональных данных пользователя |
| getAllUsers | Просмотр списка пользователей | Просмотр списка всех пользователей приложения |
| toogleBlockUser | Блокирование пользователя | Ограничение доступа пользователя к функционалу сервиса |
| toogleBlockUser | Разблокирование пользователя | Возвращение пользователю доступа к функционалу сервиса |
| removeUser | Удаление пользователя | Удаление пользователя, всех его услуг, отзывов и бронирований |

### 3.6.1 Регистрация

Метод «register» в «AuthService» реализует процесс регистрации нового пользователя. Он принимает данные регистрации, включая электронную почту, пароль, имя, фамилию, номер телефона и путь к изображению профиля. Реализация метода представлена в листинге 3.2.

async register(registerDto: RegisterDto, imagePath: string): Promise<Tokens> {  
 const { email, password, firstName, lastName, phone } = registerDto;  
 const userExists = await this.prisma.user.findUnique({  
 where: { email },  
 });  
 if (userExists) {  
 throw new ConflictException('Email уже используется');  
 }  
 const hashedPassword = await this.hashPassword(password);  
 const user = await this.prisma.user.create({  
 data: {  
 email,  
 password: hashedPassword,  
 firstName,  
 lastName,  
 phone,  
 userType: Role.*User*,  
 profilePhotoPath : imagePath  
 },  
 });  
 const tokens = await this.jwtTokensService.generateTokens(user.id, user.email, user.userType as Role);  
 await this.jwtTokensService.storeRefreshToken(user.id, tokens.refreshToken);  
 return tokens;  
}

Листинг 3.2 – Реализация метода «register»

### 3.6.2 Аутентификация

Метод «login» в «AuthService» выполняет аутентификацию пользователя. Он проверяет учётные данные и генерирует access (JWT) и refresh токены.

Реализация метода «login» представлена в листинге 3.3.

async login(loginDto: LoginDto): Promise<{ accessToken: string; refreshToken: string; user: Omit<User, 'password'> }> {  
 const user = await this.validateUser(loginDto.email, loginDto.password);  
  
 const tokens = await this.jwtTokensService.generateTokens(user.id, user.email, user.userType as Role);  
 await this.jwtTokensService.storeRefreshToken(user.id, tokens.refreshToken);  
  
 return {  
 accessToken: tokens.accessToken,  
 refreshToken: tokens.refreshToken,  
 user,  
 };  
}

Листинг 3.3 – Реализация метода «login»

### 3.6.3 Просмотр, поиск и фильтрация услуг

Метод «findAllServices» в «ServicesService» предназначен для получения списка услуг из базы данных с учетом заданных фильтров, а также поддержки пагинации. Входными параметрами метода являются номер страницы, количество элементов на странице и объект фильтров, содержащий условия по цене, категории, рейтингу и названию. Реализация метода представлена в листинге 3.4.

async findAllServices(page: number, limit: number, filters: ServiceFilters) {  
 const skip = (page - 1) \* limit;  
 const where: any = {};  
 if (filters.minPrice !== undefined || filters.maxPrice !== undefined) {  
 where.price = {};  
 if (filters.minPrice !== undefined) {  
 where.price.gte = filters.minPrice;  
 }  
 if (filters.maxPrice !== undefined) {  
 where.price.lte = filters.maxPrice;  
 }  
 }  
 if (filters.category) { where.category = filters.category;}  
 if (filters.minRating !== undefined) {where.rating = { gte: filters.minRating };}  
 if (filters.title) {  
 where.title = { contains: filters.title, mode: 'insensitive' };  
 }  
 const [services, total] = await this.prisma.$transaction([  
 this.prisma.service.findMany({  
 skip,  
 take: limit,  
 where,  
 include: {  
 provider: true,  
 reviews: true,  
 },  
 }),  
 this.prisma.service.count({ where }),  
 ]);  
 return {  
 data: services,  
 meta: {  
 total,  
 page,  
 limit,  
 totalPages: *Math*.ceil(total / limit),  
 },  
 };  
}

Листинг 3.4 – Реализация метода «getAllServices»

### 3.6.4 Просмотр информации об услуге

Метод «findService» в «ServicesService» используется для получения полной информации об определённой услуге по её идентификатору. В случае успешного выполнения метод возвращает объект, представляющий полную информацию об услуге, включая поставщика и отзывы с авторами. Реализация метода представлена в листинге 3.5.

async findService(id: string) {  
 const service = await this.prisma.service.findUnique({  
 where: { id },  
 include: {  
 provider: true,  
 reviews: {  
 include:{  
 user:true  
 }  
 },  
 },  
 });  
  
 if (!service) {  
 throw new NotFoundException('Услуга не найдена');  
 }  
 return service;  
}

Листинг 3.5 – Реализация метода «findService»

### 3.6.5 Добавление услуги в избранное

Для добавления услуги в избранное используется метод «createFavorite» из «FavoritesService», который просто добавляет запись в базу данных, реализация метода представлена в листинге 3.6.

async addToFavorites(dto: CreateFavoriteDto) {  
 const existing = await this.prisma.favorite.findUnique({  
 where: {  
 userId\_serviceId: {  
 userId: dto.userId,  
 serviceId: dto.serviceId,  
 },  
 },  
 });  
 if (existing) {  
 throw new HttpException(  
 'Сервис уже находится в избранном',  
 HttpStatus.*BAD\_REQUEST*,  
 );  
 }  
 return this.prisma.favorite.create({  
 data: {  
 user: { connect: { id: dto.userId } },  
 service: { connect: { id: dto.serviceId } },  
 },  
 include: {  
 service: true,  
 },  
 });  
}

Листинг 3.6 – Реализация метода «createFavorite»

### 3.6.6 Удаление услуги из избранного

Для удаления услуги из избранного используется метод «removeFromFavorite» из «FavoritesService», который просто удаляет запись из базы данных. Реализация метода представлена в листинге 3.7.

async removeFromFavorites(userId: string, serviceId: string) {  
 return this.prisma.favorite.delete({  
 where: {  
 userId\_serviceId: {  
 userId,  
 serviceId,  
 },  
 },  
 });  
}

Листинг 3.7 – Реализация метода «removeFromFavorite»

### 3.6.7 Добавление услуги

Метод «createService» в «Services» используется для создания новой услуги и добавления ее в таблицу. Метод принимает объект, содержащий данные о создаваемой услуге. Так же при создании передается информация о создателе услуги. Реализация метода представлена в листинге 3.8.

async createService(data: CreateServiceData) {

const { title, description, price, category, providerId, photoPath, duration } = data;

return this.prisma.service.create({

data: {

title,

description,

price,

category,

photoPath,

duration,

provider: {

connect: { id: providerId },

},

},

include: {

provider: true,

reviews: true,

},

});

}

Листинг 3.8 – Реализация метода «createService»

### 3.6.8 Изменение услуги

Метод updateService реализует функциональность редактирования существующей услуги. На первом этапе метод проверяет существование услуги по заданному идентификатору. В случае её отсутствия выбрасывается исключение. Затем выполняется проверка авторизации: если идентификатор создателя услуги и текущего пользователя не совпадает, выбрасывается исключение «ForbiddenException», запрещающее изменение чужой услуги. Реализация метода представлена в листинге 3.9.

async updateService(id: string, data: UpdateServiceData, userId: string) {  
 const service = await this.prisma.service.findUnique({ where: { id } });  
 if (!service) {  
 throw new NotFoundException('Услуга не найдена');  
 }  
 if (service.providerId !== userId) {  
 throw new ForbiddenException('Нет доступа к изменению этой услуги');  
 }  
 const { rating, ...sanitizedData } = data as any;  
 return this.prisma.service.update({  
 where: { id },  
 data: sanitizedData,  
 include: {  
 provider: true,  
 reviews: true,  
 },  
 });  
}

Листинг 3.9 – Реализация метода «updateService»

### 3.6.9 Удаление услуги

Метод «removeService» в «ServicesService» предназначен для удаления услуги из базы данных. В методе есть проверка на то, что пользователь, удаляющий услугу, является пользователем роли «Администратор» или создателем услуги. Реализация метода представлена в листинге 3.10.

async removeService(id: string, userId: string, userType:string) {  
 const service = await this.findService(id);  
 if (service.providerId !== userId && userType !== 'ADMIN') {  
 throw new ForbiddenException('У вас нет прав на удаление этого сервиса');}  
 await this.prisma.service.delete({where: { id },});  
 return { message: 'Сервис успешно удален' };  
}

Листинг 3.10 – Реализация метода «removeService»

### 3.6.10 Общение с представителем услуги в чате

Для обеспечения обмена сообщениями между пользователями в режиме реального времени используется WebSocket и соответствующий функционал в «ChatGateway» и «ChatService».

Метод «handleMessage» в «ChatGateway» подписан на событие «send\_message». Он принимает объект, содержащий данные отправителя, получателя и текст сообщения. После получение данных вызывается «saveMessage» из «ChatService» для сохранения сообщения в базу данных. Получает идентификатор сокет-соединения получателя из внутренней коллекции. Если соединение существует, сообщение отправляется получателю через WebSocket по событию receive\_message. Методы представлены в листингах 3.11 и 3.12.

@SubscribeMessage('send\_message')  
async handleMessage(  
 @MessageBody() data: SendMessageDto,  
 @ConnectedSocket() client: Socket,  
) {  
 const message = await this.chatService.saveMessage(data);  
 const receiverSocketId = this.userSockets.get(data.receiverId);  
 *console*.log(`Received message from ${*JSON*.stringify(receiverSocketId)}`);  
 if (receiverSocketId) {  
 client.to(receiverSocketId).emit('receive\_message', message);  
 }  
 return message;  
}

Листинг 3.11 – Реализация метода «handleMessage»

async saveMessage(dto: SendMessageDto) {  
 return this.prisma.message.create({  
 data: {  
 senderId: dto.senderId,  
 receiverId: dto.receiverId,  
 content: dto.content,  
 },  
 });  
}

Листинг 3.12 – Реализация метода «saveMessage»

Метод «findByUsers» в «ChatService» возвращает полную историю переписки между двумя пользователями. Поиск выполняется по условию, что один пользователь является отправителем, а второй — получателем, и наоборот. Дополнительно загружаются данные об отправителе и получателе. Метод представлен в листинге 3.13.

async findByUsers(userId1: string, userId2: string) {  
 return this.prisma.message.findMany({  
 where: {  
 OR: [  
 { senderId: userId1, receiverId: userId2 },  
 { senderId: userId2, receiverId: userId1 },  
 ],  
 },  
 include: {  
 sender: true,  
 receiver: true,  
 },  
 });  
}

Листинг 3.13 – Реализация метода «findByUsers»

### 3.6.11 Бронирование услуги

Метод «createBooking» из «BookingsService» предназначен для создания нового бронирования услуги пользователем. Реализация метода представлена в листинге 3.14.

async createBooking(data: CreateBookingData) {

return this.prisma.booking.create({

data: {

serviceId: data.serviceId,

userId: data.userId,

date: data.date,

status: data.status,

},

include: {

service: true,

user: true,

},

});

}

Листинг 3.14 – Реализация метода «createBooking»

Дополнительно, для улучшения пользовательского опыта и информирования о процессе бронирования, в сервисе реализован метод sendOrderStatusChangeEmail, который отправляет email-уведомление о смене статуса бронирования пользователю. Реализация метода представлена в листинге 3.15.

async sendOrderStatusChangeEmail({

to,

userName,

serviceTitle,

newStatus,

providerName,

}: {

to: string;

userName: string;

serviceTitle: string;

newStatus: string;

providerName: string;

}) {

const subject = `Обновление статуса бронирования услуги "${serviceTitle}"`;

const text = `${userName}, забронированная вами услуга "${serviceTitle}", предоставляемая специалистом ${providerName}, изменила статус на: "${this.statusMap[newStatus]}".`;

try {

await this.transporter.sendMail({

from: `Платформа бронирования услуг`,

to,

subject,

text,

});

this.logger.log(`Письмо о статусе бронирования услуги "${serviceTitle}" отправлено на ${to}`);

} catch (error) {

this.logger.error('Ошибка при отправке письма:', error);

}

}

Листинг 3.15 – Реализация метода «sendOrderStatusChangeEmail»

### 3.6.12 Просмотр истории бронирований

Метод «findBookingsByUser» из «BookingsService» осуществляет получение истории бронирований пользователя с возможностью фильтрации по статусу и дате, а также с поддержкой пагинации. Возвращаются данные о бронированиях с привязанными услугами и пользователями, отсортированные по дате по убыванию. Реализация метода представлена в листинге 3.16.

async findBookingsByUser( userId: string, options?: {

status?: string;

startDate?: string;

endDate?: string;

page?: number;

limit?: number;},)

{

const where: any = { userId };

if (options?.status) {

where.status = options.status;

}

if (options?.startDate || options?.endDate) {

where.date = {};

if (options.startDate) where.date.gte = new Date(options.startDate);

if (options.endDate) where.date.lte = new Date(options.endDate);

}

const page = options?.page ?? 1;

const limit = options?.limit ?? 10;

const skip = (page - 1) \* limit;

const [items, total] = await Promise.all([

this.prisma.booking.findMany({

where,

include: { service: true, user: true },

skip,

take: limit,

orderBy: { date: 'desc' },

}),

this.prisma.booking.count({ where }),

]);

return {

total,

page,

limit,

items,

};

}

Листинг 3.16 – Реализация метода «findBookingsByUser»

### 3.6.13 Просмотр бронирований своей услуги

Метод «findBookingsByService» из «BookingsService» возвращает бронирования, связанные с конкретной услугой, с поддержкой фильтрации по статусу и дате, а также пагинацией. Результаты включают данные об услуге и пользователе, отсортированы по дате убыванию. Метод представлен в листинге 3.17.

async findBookingsByService( serviceId: string, options?: {  
 status?: string;  
 startDate?: string;  
 endDate?: string;  
 page?: number;  
 limit?: number;}) {  
 const where: any = { serviceId };  
  
 if (options?.*status*) {  
 where.status = options.*status*;  
 }  
 if (options?.*startDate* || options?.*endDate*) {  
 where.date = {};  
 if (options.*startDate*) {  
 where.date.gte = new *Date*(options.*startDate*);  
 }  
 if (options.*endDate*) {  
 where.date.lte = new *Date*(options.*endDate*);  
 }  
 }  
 const page = options?.*page* ?? 1;  
 const limit = options?.*limit* ?? 10;  
 const skip = (page - 1) \* limit;  
 const [items, total] = await *Promise*.all([  
 this.prisma.booking.findMany({  
 where,  
 include: {  
 service: true,  
 user: true,  
 },  
 skip,  
 take: limit,  
 orderBy: { date: 'desc' },  
 }),  
 this.prisma.booking.count({ where }),  
 ]);  
 return {  
 total,  
 page,  
 limit,  
 items,  
 };  
}

Листинг 3.17 – Реализация метода «findBookingsByService»

### 3.6.14 Изменение статуса бронирования

Метод «updateBooking» из «BookingsService» обновляет дату и статус бронирования по идентификатору. Если новый статус отличается от «PAID», пользователю отправляется уведомление на почту о смене статуса с указанием названия услуги и имени поставщика. Возвращается обновленное бронирование с вложенными данными о пользователе и услуге. Метод представлен в листинге 3.18.

async updateBooking(id: string, data: UpdateBookingData) {  
 const booking = await this.prisma.booking.update({  
 where: { id },  
 data: {  
 date: data.date,  
 status: data.status,  
 },  
 include: {  
 service: {  
 include: {  
 provider: true,  
 },  
 },  
 user: true,  
 },  
 });  
 if (data.*status* && data.*status* !== 'PAID') {  
 await this.mailService.sendOrderStatusChangeEmail({  
 to: booking.user.email,  
 userName: booking.user.firstName,  
 serviceTitle: booking.service.title,  
 newStatus: data.status,  
 providerName: `${booking.service.provider.firstName} ${booking.service.provider.lastName}`,  
 });  
 }  
 return booking;  
}

Листинг 3.18 – Реализация метода «updateBooking»

### 3.6.15 Оплата выполненной услуги

Метод «createCheckoutSession» из «StripeService» инициирует процесс оплаты услуги через Stripe, подготавливая сессию оплаты на основе переданных параметров — идентификатора бронирования и суммы. В процессе он извлекает данные о бронировании, в частности email пользователя, чтобы использовать его в качестве контактного адреса для отправки чека и других уведомлений. Также в Stripe-сессию включаются настройки продукта, информация о платеже, email для отправки квитанции и ссылки для перенаправления пользователя в случае успешной или отменённой оплаты. Метод представлен в листинге 3.19.

