

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ  
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Информационных систем и технологий

Специальность 1-40 01 01 «Программное обеспечение информационных технологий»

Специализация 1-40 01 01 10 «Программное обеспечение информационных технологий (программирование интернет-приложений)»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА  
К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

Web-приложение «Салон красоты»

Выполнил студент Голодок Анастасия Юрьевна  
(Ф.И.О.)

Руководитель проекта ст. преп. Дубовик М. В.  
(учен. степень, звание, должность, подпись, Ф.И.О.)

И.о.заведующего кафедрой ст.преп. Блинова Е.А.  
(учен. степень, звание, должность, подпись, Ф.И.О.)

Консультанты ст. преп. Дубовик М. В.  
(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтролёр ст. преп. Дубовик М. В.  
(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой \_\_\_\_\_

Минск 2024

## Содержание

Введение.....	3
1 Постановка задачи.....	4
1.1 Обзор аналогов.....	4
1.1.1 Культура маникюра.....	4
1.1.2 Сафина.....	5
1.2 Формирование требований.....	6
2 Проектирование web-приложения.....	7
2.1 Архитектура приложения.....	7
2.2 Диаграмма UML.....	8
2.3 Структура базы данных.....	9
3 Разработка web-приложения.....	12
3.1 Реализация серверной части.....	12
3.2 Обмен сообщениями с клиентом в реальном времени.....	15
3.3 Реализация клиентской части.....	16
4 Тестирование web-приложения.....	20
4.1 Тестирование web-приложения.....	20
5 Руководство пользователя.....	23
5.1 Гость.....	23
5.2 Пользователь.....	25
5.3 Администратор.....	27
5.4 Установка приложения.....	28
Заключение.....	29
Список используемых источников.....	30
Приложение А.....	31
Приложение Б.....	33
Приложение В.....	37

## **Введение**

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

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

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

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

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

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

# 1 Постановка задачи

## 1.1 Обзор аналогов

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

### 1.1.1 Культура маникюра

Рассмотрим в качестве примера сайт салона красоты «Культура маникюра» представленный на рисунке 1.1.