async createCheckoutSession(data: {  
 bookingId: string;  
 amount: number;  
}) {  
 const booking = await this.bookingsService.findBooking(data.*bookingId*);  
 return this.stripe.checkout.*sessions*.create({  
 payment\_method\_types: ['card'],  
 line\_items: [  
 {  
 price\_data: {  
 currency: 'usd',  
 product\_data: {  
 name: `Booking #${data.*bookingId*}`,  
 },  
 unit\_amount: data.*amount* \* 100,  
 },  
 quantity: 1,  
 },  
 ],  
 mode: 'payment',  
 metadata: {  
 bookingId: data.*bookingId*,  
 },  
 success\_url: 'http://localhost:3001/bookings',  
 cancel\_url: 'http://localhost:3001/',  
 customer\_email: booking.user.email,  
 payment\_intent\_data: {  
 receipt\_email: booking.user.email,  
 },  
 invoice\_creation: {  
 enabled: true,  
 },  
 });  
}

Листинг 3.19 – Реализация метода «createCheckoutSession»

Метод «handleWebhook» служит обработчиком событий от Stripe. Когда Stripe присылает уведомление о завершении сессии оплаты (тип события checkout.session.completed), метод извлекает метаданные сессии, включая идентификатор бронирования. После этого он запрашивает информацию о выставленном счёте (инвойсе), получая ссылку на его PDF-версию. Далее происходит отправка письма с этой ссылкой пользователю, а статус соответствующего бронирования в базе данных обновляется на «PAID», фиксируя факт успешной оплаты. Метод представлен в листинге 3.20.

async handleWebhook(event: Stripe.Event) {  
 if (event.type === 'checkout.session.completed') {  
 const session = event.data.object as Stripe.Checkout.Session;  
 const bookingId = session.metadata.bookingId;  
 const booking = await this.bookingsService.findBooking(bookingId);  
 const invoice = await this.stripe.invoices.retrieve(session.invoice as string);  
 await this.mailService.sendInvoiceLink({  
 to: booking.user.email,  
 pdfUrl: invoice.invoice\_pdf,  
 invoiceNumber: invoice.number,  
 userName: booking.user.firstName + ' ' + booking.user.lastName,  
 serviceTitle: `Бронирование #${bookingId}`  
 });  
 await this.bookingsService.updateBooking(bookingId, { status: 'PAID' });  
 }  
}

Листинг 3.20 – Реализация метода «handleWebhook»

### 3.6.16 Добавление отзыва на услугу

Метод «createReview» из «ReviewsService» реализует добавление нового отзыва на услугу. В методе присутствует валидация оценки отзыва и валидный идентификатор услуги. Реализация метода приведена в листинге 3.21.

async createReview(data: {  
 rating: number;  
 comment: string;  
 serviceId: string;  
 userId: string;  
}) {  
 const service = await this.prisma.service.findUnique({  
 where: { id: data.*serviceId* },  
 });  
 if (!service) {  
 throw new NotFoundException('Сервис не найден');  
 }  
 if (data.*rating* < 1 || data.*rating* > 5) {  
 throw new BadRequestException('Рейтинг должен быть между 1 и 5');  
 }  
 const review = await this.prisma.review.create({  
 data,  
 include: { service: true, user: true },  
 });  
 await this.updateServiceRating(data.*serviceId*);  
 return review;  
}

Листинг 3.21 – Реализация метода «createReview»

### 3.6.17 Удаление отзыва на услугу

Реализация удаления тега представлена методом «removeReview» в «ReviewsService». Можно удалять отзыв только его создателю. Реализация метода представлена в листинге 3.22.

async removeReview(id: string, userId: string) {  
 const review = await this.findOne(id);  
 if (review.userId !== userId) {  
 throw new ForbiddenException('Вы можете удалить только свои отзывы');  
 }  
 await this.prisma.review.delete({  
 where: { id },  
 });  
 await this.updateServiceRating(review.serviceId);  
 return { message: 'Отзыв успешно удалён' };  
}

Листинг 3.22 – Реализация метода «removeReview»

### 3.6.18 Изменение информации о пользователе

Метод «updateUser» из «UsersService» позволяет изменить данные пользователя по его идентификатору. Входные параметры включают имя, фамилию, номер телефона и путь к фотографии профиля. После выполнения обновления возвращается актуальная информация. Метод представлен в листинге 3.23.

async updateUser(id: string, data: {  
 firstName?: string;  
 lastName?: string;  
 phone?: string;  
 profilePhotoPath?: string;  
}) {  
  
 return this.prisma.user.update({  
 where: { id },  
 data,  
 select: {  
 id: true,  
 email: true,  
 firstName: true,  
 lastName: true,  
 phone: true,  
 userType: true,  
 profilePhotoPath: true,  
 createdAt: true,  
 },  
 });  
}

Листинг 3.23 – Реализация метода «updateUser»

### 3.6.19 Просмотра списка пользователей

Метод «findAllUsers» из «UsersService» выполняет постраничный просмотр пользователей с возможностью фильтрации по email и статусу блокировки. Запрос возвращает список пользователей с основными полями и метаинформацией о пагинации: текущая страница, лимит, общее количество и число страниц. Данные упорядочены по дате создания в порядке убывания. Метод представлен в листинге 3.24.

async findAllUsers(page: number, limit: number, filters: UserFilters) {  
 const skip = (page - 1) \* limit;  
 const where: any = {};  
 if (filters.email) {  
 where.email = { contains: filters.email, mode: 'insensitive'};  
 }  
 if (filters.isBlocked !== undefined) {  
 where.isBlocked = filters.isBlocked;  
 }  
  
 const [users, total] = await this.prisma.$transaction([  
 this.prisma.user.findMany({  
 skip,  
 take: limit,  
 where,  
 orderBy: { createdAt: 'desc' },  
 select: {  
 id: true,  
 email: true,  
 firstName: true,  
 lastName: true,  
 phone: true,  
 userType: true,  
 profilePhotoPath: true,  
 isBlocked: true,  
 createdAt: true,  
 },  
 }),  
 this.prisma.user.count({ where }),  
 ]);  
 return {  
 data: users,  
 meta: {  
 total,  
 page,  
 limit,  
 totalPages: *Math*.ceil(total / limit),  
 },  
 };  
}

Листинг 3.24 – Реализация метода «findAllUsers»

### 3.6.20 Блокирование и разблокирование пользователя

Метод «toggleBlockUser» обновляет статус блокировки пользователя по его идентификатору. В зависимости от переданного значения, пользователь либо блокируется, либо разблокируется. Возвращается обновленная информация: идентификатор, email, имя, фамилия и текущий статус блокировки. Метод представлен в листинге 3.25.

async toggleBlockUser(id: string, isBlocked: boolean) {  
 return this.prisma.user.update({where: { id }, data: { isBlocked },  
 select: {id: true,  
 email: true,  
 firstName: true,  
 lastName: true,  
 isBlocked: true,  
 },  
 });  
}

Листинг 3.25 – Реализация метода «toogleBlockUser»

### 3.6.21 Удаление пользователя

Метод «removeUser» из «UserServices» удаляет пользователя из базы данных. Благодаря каскадному удалению также удаляются все созданные им бронирования, услуги и отзывы. Метод представлен в листинге 3.26.

async removeUser(id: string) {  
 await this.findUser(id);  
  
 await this.prisma.user.delete({  
 where: { id },  
 });  
 return { message: 'Пользователь успешно удален' };  
}

Листинг 3.26 – Реализация метода «removeUser»

## 3.7 Структура клиентской части

Файлы проекта сгруппированы по их назначению. В директории src размещён основной программный код. В таблице 3.6 приведён список директорий проекта.

Таблица 3.6 – Описание директорий проекта клиентской части

|  |  |
| --- | --- |
| Директория | Описание |
| api | Содержит настройку конфигурации axios, используемой для обработки каждого запроса и ответа |
| components | Компоненты, которые могут быть переиспользованы в разных частях приложения и представляют собой обособленные единицы с логикой, не выходящие за пределы их назначения |
| hooks | Пользовательские хуки, которые были использованы в приложении |
| pages | Содержит страницы, которые включают в себя компоненты, а также отдельную логику, написанную для самой страницы |
| data | Содержит константные данные, которые используются в проекте |

Для составления пользовательского интерфейса на react используются компоненты, перечень компонентов, разработанных для клиентской части перечислен в таблице 3.7

Таблица 3.7 – Описание компонентов приложения

|  |  |
| --- | --- |
| Компонент | Описание |
| AddReviewModal | Компонент, модальное окно добавления отзыва |
| BookingDialog | Компонент, модальное окно создания бронирования |
| BookingList | Компонент, история бронирований клиента |
| Catalog | Компонент, каталог услуг |
| ChatComponent | Компонент, пользовательский чат |
| Header | Компонент навигационное меню, подключается на каждой странице |
| LoginForm | Компонент, форма аутентификации |
| Profile | Компонент, профиль пользователя |
| RegisterForm | Компонент, форма регистрации |
| ServiceBookingsList | Компонент, список бронирований услуги |
| ServiceCard | Компонент, карточка товара |
| ServiceDetails | Компонент, детали услуги |
| ServiceForm | Компонент для создания и изменения услуги |
| UserServices | Компонент, услуги клиента |
| UserListComponent | Компонент, список всех пользователей |

Окончание таблицы 3.7

|  |  |
| --- | --- |
| Компонент | Описание |
| BookingsPage | Страница истории бронирований клиента |
| CatalogPage | Страница каталога всех услуг с поиском и фильтрацией |
| ChatPage | Страница пользовательского чата |
| FavoritesPage | Страница списка избранных услуг |
| LoginPage | Страница для аутентификации пользователя |
| ManageServicePage | Страница для создания или изменения услуги |
| ProfilePage | Страница профиля пользователя с возможностью изменения данных |
| RegisterPage | Страница для регистрации |
| ServiceBookingsPage | Страница истории бронирований для услуги |
| ServiceDetailsPage | Страница для деталей об услуге и отзывах на нее |
| UserServicesPage | Страница списка услуг, созданных клиентом для управления |
| UsersPage | Страница списка всех пользователей для администратора |

## 3.8 Использование Docker

В проекте используется Docker для контейнеризации приложения, что упрощает развёртывание. Проект состоит из 3 контейнеров: серверная часть, клиентская часть и база данных.

Для сборки Docker-контейнера с серверной частью использовался образ «node:22.8.0-alpine», который включает в себя Node.js версии 22.8.0. Alpine [26] – легковесный дистрибутив Linux [27], которое занимает меньше места и потребляет меньше ресурсов.

Контейнер с серверной частью работает на порту 3000. Dockerfile серверной части представлен на листинге 3.27.

FROM node:22.8.0-alpine AS builder

WORKDIR /app

COPY package\*.json ./

RUN npm install

COPY prisma ./prisma

RUN npx prisma generate

RUN npm install -g @nestjs/cli

COPY . .

RUN npm run build

RUN ls -la /app/dist && ls -la /app/dist/prisma

FROM node:22.8.0-alpine

WORKDIR /app

COPY --from=builder /app/dist ./dist

COPY --from=builder /app/package\*.json ./

COPY --from=builder /app/prisma ./prisma

RUN npm install --omit=dev

RUN npx prisma generate

COPY docker-entrypoint.sh ./

RUN chmod +x docker-entrypoint.sh

ENV NODE\_ENV=production

EXPOSE 3000

ENTRYPOINT ["./docker-entrypoint.sh"]

Листинг 3.27 – Dockerfile серверной части

Для сборки клиентской части приложения использовался мультистадийный Dockerfile. На этапе сборки применён образ node:20-bullseye, содержащий Node.js версии 20 и основанный на Debian Bullseye — стабильном дистрибутиве Linux, обеспечивающем хорошую совместимость с большинством npm-пакетов. В этом этапе происходит установка зависимостей, компиляция TypeScript и сборка проекта с помощью Vite.

На этапе запуска используется образ nginx:alpine, лёгкий веб-сервер на базе Alpine Linux, который позволяет минимизировать размер итогового контейнера и ускорить его загрузку. Готовые файлы из каталога сборки копируются в директорию, обслуживаемую Nginx, после чего сервер запускается на порту 80.

Dockerfile клиентской части представлен в листинге 3.28.

FROM node:20-bullseye AS *build*RUN groupadd -r appgroup && useradd -r -u 1001 -g appgroup appuser  
WORKDIR /app  
COPY --chown=appuser:appgroup package\*.json ./  
RUN npm cache clean --force && \  
 npm install --legacy-peer-deps  
COPY --chown=appuser:appgroup . .  
RUN npx tsc && npx vite build  
  
FROM nginx:alpine  
COPY --from=*build* /app/dist /usr/share/nginx/html  
COPY nginx/nginx.conf /etc/nginx/conf.d/default.conf  
EXPOSE 80  
CMD ["nginx", "-g", "daemon off;"]

Листинг 3.28 – Dockerfile клиентской части

Для удобства работы с контейнерами используется Docker Compose, который представлен файлом «docker-compose.yml», где прописаны два сервера и база данных, а также переопределены переменные окружения. Для сохранения базы данных при перезапусках, создан volume postgres\_data.

У контейнера с серверной частью переопределена точка входа, перед запуском процесса сервера запускается «npm run seed», который инициализирует базу данных начальными данными. Содержимое «docker-compose.yml» представлено в листинге 3.29.

services:

db:

image: postgres:15

container\_name: service\_db

restart: always

environment:

POSTGRES\_DB: service\_db

POSTGRES\_USER: service\_user

POSTGRES\_PASSWORD: service\_pass

ports:

- "5432:5432"

volumes:

- postgres\_data:/var/lib/postgresql/data

networks:

- service\_network

backend:

build:

context: /backend

dockerfile: Dockerfile

container\_name: service\_backend

restart: always

volumes:

- ./backend/src/uploads:/app/dist/src/uploads

env\_file:

- /backend/.env

environment:

DATABASE\_URL: postgres://service\_user:service\_pass@db:5432/service\_db

NODE\_ENV: production

ports:

- "3000:3000"

depends\_on:

- db

networks:

- service\_network

frontend:

build:

context: ./frontend

dockerfile: Dockerfile

env\_file:

- ./frontend/.env.production

container\_name: frontend

ports:

- "3001:80"

networks:

- service\_network

depends\_on:

- backend

volumes:

postgres\_data:

networks:

service\_network:

driver: bridge

Листинг 3.29 – Содержимое «docker-compose.yml»

## 3.9 Использование Nginx

Для обеспечения единой точки входа в веб-приложение и эффективного маршрутизации запросов был использован веб-сервер Nginx [28], выполняющий роль обратного прокси-сервера. Nginx обеспечивает разделение клиентской и серверной логики: статика, сгенерированная React-приложением, обслуживается непосредственно веб-сервером, тогда как API-запросы и обращения к файловым ресурсам проксируются на серверную часть, реализованную на NestJS.

Все HTTP-запросы по корневому пути / обслуживаются статическими файлами из каталога React-приложения. Запросы, направленные по пути /api/, проксируются на адрес серверной части по порту 3000. Отдельно обрабатываются запросы к загруженным файлам по пути /uploads/, которые также перенаправляются к backend-серверу, при этом используется директива rewrite для корректной маршрутизации.

Для поддержки WebSocket-соединений в конфигурацию Nginx добавлен блок location /socket.io/, который проксирует запросы на backend-сервер с обязательной установкой заголовков Upgrade, Connection и использованием HTTP/1.1 для обеспечения двустороннего постоянного соединения. Такая настройка позволяет корректно обрабатывать запросы WebSocket, обеспечивая работу функций обмена сообщениями и уведомлений в реальном времени между клиентом и сервером. Листинг конфигурационного файла Nginx представлен в листинге 3.30.

server {  
 listen 80;  
 server\_name localhost;  
  
 location / {  
 root /usr/share/nginx/html;  
 index index.html index.htm;  
 try\_files $uri $uri/ /index.html;  
 }  
 location /uploads/ {  
 rewrite ^/uploads/(.\*) /api/uploads/$1 break;  
 proxy\_pass http://backend:3000;  
 proxy\_set\_header Host $host;  
 proxy\_set\_header X-Real-IP $remote\_addr;  
 proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;  
 }  
  
 location /api/ {  
 proxy\_pass http://backend:3000;  
 proxy\_set\_header Host $host;  
 proxy\_set\_header X-Real-IP $remote\_addr;  
 }  
 location /socket.io/ {  
 proxy\_pass http://backend:3000;  
 proxy\_http\_version 1.1;  
 proxy\_set\_header Upgrade $http\_upgrade;  
 proxy\_set\_header Connection "upgrade";  
 proxy\_set\_header Host $host;  
 proxy\_set\_header X-Real-IP $remote\_addr;  
 proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;  
 }  
 }

Листинг 3.30 – Содержимое файла конфигурации Nginx

## 3.12 Выводы по разделу

* cервер разработан на платформе Node.js с использованием фреймворка Nest..js.
* клиентская часть разработана с использованием библиотеки React.
* в качестве СУБД использован PostgreSQL.
* используется веб-сервер Nginx.
* используется контейнеризация Docker.
* структура приложения основана на модульном подходе с применением современных библиотек для клиентской и серверной частей;
* для передачи да5нных используется JSON для текстовой информации. Для работы с медиа используется FormData и multer;
* реализованы все функции для всех ролей: гостя, клиента, администратора. Количество функций веб-приложения составило 22.

# 4 Тестирование веб-приложения

## 4.1 Функциональное тестирование

Для проверки корректности работы всех функций разработанного веб- приложения было проведено ручное тестирование, описание и итоги которого представлены в таблице 4.1.

Таблица 4.1 – Результаты тестирования функций

|  |  |  |  |
| --- | --- | --- | --- |
| Функция веб-приложения | Описание тестирования | Ожидаемый результат | Статус теста |
| Регистрация | Под ролью «Гость» перейти на страницу регистрации(/register), в поле «Email» ввести адрес электронной почты, в поле «Пароль» (от 6 до 64 символов), поля «Имя» и «Фамилия» должны быть непустыми, в поле «Телефон» ввести номер телефона в соответствии с выражением  «+375 25|29|33| 44XXXXXXX», затем нажать кнопку «ЗАГРУЗИТЬ АВАТАР», выбрать в открывшемся окне изображение и нажать кнопку «ЗАРЕГИСТРИРОВАТЬСЯ». | Пользователю выводится сообщение об успешной регистрации и происходит переадресация на страницу «/login». | Пройден |
| Аутентификация | Под ролью «Гость» перейти на странницу аутентификации (/login), в поле «Email» ввести Email существующего пользователя, в поле «Пароль» ввести пароль существующего пользователя и нажать кнопку «ВОЙТИ». | Пользователь перенаправлен на страницу каталога(/). | Пройден |
| Просмотр, фильтрация и поиск услуг | Под ролью «Гость» или «Клиент» или «Администратор», перейти на страницу каталога (/), выбрать из выпадающего списка категорий необходимую и в поисковой строке ввести строку. | На странице будет отображён список услуг, удовлетворяющих выбранной категории и строке, введенной в строке поиска. | Пройден |

Продолжение таблицы 4.1

|  |  |  |  |
| --- | --- | --- | --- |
| Функция веб-приложения | Описание тестирования | Ожидаемый результат | Статус теста |
| Просмотр информации об услуге | Под любой ролью перейти на страницу каталога (/) и для интересующей услуги нажать кнопку «Подробнее». | Переход на страницу услуги и отображение информации об этой услуги. | Пройден |
| Добавление услуги в избранное | Под ролью «Клиент», перейти на страницу каталога (/) и для интересующей услуги (не находящейся в избранном для текущего пользователя) нажать кнопку в форме сердца. Затем перейти на страницу «ИЗБРАННОЕ». | На открывшейся странице отображён список избранных услуг, включая добавленную. | Пройден |
| Удаление услуги в избранное | Под ролью «Клиент», перейти на страницу каталога (/) и для интересующей услуги (находящейся в избранном для текущего пользователя) нажать кнопку в форме сердца. Затем перейти на страницу «ИЗБРАННОЕ». | На открывшейся странице отображён список избранных услуг, не включая добавленную. | Пройден |
| Добавление услуги | Под ролью «Клиент», перейти на страницу «МОИ УСЛУГИ», нажать кнопку «ДОБАВИТЬ УСЛУГУ». В открывшейся форме создания новой услуги при корректном заполнении всех обязательных полей и нажатии кнопки «ЗАГРУЗИТЬ ФОТО» и выборе изображения нажать кнопку «СОЗДАТЬ УСЛУГУ». | Происходит переход на страницу каталога (/), где можно найти созданную услугу. | Пройден |
| Изменение услуги | Под ролью «Клиент», перейти на страницу «МОИ УСЛУГИ», нажать кнопку «РЕДАКТИРОВАТЬ» для своей услуги. В открывшейся форме редактирования услуги при корректном редактировании интересующих полей нажать кнопку «СОХРАНИТЬ ИЗМЕНЕНИЯ». | Происходит переход на страницу каталога (/), где можно найти отредактированную услугу. | Пройден |

Продолжение таблицы 4.1

|  |  |  |  |
| --- | --- | --- | --- |
| Функция веб-приложения | Описание тестирования | Ожидаемый результат | Статус теста |
| Удаление услуги | Под ролью «Клиент», перейти на страницу «МОИ УСЛУГИ», нажать кнопку «УДАЛИТЬ» для своей услуги и подтвердить удаление этой услуги. | На странице «МОИ УСЛУГИ» не будет отображаться удаленная услуга. | Пройден |
| Общение с представителем услуги в чате | Под ролью «Клиент» перейти на страницу каталога (/) и для чужой услуги нажать кнопку «Подробнее». На открывшейся странице нажать кнопку «Написать», ввести сообщение в поле ввода и нажать клавишу «Enter». | Сообщение будет отправлено другому пользователю и выведено в чате с этим пользователем. | Пройден |
| Бронирование услуги | Под ролью «Клиент» перейти на страницу каталога (/) и для чужой услуги нажать кнопку «Подробнее». На открывшейся странице нажать кнопку «Забронировать», выбрать дату и время и нажать кнопку «ЗАБРОНИРОВАТЬ». | На странице «БРОНИРОВАНИЯ» (/bookings) будет отображена забронированная услуга со статусом «В ожидании» | Пройден |
| Просмотр истории личных бронирований | Под ролью «Клиент» перейти на страницу «БРОНИРОВАНИЯ» (/bookings). | На открывшейся странице будет выведена история бронирований услуг. | Пройден |
| Просмотр истории бронирований своей услуги | Под ролью «Клиент», перейти на страницу «МОИ УСЛУГИ», нажать кнопку «ИСТОРИЯ БРОНИРОВАНИЙ». | На открывшейся странице будет выведена история бронирований данной услуги. | Пройден |
| Изменение статуса бронирований услуги | Под ролью «Клиент», перейти на страницу «МОИ УСЛУГИ», нажать кнопку «ИСТОРИЯ БРОНИРОВАНИЙ». Затем нажать кнопку «ЗАВЕРШИТЬ» для необходимой забронированной услуги со статусом «ПОДТВЕРЖДЕНО». | Будет выведено сообщение «Действие выполнено успешно» и статус забронированной услуги будет изменен на «ЗАВЕРШЕНО» | Пройден |

Продолжение таблицы 4.1

|  |  |  |  |
| --- | --- | --- | --- |
| Функция веб-приложения | Описание тестирования | Ожидаемый результат | Статус теста |
| Оплата выполненной услуги | Под ролью «Клиент» перейти на страницу «БРОНИРОВАНИЯ» (/bookings). Для забронированной услуги со статусом «Завершено» нажать кнопку «ОПЛАТИТЬ». Далее на открывшейся странице ввести необходимые платежные данные и нажать кнопку «Оплатить». | Статус брони изменится на «Оплачено» и на адрес электронной почты пользователя придет письмо со ссылкой на чек в формате PDF. | Пройден |
| Добавление отзыва на услугу | Под ролью «Клиент» перейти на страницу «БРОНИРОВАНИЯ» (/bookings). Для услуг со статусом «Оплачено» нажать кнопку «ОСТАВИТЬ ОТЗЫВ». Далее указать количество звезд от 1 до 5, ввести комментарий и нажать кнопку «ОПУБЛИКОВАТЬ ОТЗЫВ». | На странице данной услуги будет отображен отзыв и измененный рейтинг. | Пройден |
| Удаление отзыва на услугу | Под ролью «Клиент» перейти на страницу каталога (/) и для интересующей услуги нажать кнопку «Подробнее». Для отзыва, оставленным текущим пользователем, нажать кнопку «УДАЛИТЬ». | Отзыв, удаленный пользователем, исчез. | Пройден |
| Изменение информации о пользователе | Под ролью «Клиент» перейти на страницу профиля текущего пользователя (/profile) и нажать на кнопку «СМЕНИТЬ ПАРОЛЬ». Далее необходимо ввести текущий пароль, новый пароль от 6 до 64 символов, продублировать его в следующее поле, нажать кнопку «СМЕНИТЬ ПАРОЛЬ». | Выведено сообщение «Пароль успешно изменен». | Пройден |

Окончание таблицы 4.1

|  |  |  |  |
| --- | --- | --- | --- |
| Функция веб-приложения | Описание тестирования | Ожидаемый результат | Статус теста |
| Просмотр списка пользователей | Под ролью «Администратор» перейти на страницу «ПОЛЬЗОВАТЕЛИ» (/users). | Будет отображен список всех пользователей веб-приложения. | Пройден |
| Блокирование пользователя | Под ролью «Администратор» перейти на страницу «ПОЛЬЗОВАТЕЛИ» (/users) и выбрать незаблокированного пользователя с ролью «Клиент». Далее на открывшейся странице нажать кнопку «ЗАБЛОКИРОВАТЬ». | Будет выведено сообщение «Пользователь успешно заблокирован» и кнопка «ЗАБЛОКИРОВАТЬ» изменится на «РАЗБЛОКИРОВАТЬ». | Пройден |
| Разблокирование пользователя | Под ролью «Администратор» перейти на страницу «ПОЛЬЗОВАТЕЛИ» (/users) и выбрать заблокированного пользователя с ролью «Клиент». Далее на открывшейся странице нажать кнопку «РАЗБЛОКИРОВАТЬ». | Будет выведено сообщение «Пользователь успешно разблокирован» и кнопка «РАЗБЛОКИРОВАТЬ» изменится на «ЗАБЛОКИРОВАТЬ». | Пройден |
| Блокирование пользователя | Под ролью «Администратор» перейти на страницу «ПОЛЬЗОВАТЕЛИ» (/users) и выбрать необходимого пользователя с ролью «Клиент». Далее на открывшейся странице нажать кнопку «УДАЛИТЬ ПОЛЬЗОВАТЕЛЯ». | Будет выведено сообщение «Пользователь успешно удален» и открыта страница пользователей. | Пройден |

# Выводы по разделу

* тестирование проводилось ручным методом. Проведено тестирование всех 22 функций приложения. Все операции работают корректно;
* покрытие тестами веб-приложения составляет 100%;
* все основные функции приложения, работают в соответствии с требованиями;
* система безопасности обеспечивает корректную работу с ролями пользователей («Гость», «Клиент» и «Администратор»);
* все эндпоинты правильно обрабатывают запросы и возвращают ожидаемые результаты.

# 5 Руководство пользователя

## 5.1 Регистрация

Пользователь для доступа к основному функционалу веб-приложения должен сначала создать аккаунт. По нажатии на кнопку «ЗАРЕГИСТРИРОВАТЬСЯ» пользователь попадает на страницу регистрации. Форма регистрации представлена на 5.1.



Рисунок 5.1 – Форма регистрации

Для успешной регистрации пользователю необходимо ввести уникальный email, пароль, имя, фамилию, номер телефона и загрузить аватар, после чего он нажимает кнопку «ЗАРЕГИСТРИРОВАТЬСЯ».

## 5.2 Аутентификация

После регистрации пользователь перенаправляется на форму аутентификации, где ему необходимо ввести email и пароль, затем нажать на кнопку «ВОЙТИ» и он получит доступ к основному функционалу веб-приложения. Форма аутентификации представлена на рисунке 5.2.

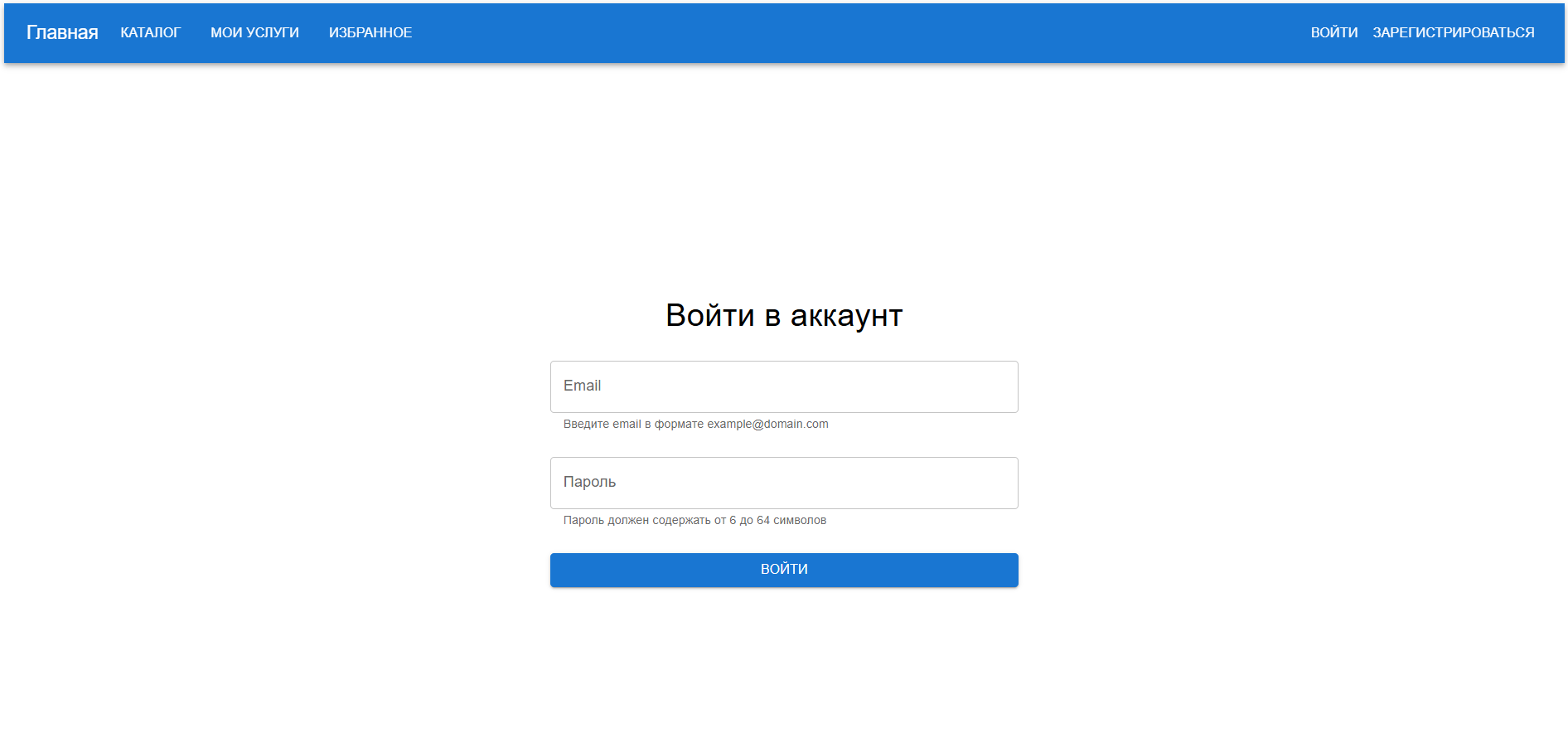


Рисунок 5.2 – Форма аутентификации

5.3 Просмотр, поиск и фильтрация услуг

После успешной аутентификации пользователь попадает на каталог, сверху находится навигационное меню, откуда можно попасть в профиль либо выйти из аккаунта. Также можно произвести навигацию по другим страницам. По центру сверху находится поисковая строка и фильтры, чуть ниже сами услуги и снизу кнопки для пагинации. Интерфейс каталога представлен на рисунке 5.3.