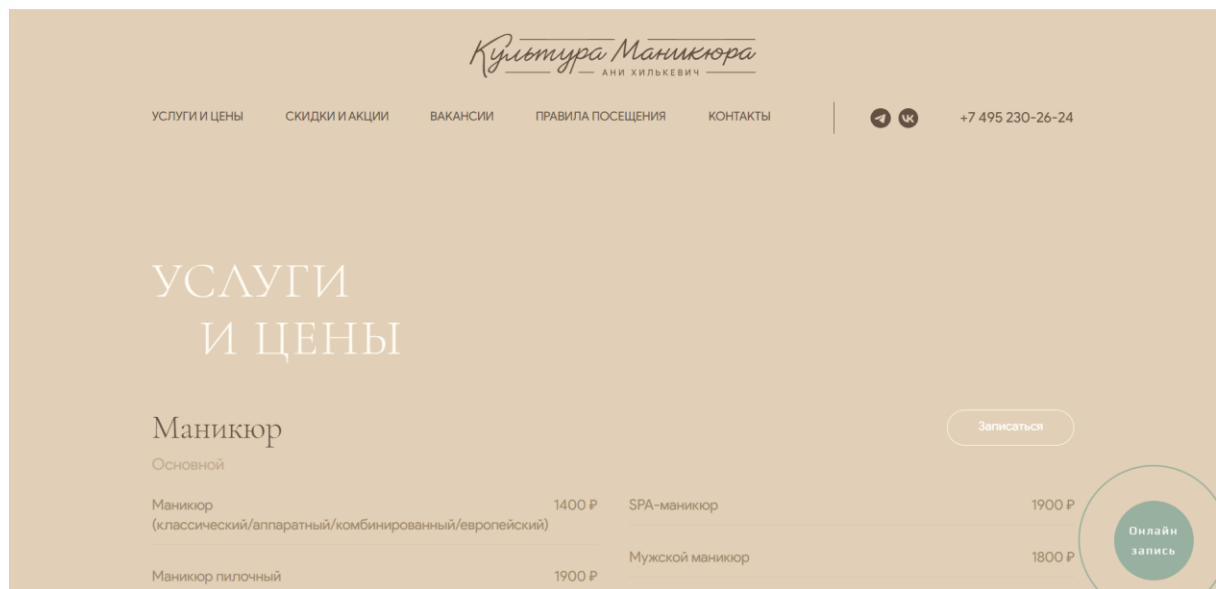


Рисунок 1.1 – Интерфейс главной страницы

Бронь услуги осуществляется в 4 этапа:

- 1) Выбор услуги;
- 2) Выбор доступного мастера и времени (рисунок 1.2);
- 3) Выбор конкретного вида услуги;
- 4) Заполнение личной информации.

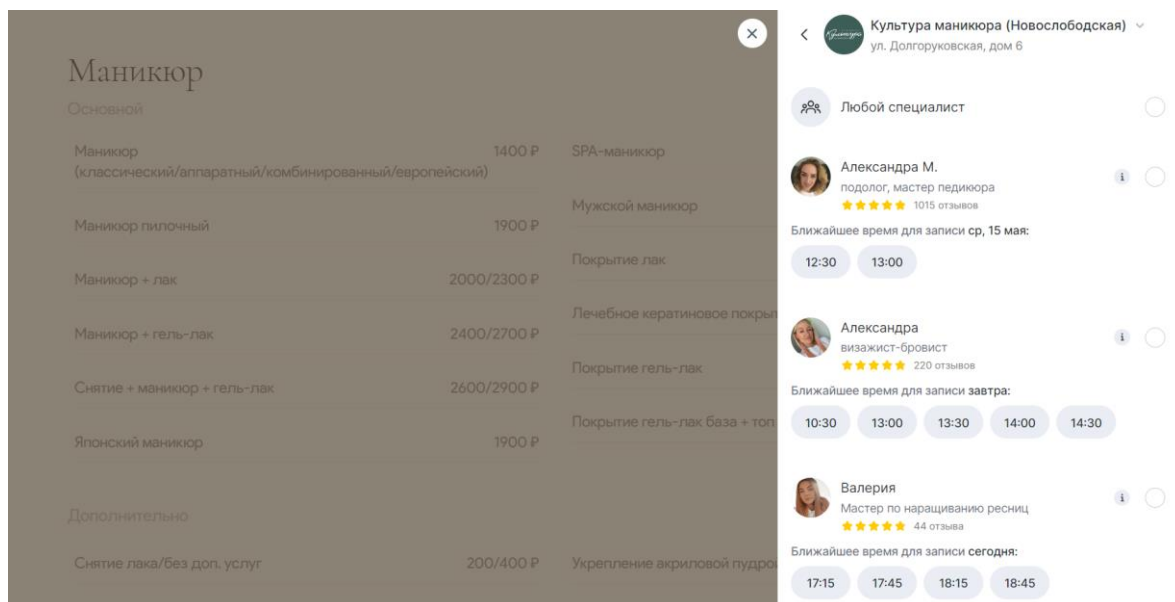


Рисунок 1.2 – Интерфейс выбора времени у мастера

Web-приложение выглядит достаточно стильно, демонстрирует высокий уровень салона, удобная последовательность и интерфейс при записи на услугу, осуществлена проверка на корректность введенных данных. Присутствует информация о каждой услуге, и цена. Из недостатков: на первый взгляд непонятно, как просмотреть или оставить отзыв. В остальном все сделано красиво и корректно.

### 1.1.2 Сафина

Далее рассмотрим сайт салона красоты «Сафина». Сайт предлагает большое количество услуг. Запись на услугу также происходит в 4 этапа (рисунок 1.3).