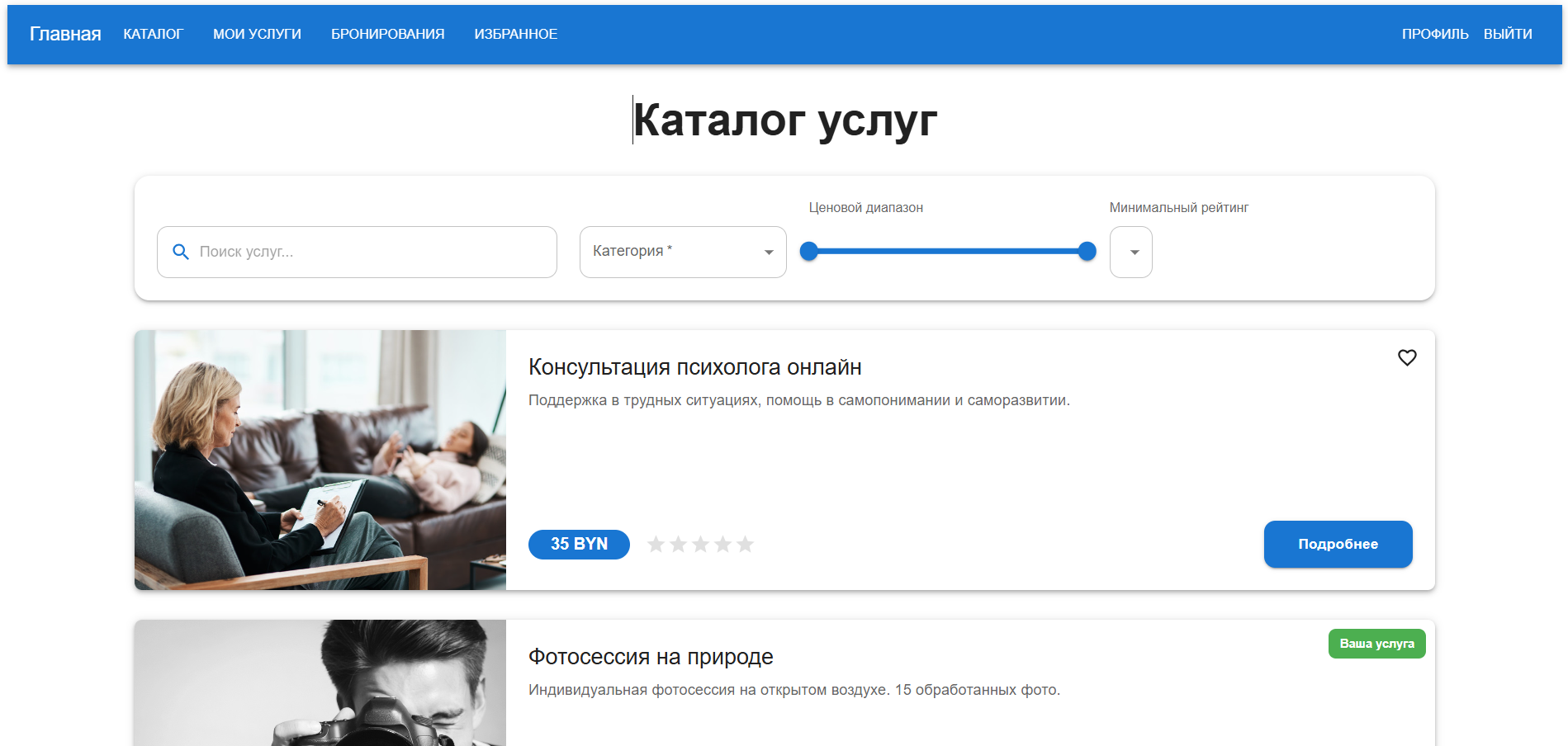


Рисунок 5.3 – Каталог веб-приложения

Услуги пользователя подписаны в верхнем правом углу карточки.

5.4 Просмотр информации об услуге

Пользователь может нажать у определенной услуги кнопку «Подробнее» для того, чтобы узнать больше информации о товаре. После нажатия пользователь перейдет на страницу, представленную на рисунке 5.4, с подробным описанием товара и отзывами.

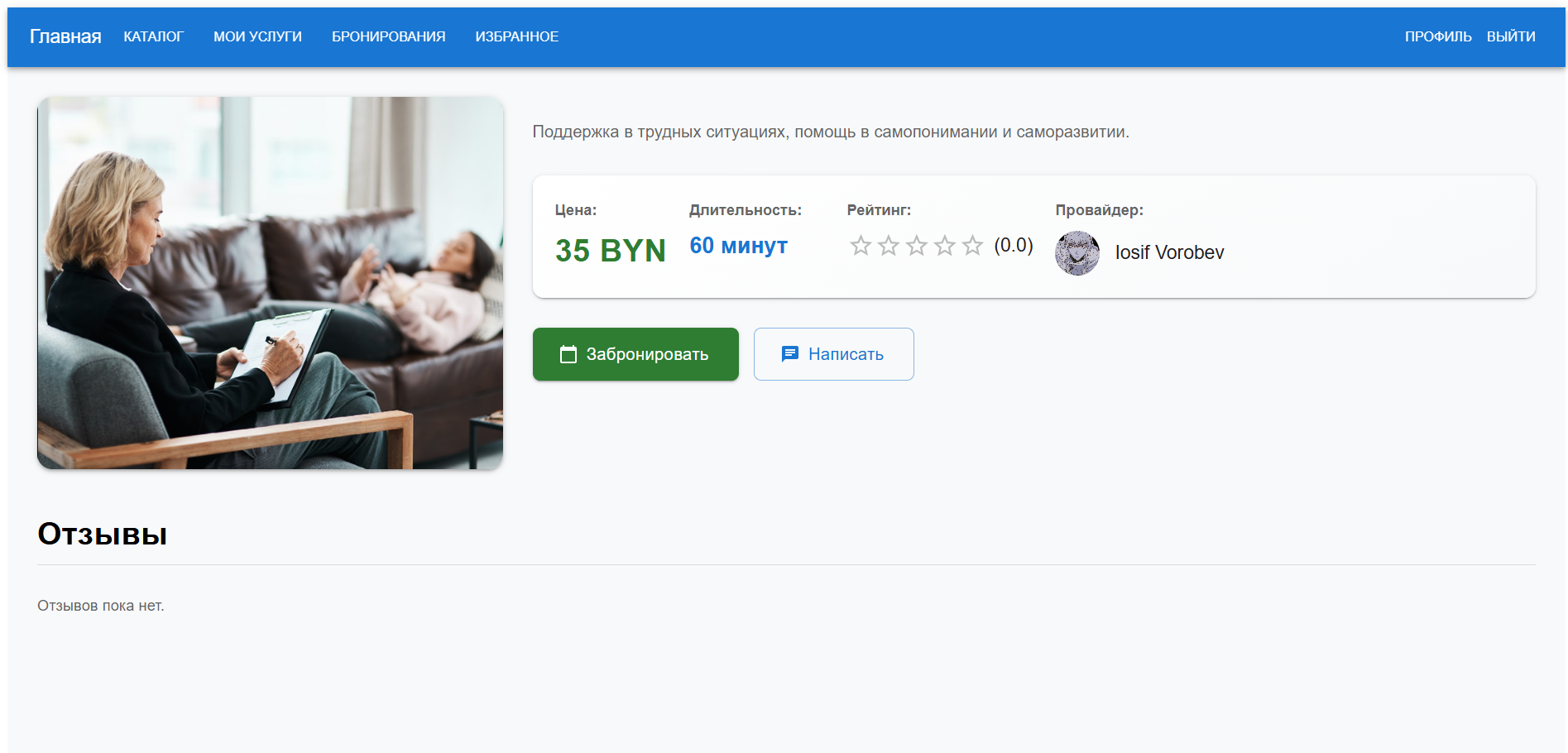


Рисунок 5.4 – Каталог веб-приложения

## 5.5 Добавление услуги в избранное

Чтобы добавить услугу в избранное, надо нажать на сердечко рядом с ней в каталоге. На рисунке 5.5 красным цветом выделена кнопка для добавления услуги в избранное.

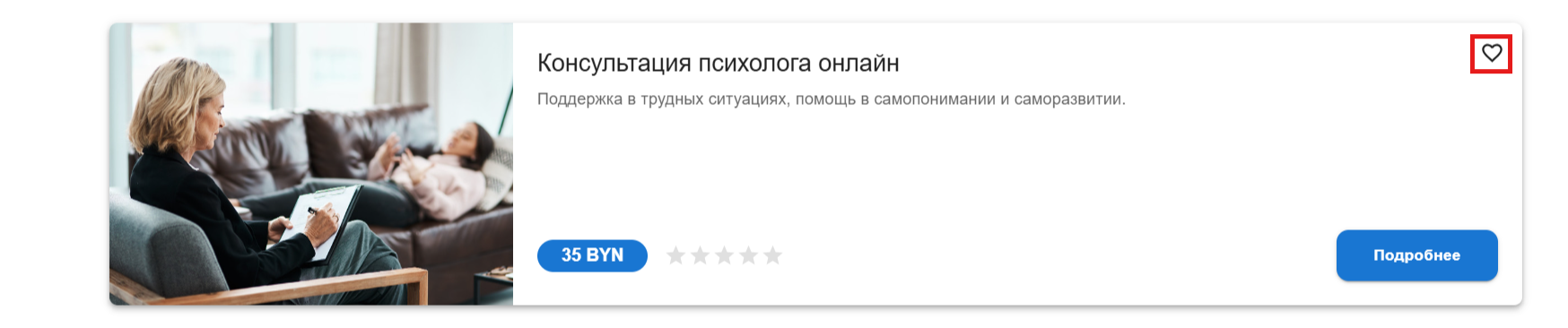


Рисунок 5.5 – Кнопка добавления услуги в избранное

## 5.6 Удаление услуги из избранного

После добавления услуги в избранное, иконка сердечка станет закрашенной, при повторном нажатии услуга удалится из избранного. Сделать это можно как из каталога, так со страницы избранных товаров. На рисунке 5.6 представлена закрашенная кнопка избранного.

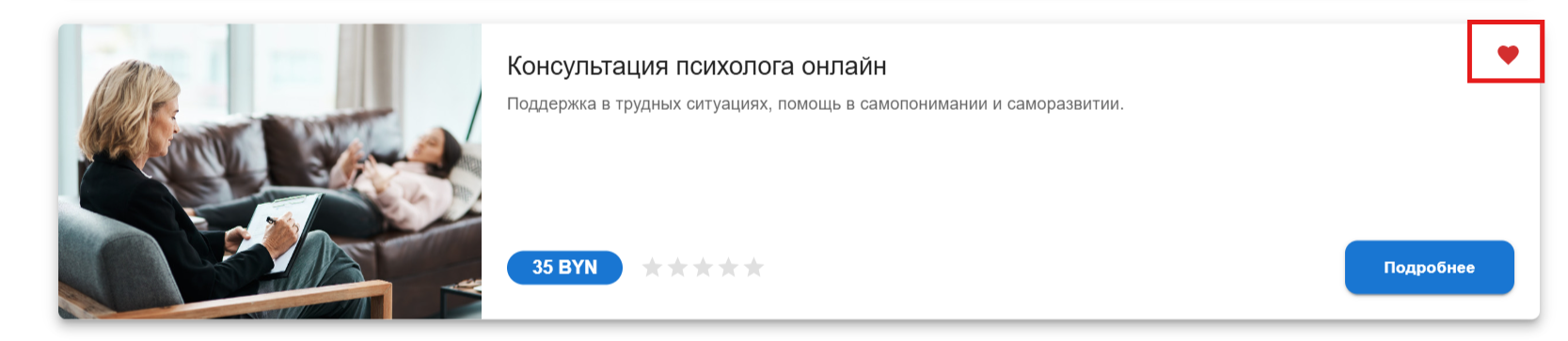


Рисунок 5.6 – Кнопка удаление услуги из избранного

## 5.7 Добавление услуги

Получить доступ к своим услугам пользователь может при переходе по кнопке в шапке «МОИ УСЛУГИ» на соответствующую страницу. Страница представлена на рисунке 5.7.

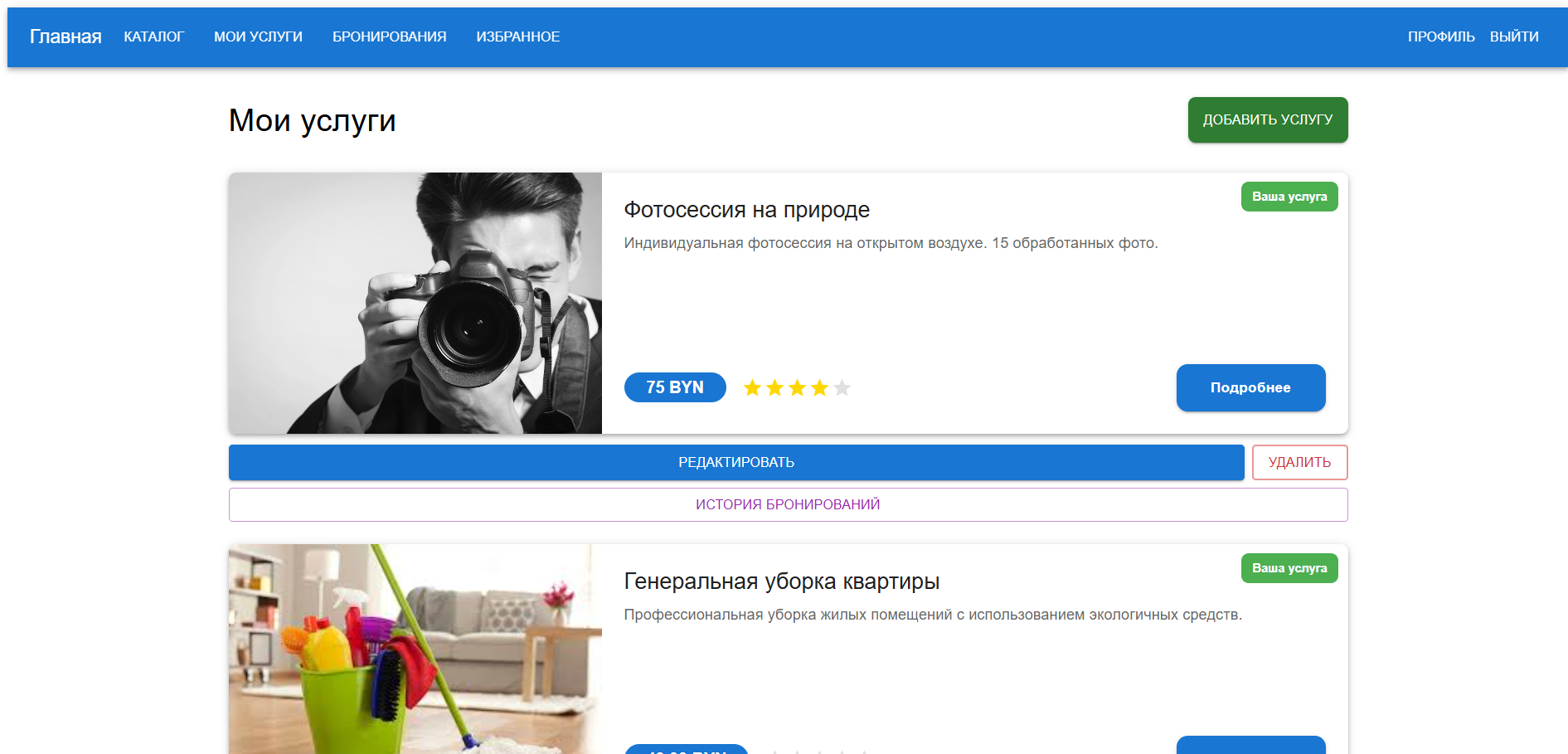


Рисунок 5.7 – Страница услуг текущего пользователя

Пользователь может нажать на кнопку «ДОБАВИТЬ УСЛУГУ», после чего откроется страница, на которой представлена форма для заполнения информации об услуге: название, описание, цена, длительность в минутах, категория и фотография. Все поля обязательны к заполнению. Форма добавления услуги представлена на рисунке 5.8.

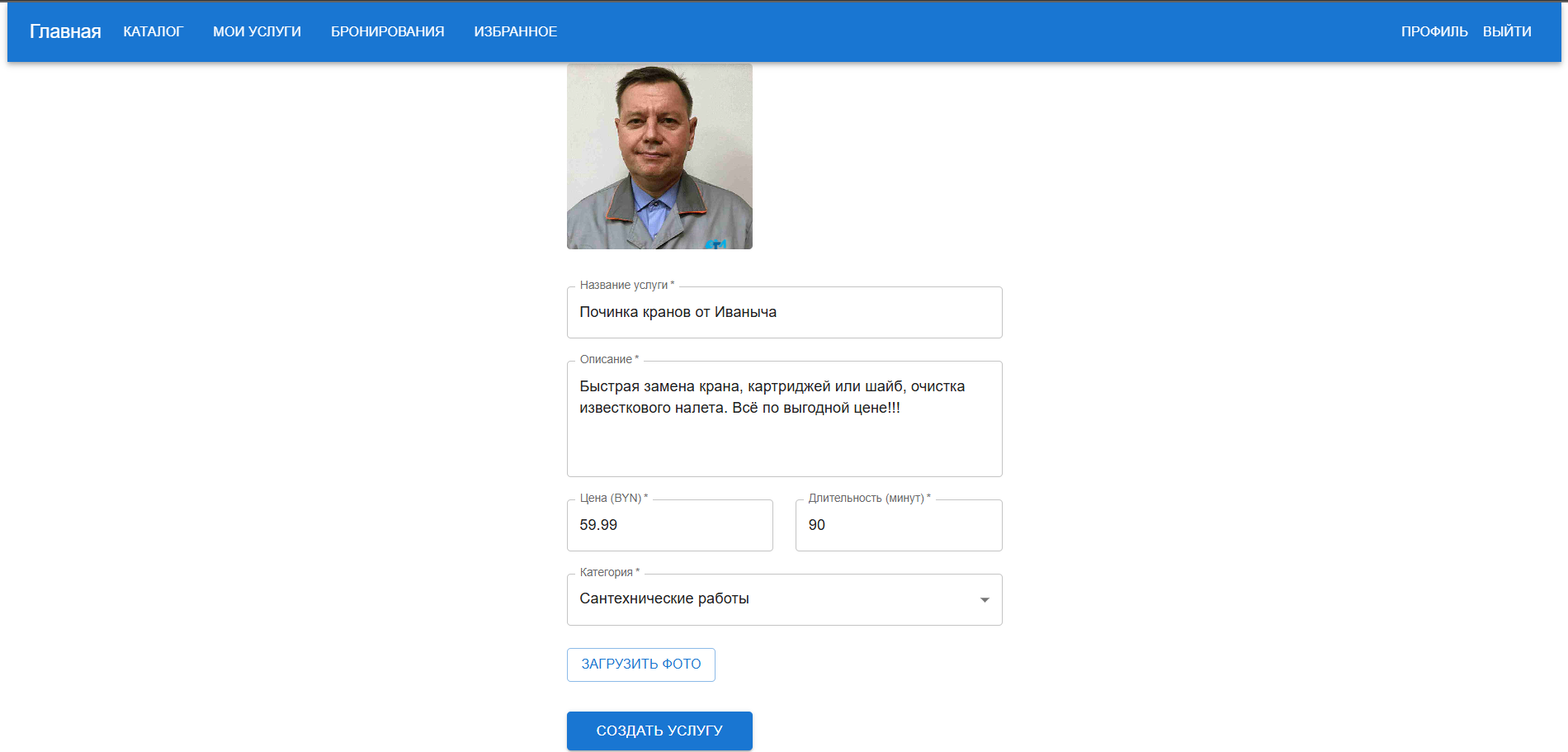


Рисунок 5.8 – Форма для добавления услуги

После нажатия на кнопку «СОЗДАТЬ УСЛУГУ», пользователь попадает в каталог, где может увидеть свою новую услугу, представленную на рисунке 5.9.

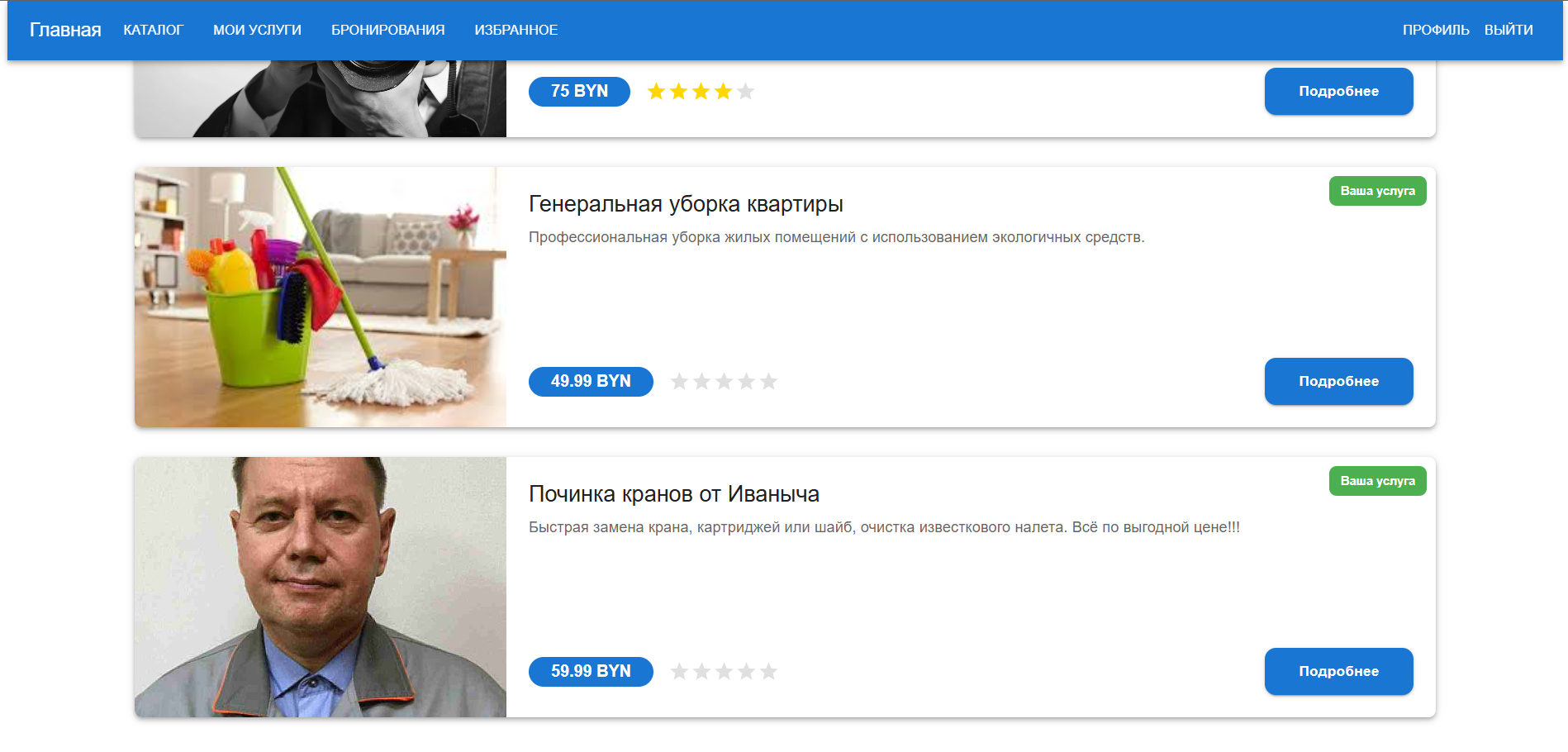


Рисунок 5.9 – Добавленная услуга

## 5.8 Изменение услуги

На странице «МОИ УСЛУГИ» пользователь может нажать кнопку «РЕДАКТИРОВАТЬ» у определенной услуги для того, чтобы открыть форму редактирования услуги, представленную на рисунке 5.10.

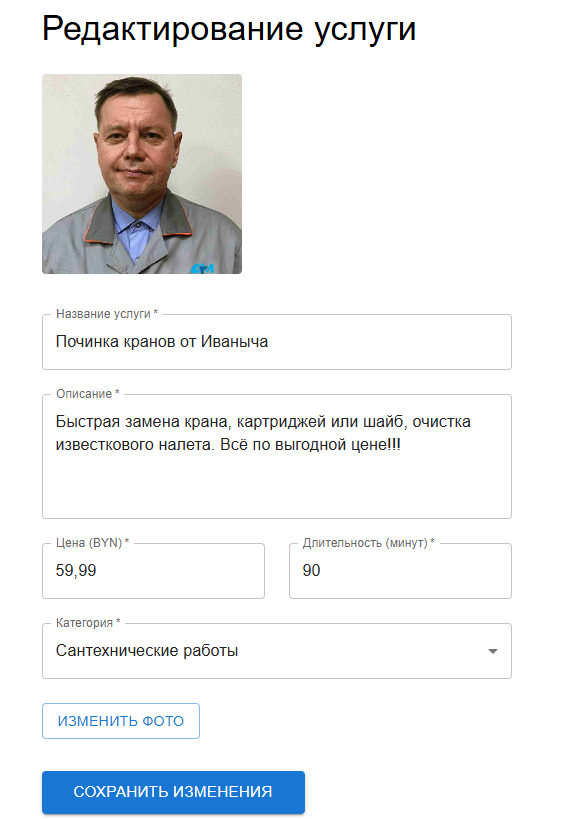


Рисунок 5.10 – Форма редактирования услуги

При нажатии на кнопку «СОХРАНИТЬ ИЗМЕНЕНИЯ», информация об услуге изменится.

## 5.9 Удаление услуги

Кроме добавления и редактирования, пользователь так же может удалить свою услугу, нажав на кнопку «УДАЛИТЬ», после нажатия которой услуга будет удалена навсегда. Пример кнопки представлен на рисунке 5.11 справа снизу.

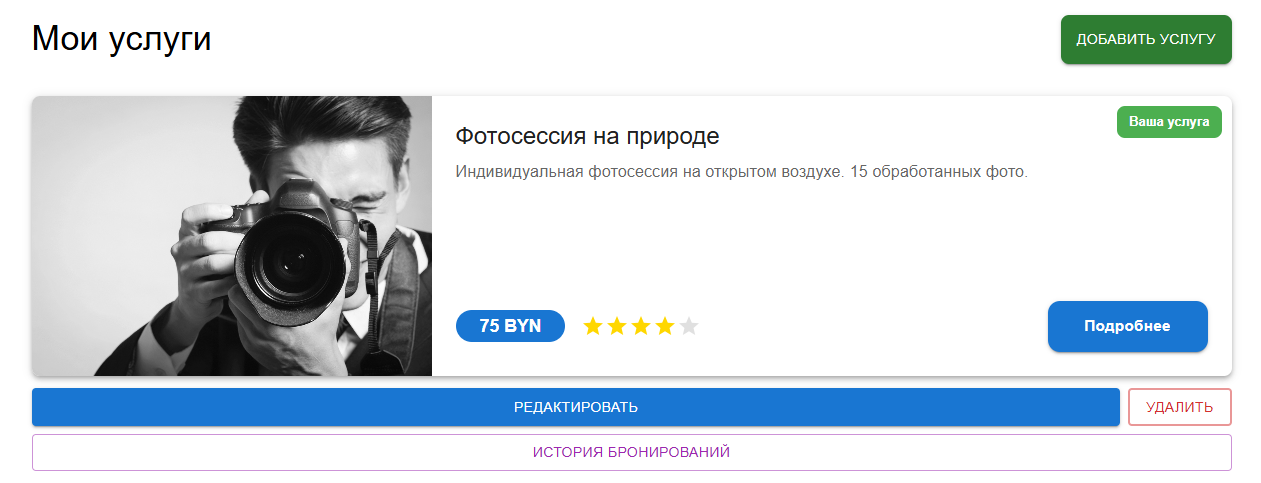


Рисунок 5.11 – Пример услуги с кнопкой удаления

## 5.10 Общение с представителем услуги в чате

На странице любой услуги, которая не принадлежит пользователю, есть кнопка «Написать», пример этой кнопки представлен на рисунке 5.12.

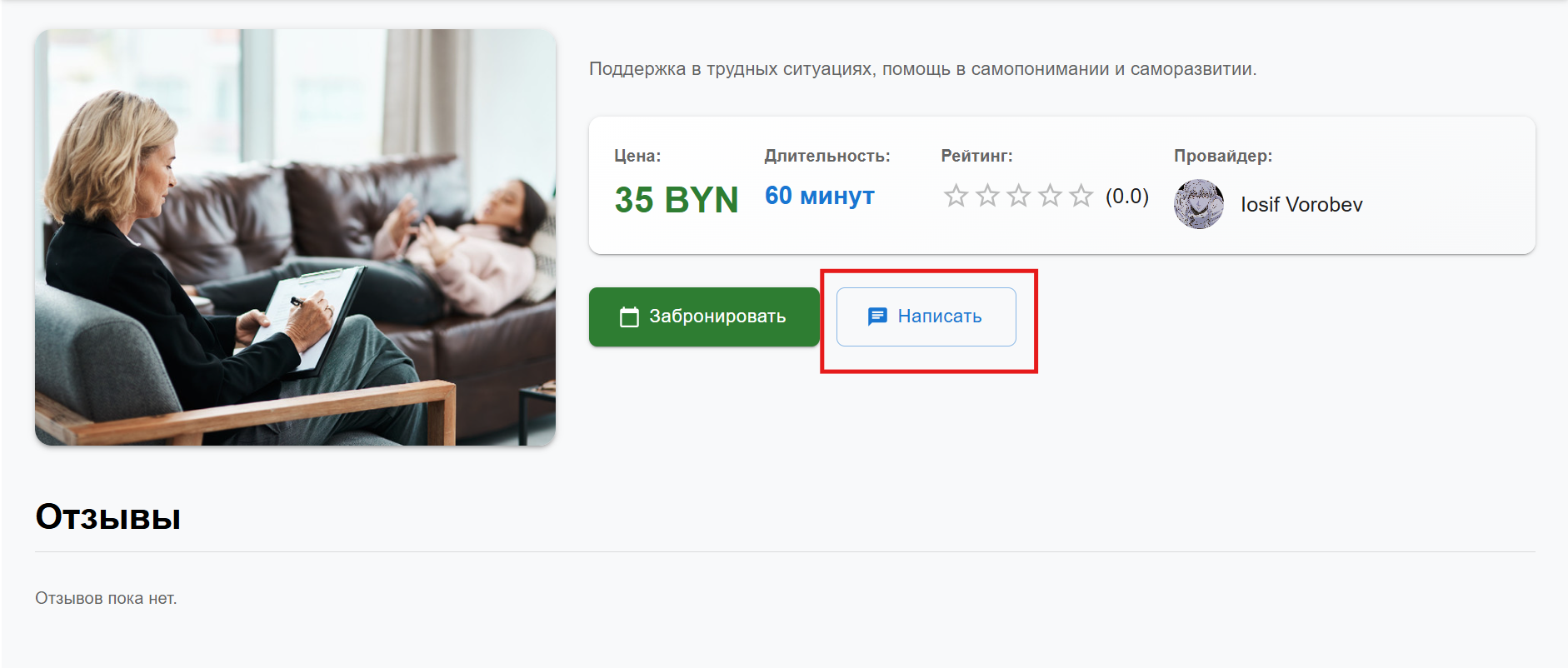


Рисунок 5.12 – Кнопка открытия чата

После этого открывается окно чата с представителем услуги, в котором они могут обмениваться сообщениями в реальном времени. Чат представлен на рисунке 5.13.

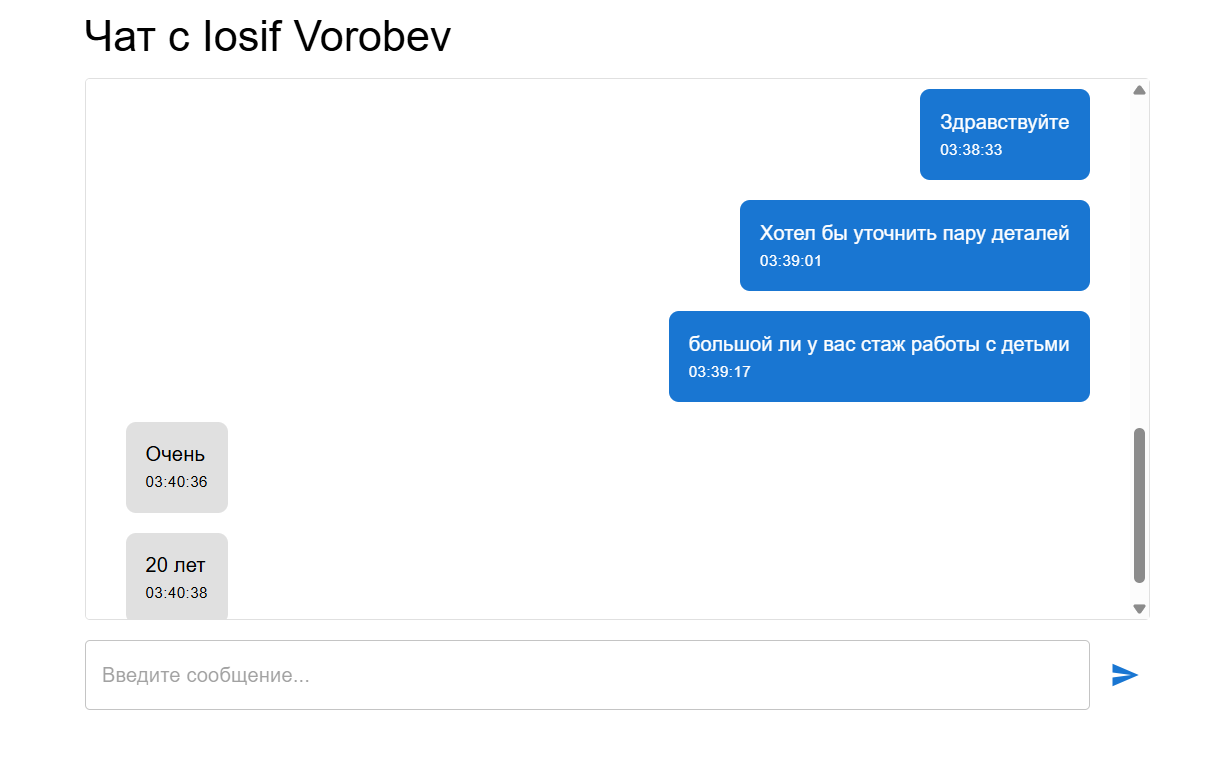


Рисунок 5.13 – Чат

## 5.11 Бронирование услуги

На все той же странице услуги пользователь может нажать кнопку «Забронировать», после чего появляется модальное окно, представленное на рисунке 5.14.