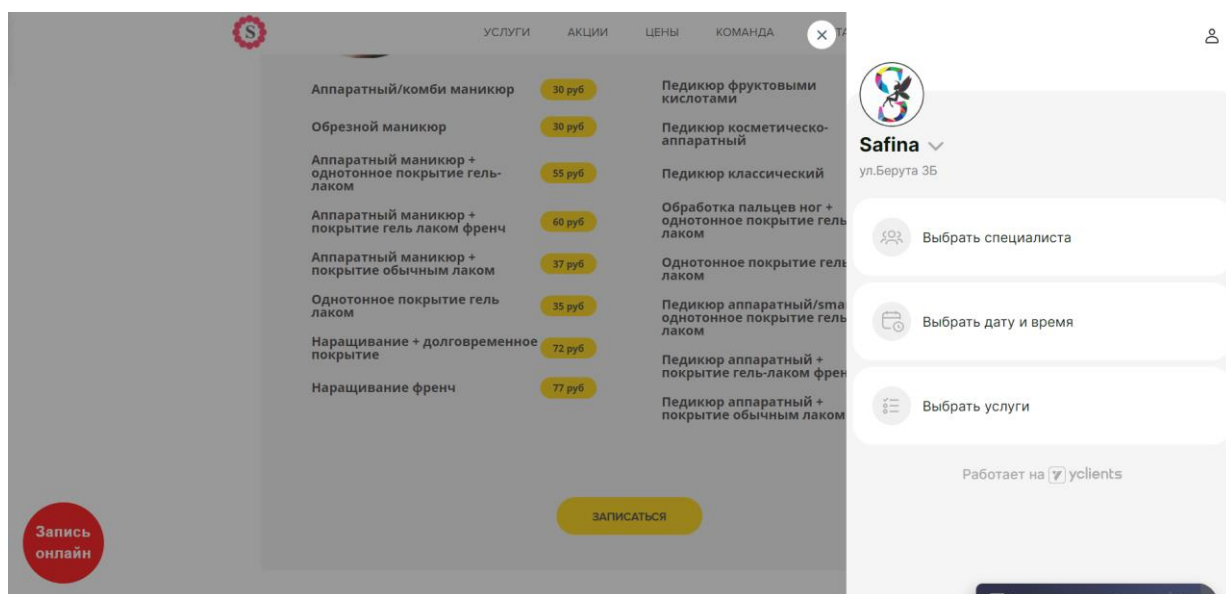


Рисунок 1.3 – Интерфейс страницы осуществления записи

Функциональность сайта не отличается от предыдущего аналога, удобная последовательность при записи на процедуру. Что касается дизайна, он более простой, выглядит не так стильно, как первый вариант, что ставит его на уровень ниже. Недостатков в работе с приложением нет.

## **1.2 Формирование требований**

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

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

Необходимо разработать несколько интерфейсов: для пользователя и для администратора.

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

Интерфейс администратора должен позволять проводить операции с содержимым сайта, описаниями. Сайт должен быть выполнен в спокойных цветах и оттенках.

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

- создать пользовательский интерфейс для взаимодействия с опциями приложения;
- регистрировать и авторизовать пользователей;
- изменять информацию о пользователях;
- просматривать услуги;
- осуществлять запись на услуги;
- осуществлять отмену записи;
- добавлять, изменять и удалять услуги;
- добавлять, изменять и удалять мастеров;
- добавлять и просматривать отзывы.

## 2 Проектирование web-приложения

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

### 2.1 Архитектура приложения

Приложение построено на основе клиент-серверной архитектуры.

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

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

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

Взаимодействие между клиентской и серверной частью приложения осуществляется следующим образом: клиент отправляет HTTPS-запросы на сервер с помощью axios. Сервер обрабатывает запросы, выполняет необходимые операции в базе данных и формирует HTTPS-ответы в формате JSON. Клиентская часть обрабатывает ответы от сервера и обновляет пользовательский интерфейс.

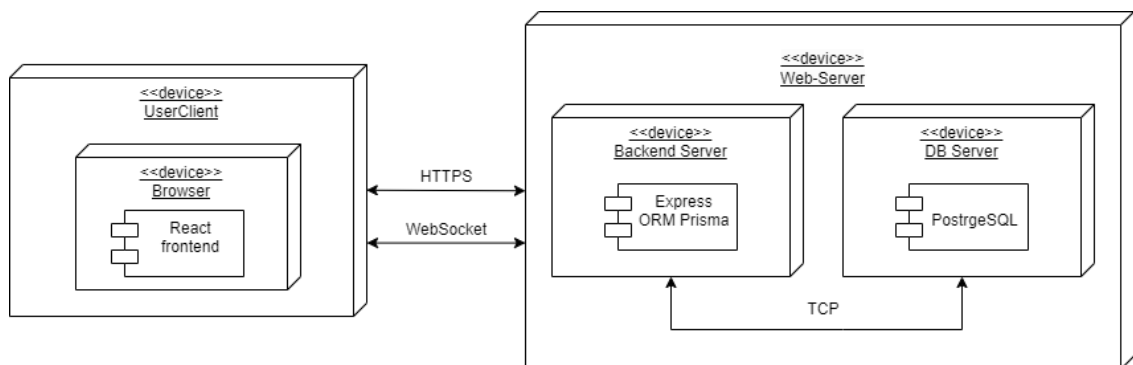


Рисунок 2.1 – Диаграмма развёртывания приложения

На диаграмме представлены все вышеописанные связи основных модулей приложения.

## 2.2 Диаграмма UML

Диаграмма UML – это графическое представление набора элементов, изображаемое в виде связанного графа с вершинами (сущностями) и ребрами (отношениями). Диаграмма использования представлена на рисунке 2.2.



Рисунок 2.2 – Диаграмма вариантов использования

На данной диаграмме представлены 3 роли:

- гость;
- клиент;
- администратор.

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



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

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

## 2.3 Структура базы данных

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

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

Выбор данного СУБД произошел по нескольким причинам: производительность и масштабируемость, расширяемость, безопасность, надёжность.

Все эти преимущества делают PostgreSQL привлекательным выбором для разработки и управления базами данных, включая курсовые проекты. Он предлагает гибкость, производительность и надежность, необходимые для успешной работы с данными и решения сложных задач.

Для базы данных «BeautySalon» было разработано 8 таблиц: Users, Employees, Registration, Services, Reviews, Schedule, Tips, EmployeesServices.

Таблица Users представляет собой данные о пользователе, состоит из столбцов:

- userID – идентификатор пользователя, тип number, первичный ключ;
- name – имя пользователя, тип nvarchar2(50);
- phone – телефон пользователя, nvarchar2(20);
- email – адрес электронной почты пользователя, тип nvarchar2(100);
- password – пароль пользователя, тип nvarchar2(50),
- role – роль (USER/ADMIN).

Таблица Employees представляет собой информацию о сотрудниках, состоит из столбцов:

- employeeID – идентификатор сотрудника, тип number, первичный ключ;
- serviceID – идентификатор услуги, тип number, внешний ключ;
- name – имя сотрудника, тип nvarchar2(50);
- positions – специализация сотрудника, тип nvarchar2(50);
- phone – телефон сотрудника, nvarchar2(20);
- email – адрес электронной почты сотрудника, тип nvarchar2(100).

Таблица Registration предназначена для хранения информации о записи на услугу, состоит из следующих столбцов:

- registrationID – идентификатор записи, тип number, первичный ключ;
- clientID – идентификатор клиента, тип number, внешний ключ;

- employeeID – идентификатор сотрудника, тип number, внешний ключ;
- dateTime – дата и время записи, тип TIMESTAMP;
- notes – комментарий, тип nvarchar2(255).