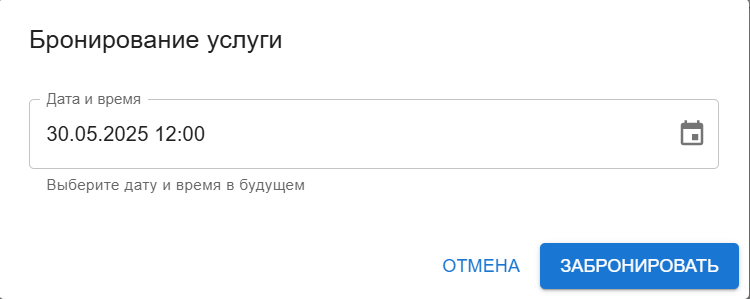


Рисунок 5.14 – Модальное окно бронирования услуги

Пользователь выбирает удобную дату, время и нажимает кнопку «ЗАБРОНИРОВАТЬ».

## 5.12 Просмотр истории личных бронирований

После успешного бронирования пользователь может отслеживать ее статус в своей истории бронирований, на которую можно перейти при нажатии на кнопку в шапке «БРОНИРОВАНИЯ». Данная страница представлена на рисунке 5.15.

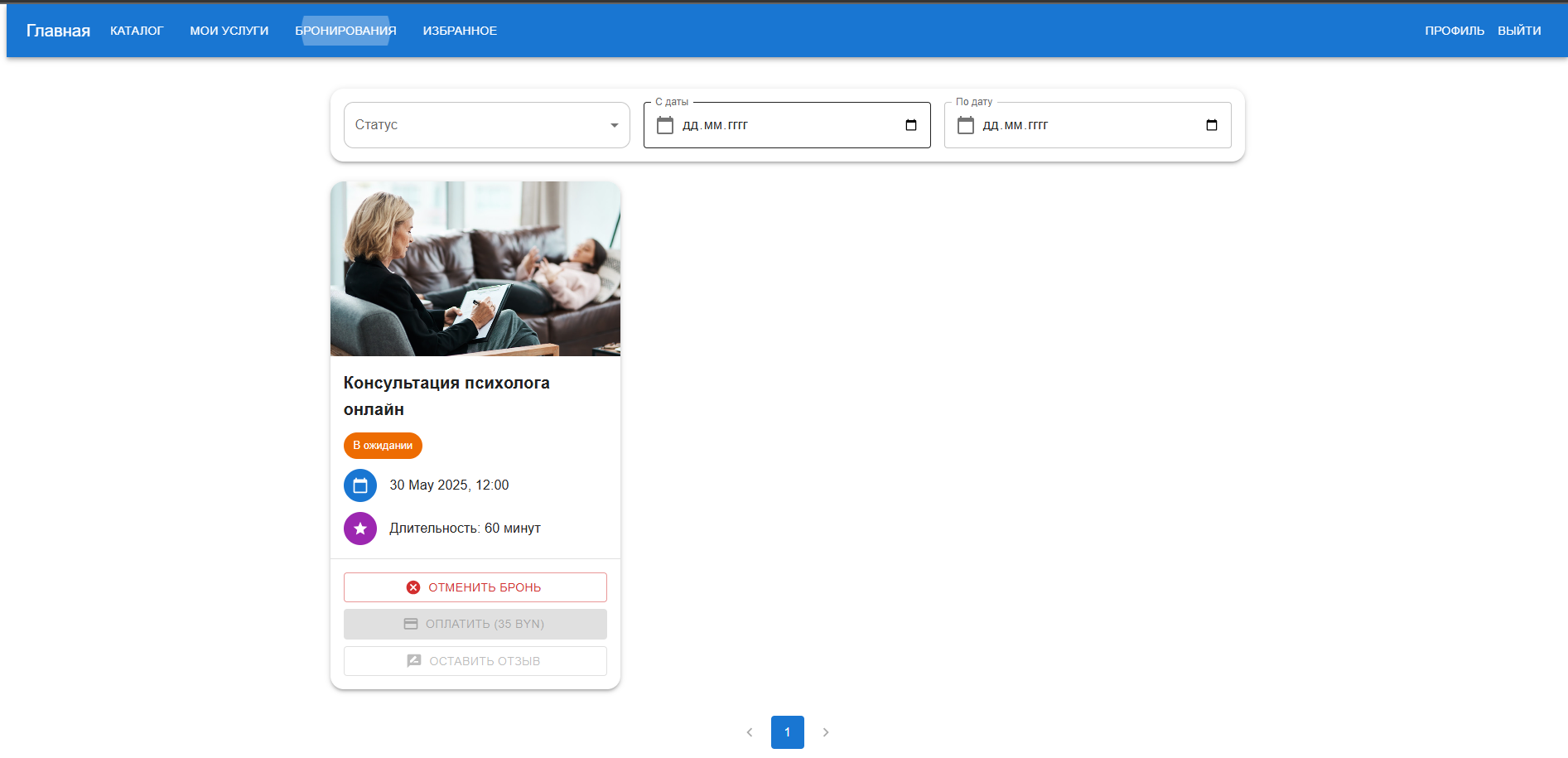


Рисунок 5.15 – Страница истории личных бронирований услуг пользователя

## 5.13 Просмотр бронирований своей услуги

Представитель услуги так же может отслеживать статус всех бронирований своей услуги, перейдя по кнопке «ИСТОРИЯ БРОНИРОВАНИЙ» на соответствующую страницу, представленную на рисунке 5.16

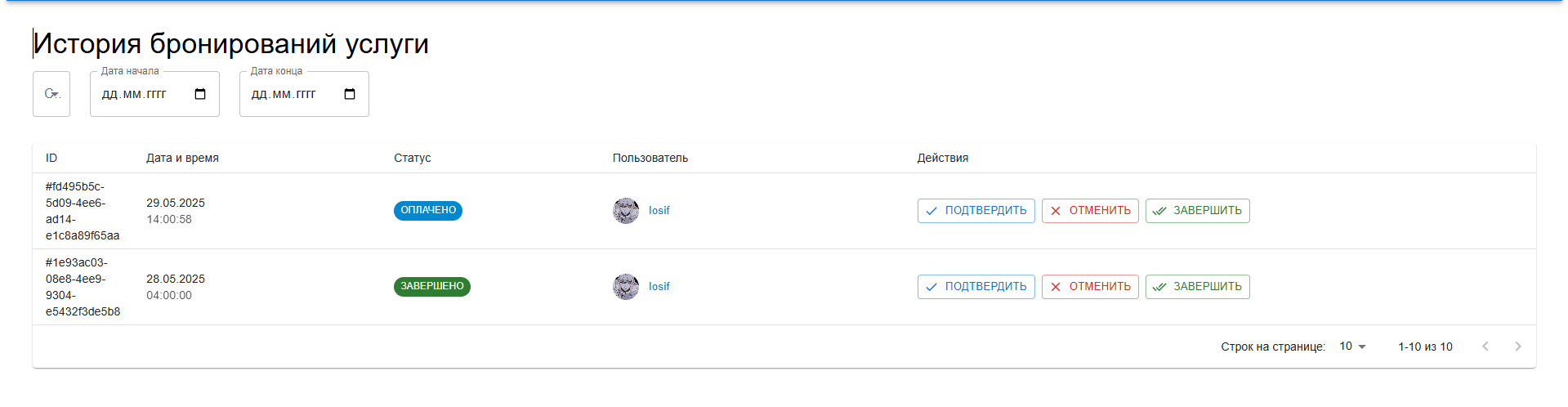


Рисунок 5.16 – История бронирований услуги

## 5.14 Изменение статуса бронирования

Изначально заказ со стороны исполнителя имеет статус «В ожидании», так же есть статуса «Подтверждено», «Отклонено», «Завершено» и «Оплачено». На рисунке 5.17 представлены все кнопки для изменения статуса бронирования со стороны представителя услуги, а на рисунке 5.18 – со стороны пользователя, которому ее оказывают.

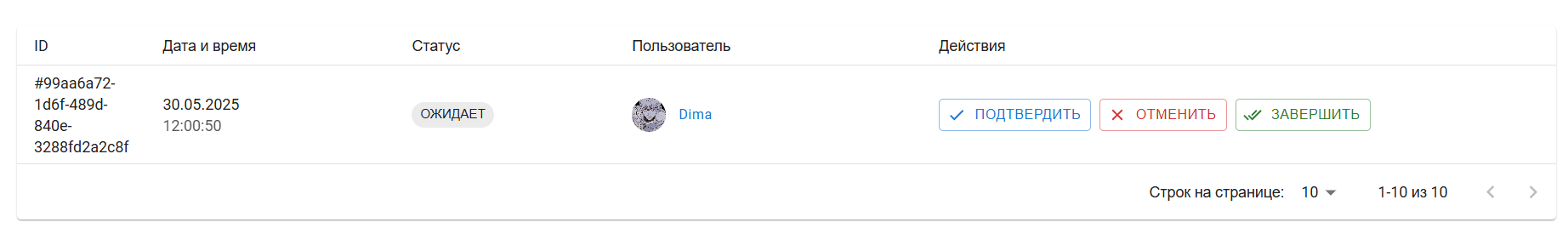


Рисунок 5.17 – Кнопки для изменения статуса представителем услуги

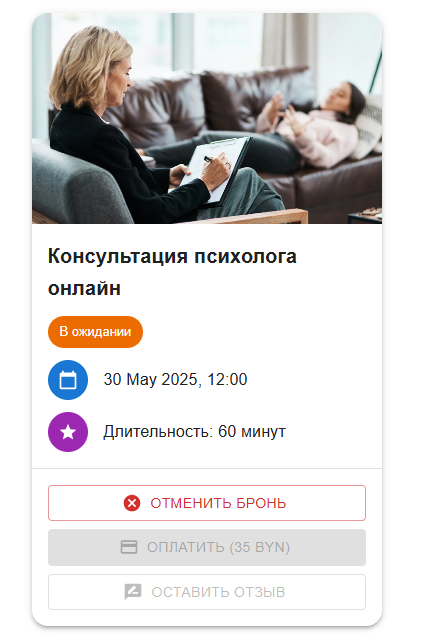


Рисунок 5.18 – Кнопки для изменения статуса обычным пользователем

## 5.15 Оплата выполненной услуги

После завершения заказа, пользователь должен оплатить услугу, которую ему предоставили. После нажатия на кнопку «ОПЛАТИТЬ», пользователь вводит свои платежные данные, после чего он возвращается на страницу, а на его email приходит чек об оплате. Пример окна для оплаты представлено на рисунке 5.19.

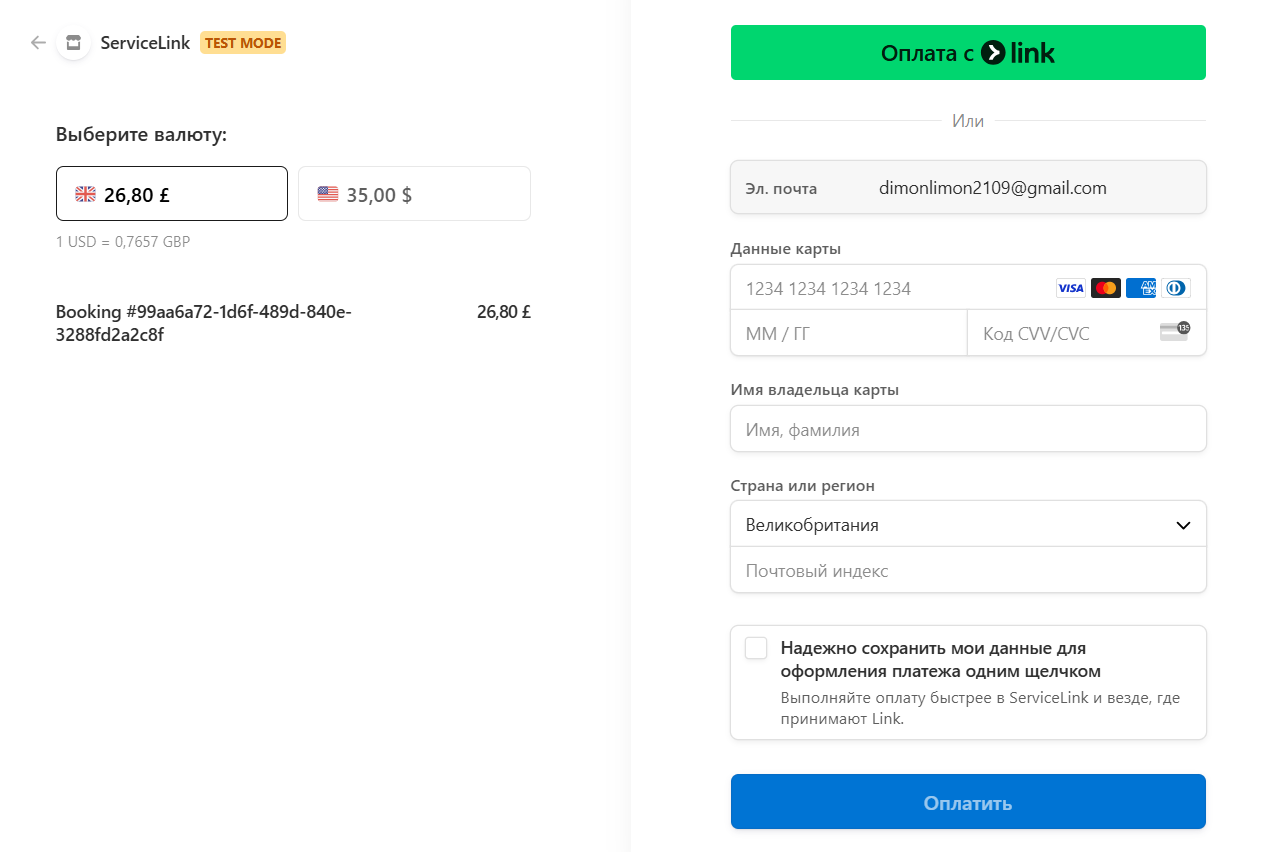


Рисунок 5.19 – Окно для оплаты услуги

## 5.16 Добавление и удаление отзывов на услугу

После оплаты пользователь может оставить отзыв на услугу, модальное окно представлено на рисунке 5.20.

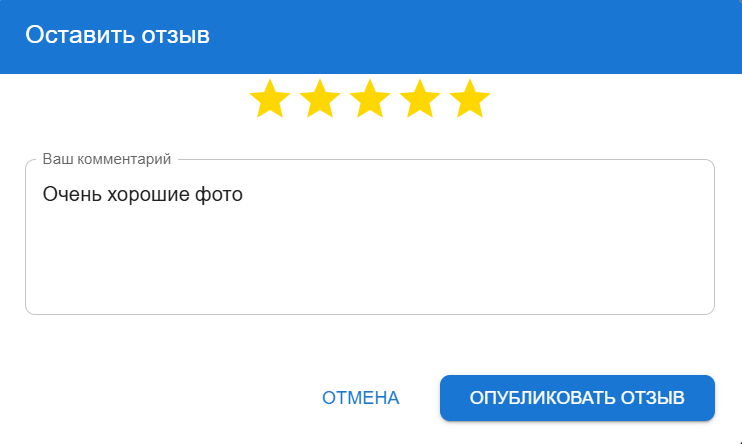


Рисунок 5.20 – Модальное окно для написания отзыва

После нажатия на кнопку «ОПУБЛИКОВАТЬ ОТЗЫВ», отзыв попадает на страницу услуги, но пользователь может передумать и удалить его с помощью кнопки «УДАЛИТЬ» справа. Пример отзыва представлен на рисунке 5.21.