Таблица Services представляет собой информацию о сервисах, состоит из столбцов:

- serviceID – идентификатор услуги, тип number, первичный ключ;
- name – название услуги, тип nvarchar2(100), внешний ключ;
- description – описание услуги, тип nvarchar2(255);
- price – цена услуги, тип decimal(10,2).

Таблица Reviews представляет собой информацию об отзывах, состоит из столбцов:

- reviewID – идентификатор отзыва, тип number, первичный ключ;
- userID – идентификатор пользователя, тип number, внешний ключ;
- employeeID – идентификатор сотрудника, тип number, внешний ключ;
- rating – рейтинг сотрудника, тип number;
- comm – отзыв, тип nvarchar2(255).

Таблица Schedule представляет собой расписание сотрудников, состоит из столбцов:

- scheduleID – идентификатор расписания, тип number, первичный ключ;
- employeeID – идентификатор сотрудника, тип number, внешний ключ;
- date – день, тип DateTime;
- startTime – начальное время, тип DateTime;
- endTime – конечное время, тип DateTime.

Таблица Tips представляет собой хранилище для советов, которые будет периодически получать пользователь, состоит из:

- id – идентификатор совета, тип number, первичный ключ;
- tip – совет, тип nvarchar2(255), внешний ключ.

Таблица EmployeesServices представляет собой связывание сотрудников и услуг для обеспечения связи многие ко многим.

Взаимосвязь всех таблиц проектируемой базы данных представлена на рисунке 2.3.

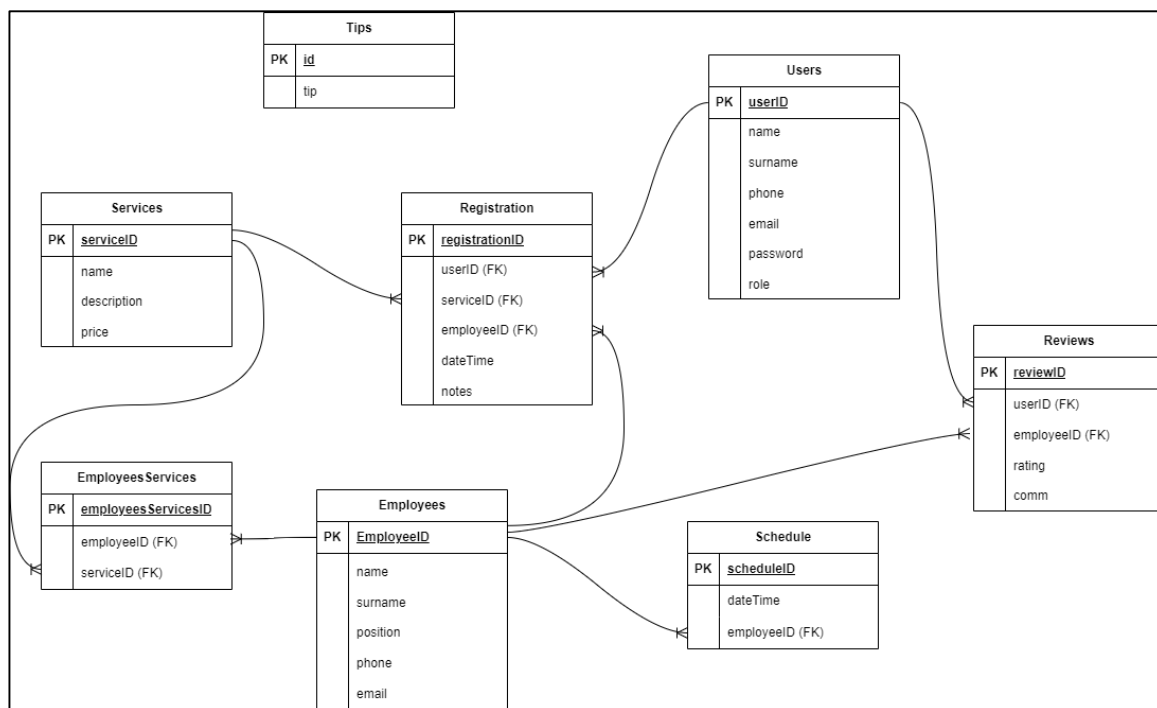


Рисунок 2.3 – Взаимосвязь таблиц базы данных

Листинг создания базы данных, используя ORM Prisma, представлен в приложении А.

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

## 3 Разработка web-приложения

### 3.1 Реализация серверной части

При написании серверной части на Node.js используется фреймворк Express.

Проект разделен на несколько основных частей, которые расположены по разным директориям. На рисунке 3.1 представлена структура проекта.

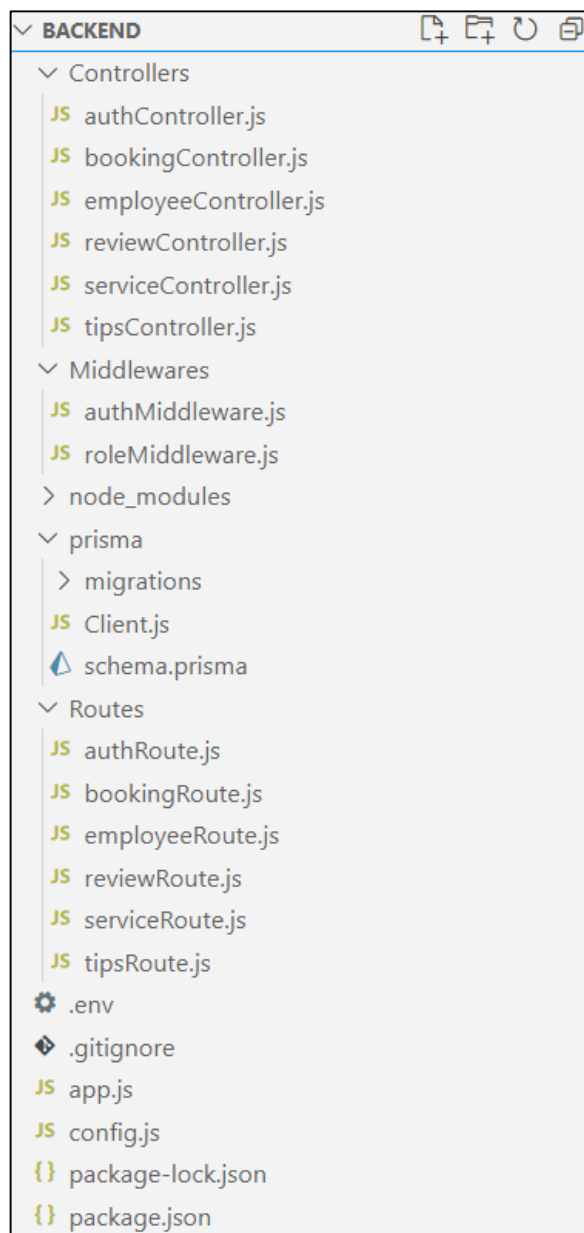


Рисунок 3.1 – Структура директорий проекта сервера

В директории «Controllers» расположены контроллеры, обеспечивающие взаимодействие пользователя и системы, обрабатывая запросы.