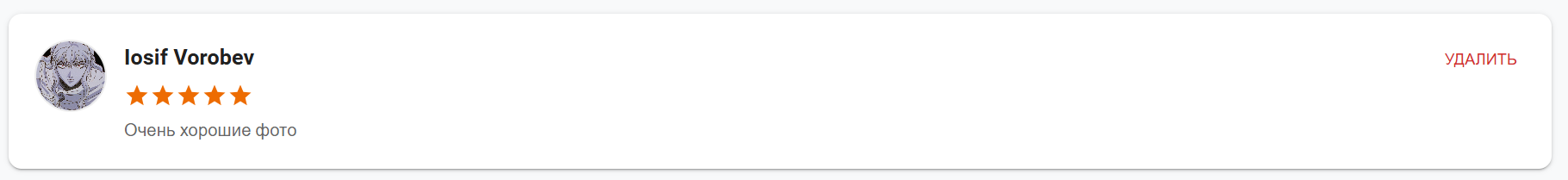


Рисунок 5.21 – Пример отзыва

## 5.17 Профиль

Пользователь может перейти в свой профиль при нажатии на соответствующую кнопку в шапке. Там пользователь может изменить свой пароль, введя старый, новый и повторно, или изменить свои личные данные и аватар. Профиль, форма изменения пароля и изменения личных данных представлены на рисунках 5.22, 5.23 и 5.24 соответственно.

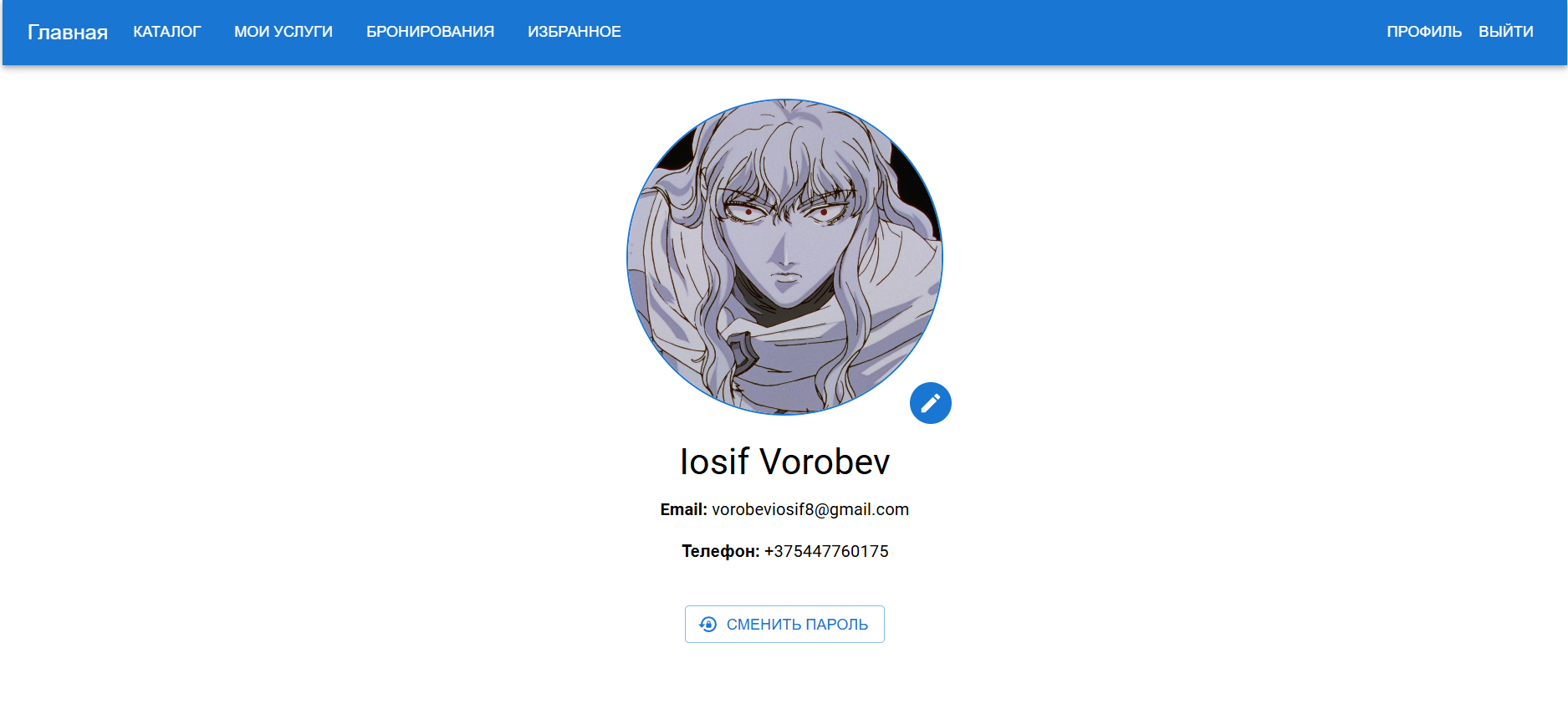


Рисунок 5.22 – Профиль пользователя

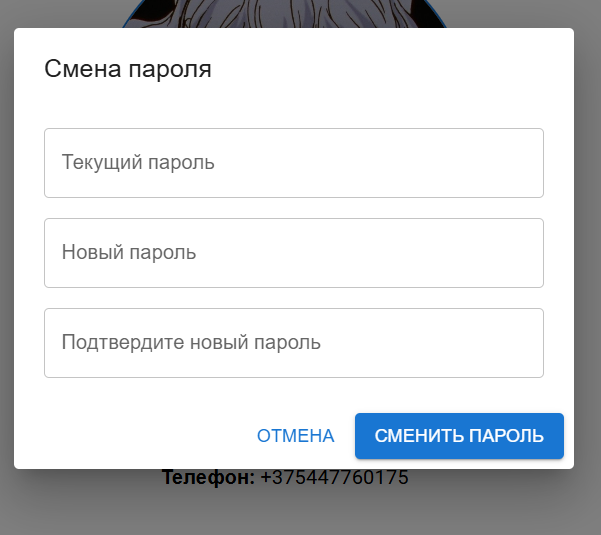


Рисунок 5.23 – Форма изменения пароля

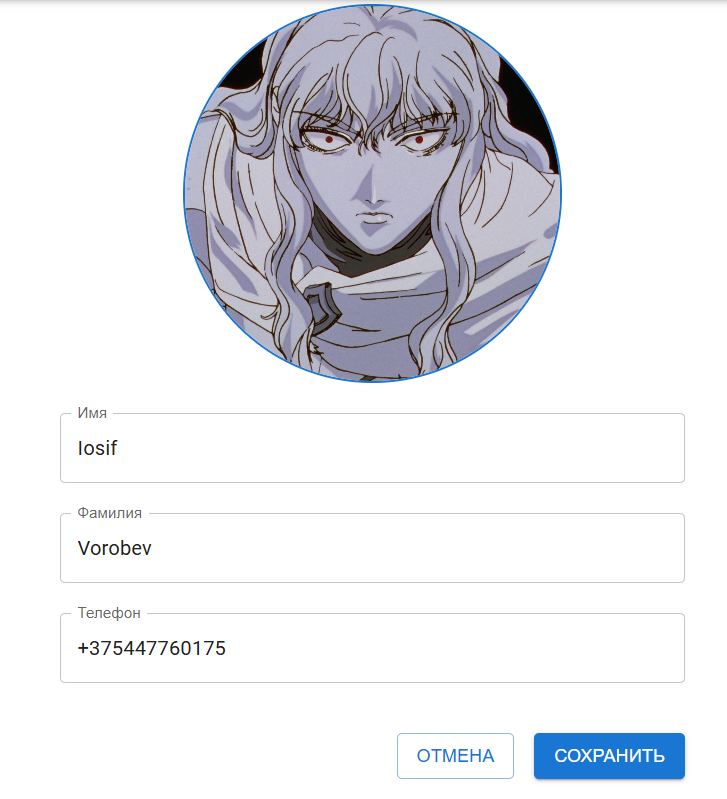


Рисунок 5.24 – Форма изменения личных данных и аватара

# Заключение

В ходе выполнения курсового проектирования было разработано веб-приложение, которое соответствует всем заявленным требованиям. Все поставленные цели были достигнуты.

В результате проектирования были:

* разработаны три роли: гость, клиент и администратор;
* реализовано 22 функции: регистрация, аутентификация, просмотр, поиск и фильтрация услуг, просмотр информации об услуге, добавление услуги в избранное, удаление услуги из избранного, добавление услуги, изменение услуги, удаление услуги, общение с представителем услуги в чате, бронирование услуги, просмотр личных бронирований, просмотр бронирований своей услуги, изменение статуса бронирования, оплата выполненной услуги, добавление отзыва на услугу, удаление отзыва на услугу, изменение информации о пользователе, просмотр списка пользователей, блокирование пользователя, разблокирование пользователя, удаление пользователя.
* было создано 7 таблиц, которые хранят необходимые для поддержания функционала данные: пользователи, refresh-токены, услуги, избранные услуги, бронирования, отзывы, сообщения.
* веб-приложение построено на клиент-серверной архитектуре. Серверная часть разработано на платформе Node.js с использованием фреймворка NestJs, клиентская – на React;
* для развертывания веб-приложения используются Docker и Docker Compose, в качестве обратного прокси выбран Nginx.
* разработаны и проведены 22 успешных функциональных теста.

На основе полученных результатов можно сделать вывод, что цель курсового проектирования была достигнута.

**Список литературы**

1. Node.js [Электронный ресурс] / Режим доступа: https://nodejs.org – Дата доступа: 25.03.2025.
2. npm [Электронный ресурс] / Режим доступа: https://www.npmjs.com – Дата доступа: 25.03.2025.
3. Onliner [Электронный ресурс] / Режим доступа: https://www.onliner.by – Дата доступа: 25.03.2025.
4. YouDo [Электронный ресурс] / Режим доступа: https://www.youdo.ru – Дата доступа: 25.03.2025.
5. Профи.ру [Электронный ресурс] / Режим доступа: https://www.profi.ru – Дата доступа: 25.03.2025
6. Secure App with Refresh-Token. The concept of refresh tokens and how… | by Max Großmann | Geek Culture | Medium [Электронный ресурс] / Режим доступа: https://medium.com/geekculture/secure-app-with-refresh-token-ddbee3716bba – Дата доступа: 27.04.2025.
7. Ubuntu 24.04.2 (Noble Numbat) [Электронный ресурс] / Режим доступа: https://releases.ubuntu.com/24.04.2/ – Дата доступа: 01.04.2025.
8. Docker Engine | Docker Docs [Электронный ресурс] / Режим доступа: https://docs.docker.com/ – Дата доступа: 10.04.2025.
9. Docker Compose | Docker Docs [Электронный ресурс] / Режим доступа: https://docs.docker.com/compose/ – Дата доступа: 10.04.2025.
10. RFC 5321 – Simple Mail Transfer Protocol (SMTP): Semantics and Content [Электронный ресурс] / Режим доступа: https://httpwg.org/specs/rfc5321.html – Дата доступа: 10.04.2025.
11. Stripe API | Stripe Docs [Электронный ресурс] / Режим доступа: https://stripe.com/docs/api – Дата доступа: 10.04.2025.
12. RFC 7231 - Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content [Электронный ресурс] / Режим доступа: https://httpwg.org/specs/rfc7231.html – Дата доступа: 10.04.2025.
13. RFC 9293: Transmission Control Proаtocol (TCP) [Электронный ресурс] / Режим доступа: https://www.ietf.org/rfc/rfc9293.html – Дата доступа: 10.04.2025.
14. The WebSocket Protocol (RFC 6455) [Электронный ресурс] / Режим доступа: https://httpwg.org/specs/rfc6455.html – Дата доступа: 10.04.2025.
15. NestJS - A progressive Node.js framework [Электронный ресурс] / Режим доступа: https://nestjs.com/ - Дата доступа: 22.04.2025.
16. Providers | NestJS - A progressive Node.js framework [Электронный ресурс] / Режим доступа: https://docs-nestjs-com-seven.vercel.app/providers – Дата доступа: 23.04.2025.
17. Data Transfer Objects (DTO) Pattern in Programming | by Amir Mustafa | Medium [Электронный ресурс] / Режим доступа: https://amirmustafaofficial.medium.com/data-transfer-objects-dto-pattern-in-programming-53f2a51a5cc9 – Дата доступа: 23.04.2025.
18. PostgreSQL: Documentation [Электронный ресурс] / Режим доступа: https://www.postgresql.org/docs/ – Дата доступа: 23.04.2025.
19. Prisma | Simplify working and interacting with databases [Электронный ресурс] / Режим доступа: https://www.prisma.io/ – Дата доступа 24.04.2025.
20. RFC 7159 - The JavaScript Object Notation (JSON) Data Interchange Format [Электронный ресурс] – Режим доступа: https://datatracker.ietf.org/doc/html/rfc7159 – Дата доступа: 25.04.2025.
21. FormData - Веб APIs | MDN [Электронный ресурс] / Режим доступа: https://developer.mozilla.org/en-US/docs/Веб/API/FormData – Дата доступа: 26.04.2025.
22. REST API standards — REST Веб Services | by Chanuka Asanka | Medium [Электронный ресурс] / Режим доступа: https://cagline.medium.com/restful-веб-services-ddafb8019f2e – Дата доступа: 27.04.2025.
23. RFC 7519 - JSON Веб Token (JWT) [Электронный ресурс] / Режим доступа: https://datatracker.ietf.org/doc/html/rfc7519 – Дата доступа: 27.04.2025.
24. Express - фреймворк веб-приложений Node.js [Электронный ресурс] / Режим доступа: https://expressjs.com/ru/ – Дата доступа: 27.04.2025.
25. OpenAPI Specification (Swagger) [Электронный ресурс] / Режим доступа: https://spec.openapis.org/oas/latest.html – Дата доступа: 10.04.2025.
26. index | Alpine Linux [Электронный ресурс] / Режим доступа: https://alpinelinux.org/ – Дата доступа: 28.04.2025.
27. What is Linux? | Linux Docs [Электронный ресурс] / Режим доступа: https://linux-docs.vercel.app/docs/what-is-linux – Дата доступа: 28.04.2025.
28. Nginx [Электронный ресурс] / Режим доступа: https://nginx.org/ru/ – Дата доступа: 02.05.2025.