Директория «Middlewares» содержит промежуточные обработчики, которые нужны для проверки JWT токена и доступности действия исходя из роли пользователя. В файле `roleMiddleware.js` функция проверяет наличие и корректность токена в заголовке запроса, извлекает информацию о пользователе и определяет соответствие роли пользователя требуемой роли для доступа к ресурсу. Код проверки роли представлен на листинге 3.1.

```
const jwt = require("jsonwebtoken");
const secret = process.env.ACCESS_TOKEN_SECRET;
module.exports = function (role) {
  return function (req, res, next) {
    if (req.method === "OPTIONS") {
      next();
    }
    try {
      const token = req.headers.authorization.split(" ")[1];
      console.log(token);
      if (!token) {
        console.log("Пользователь не авторизован");
        return res.status(403).json({ message: "Пользователь не авторизован" });
      }
      const decoded = jwt.verify(token, secret);
      if (decoded.role !== role) {
        console.log("Нет доступа к выполнению операции");
        return res
          .status(403)
          .json({ message: "Нет доступа к выполнению операции" });
      }
      req.user = decoded;
      next();
    } catch (error) {
      res.status(401).json({ message: "Не авторизован" });
      console.log("Не авторизован");
    }
  };
};
```

Листинг 3.1 – Код для проверки роли пользователя

Директория «prisma» в файле `schema.prisma` содержит модели, необходимые для генерации таблиц в базе данных. Конфигурация подключения импортируется из файла `.env`. Для того, чтобы изменить базу нужно сделать миграцию.

Главным файлом является `app.js`, в котором происходит загрузка модулей и запуск приложения. В нем используются различные модули, такие как `cors`, `express.json`, позволяющие использовать ресурсы находящихся на другом домене, обрабатывать данные в `json` формате. Также маршруты для различных запросов, такие как регистрация, аутентификация, управление услугами, отзывами, сотрудниками, бронированием и советами.

Директория «Routes» содержит роутеры, которые реализуют маршрутизацию. В листинге 3.2 представлено добавление роутеров с учётом их расположения в директориях проекта. Приложение настроено на использование роутеров по определённому пути.

```
const authRoute = require("./Routes/authRoute");
const servRoute = require("./Routes/serviceRoute");
const revRoute = require("./Routes/reviewRoute");
const emplRoute = require("./Routes/employeeRoute");
const bookingRoute = require("./Routes/bookingRoute");
const tipsRoute = require("./Routes/tipsRoute");

app.use("/auth", authRoute);
app.use("/serv", servRoute);
app.use("/rev", revRoute);
app.use("/empl", emplRoute);
app.use("/book", bookingRoute);
app.use("/tips", tipsRoute);
```

Листинг 3.2 – Регистрация роутеров

Для отправки сообщений напоминания пользователям о записи используется nodemailer. Данные извлекаются из базы данных, и из них формируется текст письма. Код представлен в листинге 3.3.

```
const nodemailer = require("nodemailer");
const transporter = nodemailer.createTransport({
  host: process.env.EMAIL_HOST,
  port: process.env.EMAIL_PORT,
  secure: process.env.EMAIL_SECURE,
  service: process.env.EMAIL_SERVICE,
  auth: {
    user: process.env.EMAIL_USER,
    pass: process.env.EMAIL_PASS, }, },
  { from: process.env.EMAIL_FROM, } );
const sendMail = async (email, data) => {
  const text = `Уважаемый ${data.name}, ${data.day},
${data.time}, ждём вас на процедуре ${data.service}. Ваш мастер
${data.employee}`;
  const message = {
    to: email,
    subject: process.env.EMAIL_SUBJECT,
    text: text,
  };
  await transporter.sendMail(message, (err, info) => {
    if (err) return console.log(err);
    console.log("Email send: ", info);
  });
};
```

Листинг 3.3 – Отправка сообщений на почту

Для взаимодействия с базой данных PostgreSQL используется схема Prisma. Она включает модели, атрибуты и ограничения.

Секретные данные для работы приложения, такие как строка подключения к базе данных, секрет для JWT-токена, пароли для отправки напоминаний на почту, а также названия сертификатов и ключей для HTTPS протокола содержатся в .env файле.

### 3.2 Обмен сообщениями с клиентом в реальном времени

Для поддержки двусторонней связи в приложении использован веб-сокет.