# Приложение А

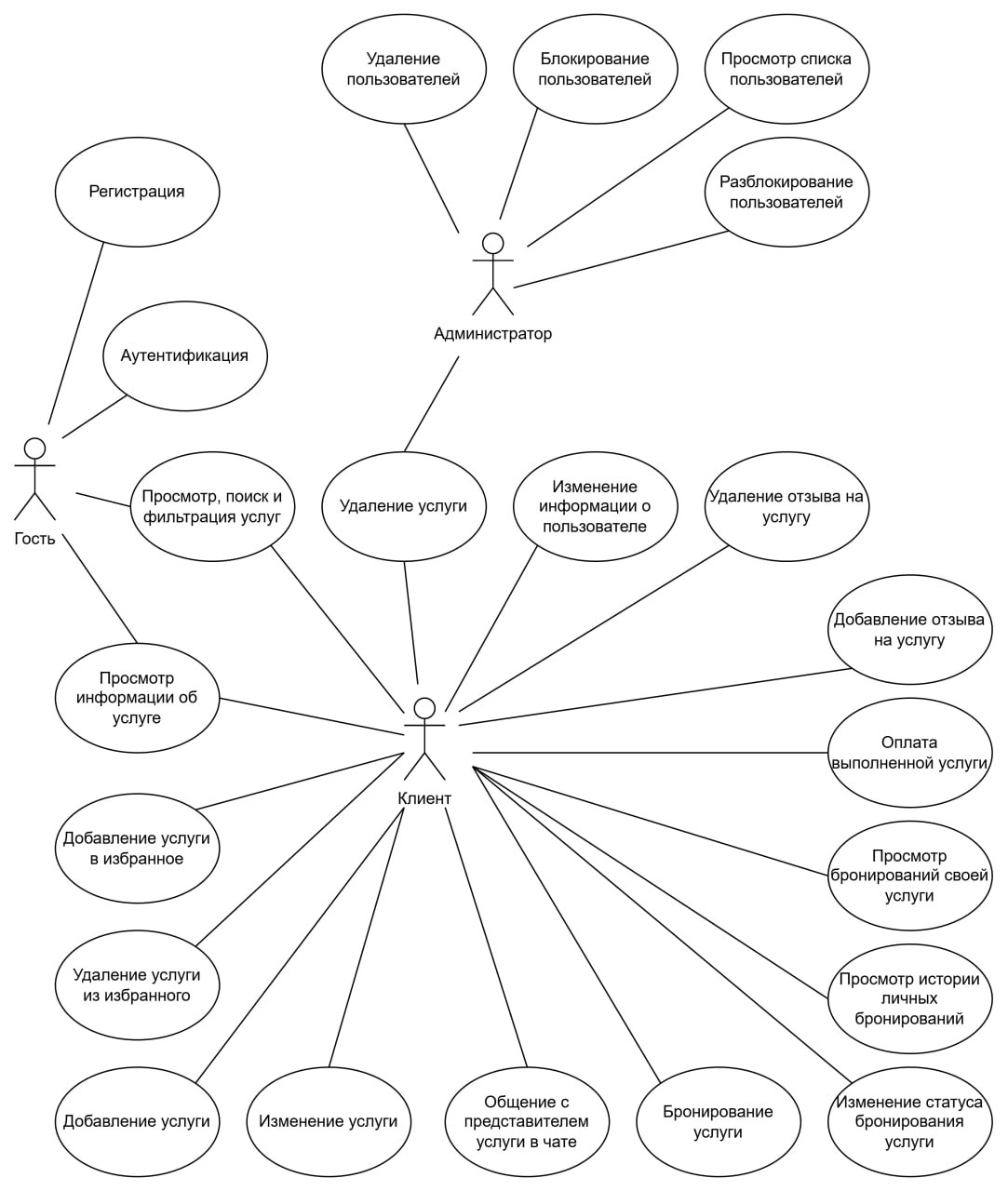


Рисунок А.1 – Диаграмма вариантов использования

# Приложение Б

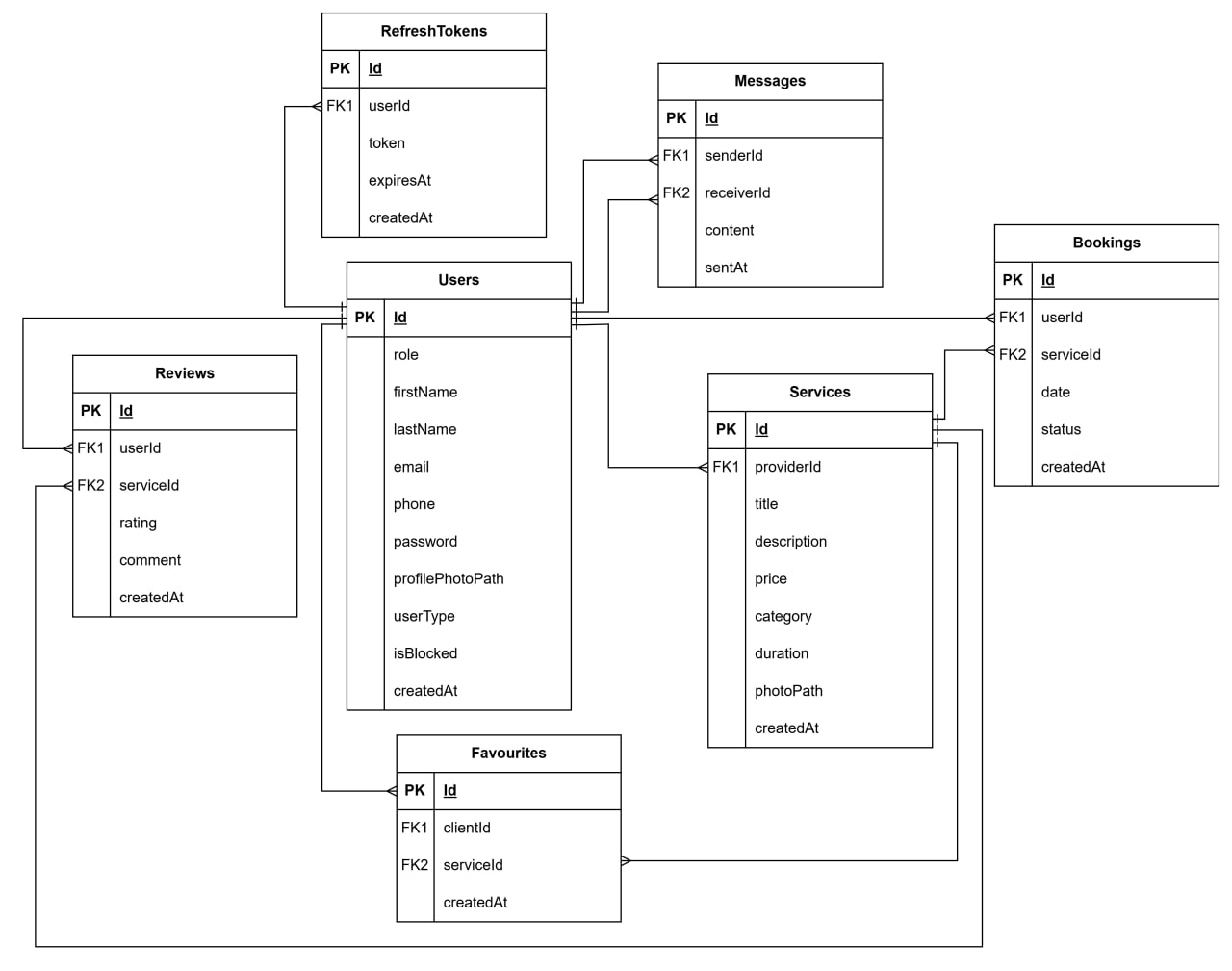


Рисунок Б.1 – Логическая схема базы данных

# Приложение В

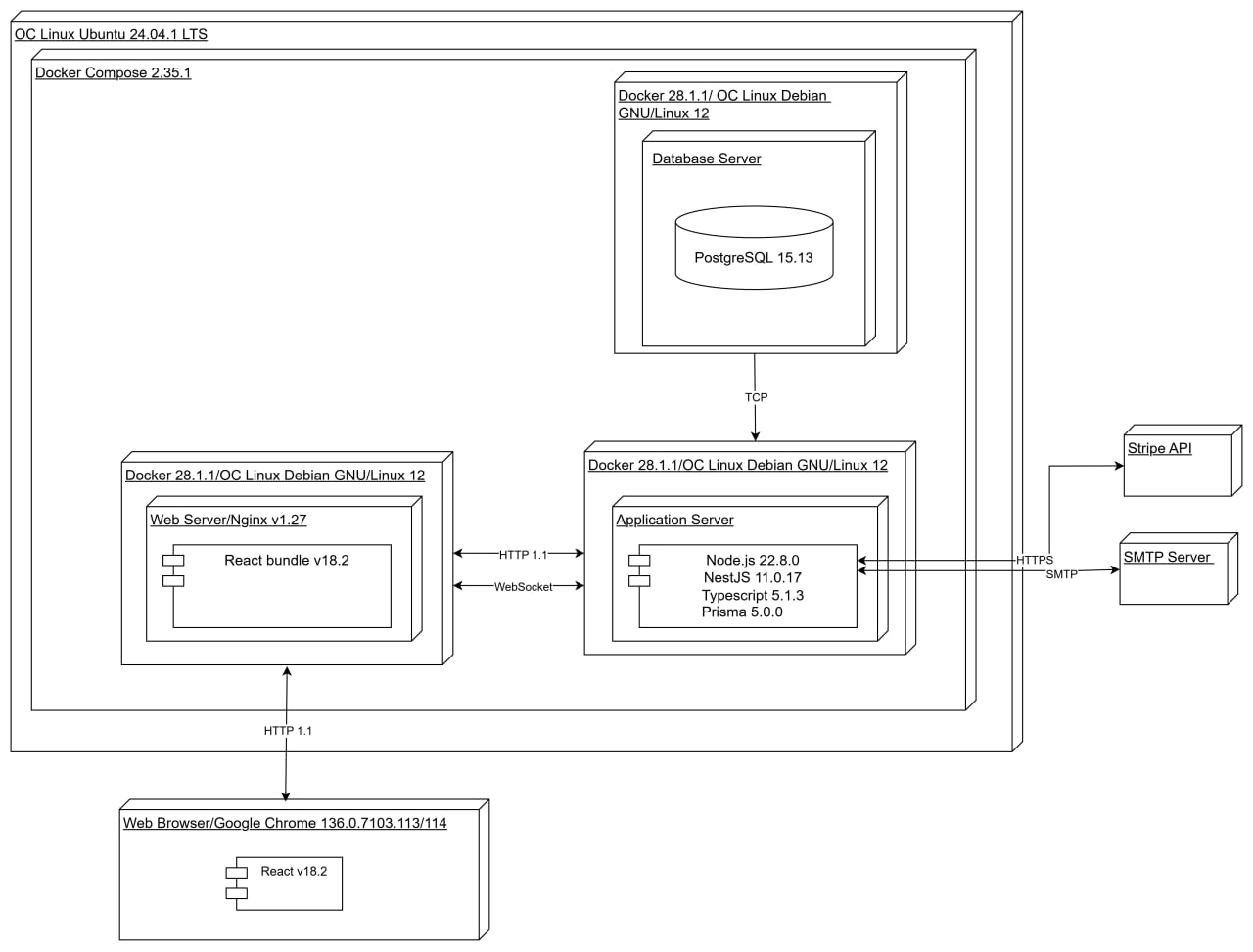


Рисунок В.1 – Диаграмма развертывания

# Приложение Г

CREATE TYPE "UserType" AS ENUM ('USER', 'ADMIN');

CREATE TYPE "BookingStatus" AS ENUM ('PENDING', 'CONFIRMED', 'CANCELLED', 'COMPLETED', 'PAID');

CREATE TABLE "Users" (

id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

firstName TEXT NOT NULL,

lastName TEXT NOT NULL,

email TEXT NOT NULL UNIQUE,

password TEXT NOT NULL,

phone TEXT NOT NULL,

profilePhotoPath TEXT,

userType "UserType" NOT NULL,

isBlocked BOOLEAN DEFAULT FALSE,

createdAt TIMESTAMPTZ DEFAULT now()

);

CREATE TABLE "Services" (

id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

title TEXT NOT NULL,

description TEXT NOT NULL,

price DECIMAL(10,2) NOT NULL,

category TEXT NOT NULL,

photoPath TEXT NOT NULL,

duration INT NOT NULL,

rating FLOAT DEFAULT 0.0,

createdAt TIMESTAMPTZ DEFAULT now(),

providerId UUID NOT NULL,

CONSTRAINT fk\_provider FOREIGN KEY (providerId) REFERENCES "User"(id) ON DELETE CASCADE

);

CREATE TABLE "Favorites" (

id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

userId UUID NOT NULL,

serviceId UUID NOT NULL,

createdAt TIMESTAMPTZ DEFAULT now(),

CONSTRAINT fk\_fav\_user FOREIGN KEY (userId) REFERENCES "User"(id) ON DELETE CASCADE,

CONSTRAINT fk\_fav\_service FOREIGN KEY (serviceId) REFERENCES "Service"(id) ON DELETE CASCADE,

CONSTRAINT unique\_favorite UNIQUE (userId, serviceId)

);

CREATE TABLE "Bookings" (

id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

date TIMESTAMPTZ NOT NULL,

status "BookingStatus" NOT NULL,

createdAt TIMESTAMPTZ DEFAULT now(),

serviceId UUID NOT NULL,

userId UUID NOT NULL,

CONSTRAINT fk\_booking\_service FOREIGN KEY (serviceId) REFERENCES "Service"(id) ON DELETE CASCADE,

CONSTRAINT fk\_booking\_user FOREIGN KEY (userId) REFERENCES "User"(id) ON DELETE CASCADE

);

CREATE TABLE "Reviews" (

id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

rating INT NOT NULL,

comment TEXT NOT NULL,

createdAt TIMESTAMPTZ DEFAULT now(),

serviceId UUID NOT NULL,

userId UUID NOT NULL,

CONSTRAINT fk\_review\_service FOREIGN KEY (serviceId) REFERENCES "Service"(id) ON DELETE CASCADE,

CONSTRAINT fk\_review\_user FOREIGN KEY (userId) REFERENCES "User"(id) ON DELETE CASCADE

);

CREATE TABLE "Messages" (

id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

content TEXT NOT NULL,

sentAt TIMESTAMPTZ DEFAULT now(),

senderId UUID NOT NULL,

receiverId UUID NOT NULL,

CONSTRAINT fk\_msg\_sender FOREIGN KEY (senderId) REFERENCES "User"(id) ON DELETE CASCADE,

CONSTRAINT fk\_msg\_receiver FOREIGN KEY (receiverId) REFERENCES "User"(id) ON DELETE CASCADE

);

CREATE TABLE "RefreshTokens" (

id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

token TEXT NOT NULL,

userId UUID NOT NULL,

createdAt TIMESTAMPTZ DEFAULT now(),

expiresAt TIMESTAMPTZ NOT NULL,

CONSTRAINT fk\_token\_user FOREIGN KEY (userId) REFERENCES "User"(id) ON DELETE CASCADE

);

Листинг Г.1 – Скрипт создания базы данных

# Приложение Д

generator client {  
 provider = "prisma-client-js"  
 enableTracing = true  
}  
  
datasource db {  
 provider = "postgresql"  
 url = env("DATABASE\_URL")  
}  
  
enum UserType {  
 USER  
 ADMIN  
}  
  
enum BookingStatus {  
 PENDING  
 CONFIRMED  
 CANCELLED  
 COMPLETED  
 PAID  
}  
  
model User {  
 id String @id @default(uuid())  
 firstName String  
 lastName String  
 email String @unique  
 password String  
 phone String  
 profilePhotoPath String?  
 userType UserType  
 isBlocked Boolean @default(false)  
 createdAt DateTime @default(now())  
 services Service[]  
 favourites Favorite[]  
 bookings Booking[]  
 reviews Review[]  
 sentMessages Message[] @relation("SentMessages")  
 receivedMessages Message[] @relation("ReceivedMessages")  
 refreshTokens RefreshToken[]  
}  
  
model Service {  
 id String @id @default(uuid())  
 title String  
 description String  
 price Decimal @db.Decimal(10, 2)  
 category String  
 photoPath String  
 duration Int  
 rating Float @default(0.0)  
 createdAt DateTime @default(now())  
 providerId String  
 provider User @relation(fields: [providerId], references: [id], onDelete: Cascade)  
 bookings Booking[]  
 favourites Favorite[]  
 reviews Review[]  
}  
  
model Favorite {  
 id String @id @default(uuid())  
 userId String  
 serviceId String  
 user User @relation(fields: [userId], references: [id], onDelete: Cascade)  
 service Service @relation(fields: [serviceId], references: [id], onDelete: Cascade)  
 createdAt DateTime @default(now())  
  
 @@unique([userId, serviceId])  
}  
  
model Booking {  
 id String @id @default(uuid())  
 date DateTime  
 status BookingStatus  
 createdAt DateTime @default(now())  
 serviceId String  
 userId String  
 service Service @relation(fields: [serviceId], references: [id], onDelete: Cascade)  
 user User @relation(fields: [userId], references: [id], onDelete: Cascade)  
}  
  
model Review {  
 id String @id @default(uuid())  
 rating Int  
 comment String  
 createdAt DateTime @default(now())  
 serviceId String  
 userId String  
 service Service @relation(fields: [serviceId], references: [id], onDelete: Cascade)  
 user User @relation(fields: [userId], references: [id], onDelete: Cascade)  
}  
  
model Message {  
 id String @id @default(uuid())  
 content String  
 sentAt DateTime @default(now())  
 senderId String  
 receiverId String  
 sender User @relation("SentMessages", fields: [senderId], references: [id], onDelete: Cascade)  
 receiver User @relation("ReceivedMessages", fields: [receiverId], references: [id], onDelete: Cascade)  
}  
  
model RefreshToken {  
 id String @id @default(uuid())  
 token String  
 userId String  
 user User @relation(fields: [userId], references: [id], onDelete: Cascade)  
 createdAt DateTime @default(now())  
 expiresAt DateTime  
}

Листинг Д.1 – Код Prisma моделей