WebSocket — протокол связи поверх TCP-соединения, предназначенный для обмена сообщениями между браузером и веб-сервером в режиме реального времени.

Для установки соединения WebSocket клиент и сервер используют протокол, похожий на HTTP. Клиент формирует особый HTTP-запрос, на который сервер отвечает определенным образом.

Отображение в браузере представлено на рисунке 3.2

Request URL:	ws://localhost:3000/ws
Request Method:	GET
Status Code:	● 101 Switching Protocols
▼ Response Headers	<input type="checkbox"/> Raw
Connection:	Upgrade
Sec-WebSocket-Accept:	D/+7aA+7BlzUjFaJah1DNqGVthE=
Upgrade:	websocket

Рисунок 3.2 – Отображение в браузере установки соединения WebSocket

Веб-сокеты в данном приложении реализованы с помощью библиотеки Socket.io и используются для получения клиентами советов по уходу. Код создания сокета представлен в листинге 3.3.

```
const server = http.createServer(app);
const io = socketIo(server, {
  cors: { origin: "*", methods: ["GET", "POST"] },
});
io.on("connection", (socket) => {
  console.log("Новое соединение установлено");
  socket.on("message", async (message) => {
    console.log("Получено сообщение от клиента:", message);
    const tip = await getRandomTip.getRandomTip();
    io.emit("message", tip);
  });
  socket.on("disconnect", () => {
    console.log("Соединение с клиентом разорвано");
  });
});
```

Листинг 3.3 – Фрагмент файла app.js, создающий websocket-сервер

После установки соединения по данному протоколу начинается обмен между сервером и пользователем. Клиент запрашивает совет, а сервер отправляет ему совет по уходу за собой.

### 3.3 Реализация клиентской части

Основную часть приложения составляет клиентская часть, потому что она обеспечивает взаимодействие пользователя с продуктом.

Для клиентской части приложения была использована библиотека React и фреймворк MUI, который позволяет использовать готовые шаблоны и компоненты. С помощью React можно легко создавать интерактивные пользовательские интерфейсы.

Структура директорий проекта представляет собой директорию «components», в ней находятся компоненты, которые можно подключать в любом месте страницы. Далее директория «pages» в ней расположены сами страницы, которые будут открываться по маршрутам, указанным в App.js, часть кода представлена в листинге 3.4.

```
<Route path="/main" element={<MainPage />} />
<Route path="/register" element={<RegistrationForm />} />
<Route path="/login" element={<LoginForm />} />
<Route path="/services" element={<ServicesPage />} />
<Route path="/serv/updserv/:id" element={<EditService />} />
```

Листинг 3.3 – Фрагмент, содержащий route

Структура директорий представлена на рисунке 3.3

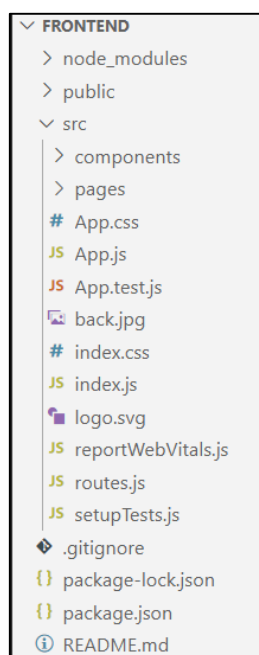


Рисунок 3.3 – Структура директорий проекта клиента



Для проверки роли и в последующем дачи доступа или закрытия используется `jwt-token`, который хранится в `localStorage` и при необходимости извлекается оттуда. С помощью токена можно проверить аутентифицирован ли пользователь, а также проверить его. В листинге 3.4 приведен код файла `AppRouter.js`, функция которого вызывается в файлах страниц для проверки доступа.

```
import React, { useEffect } from "react";
import { useNavigate } from "react-router-dom";

const AppRouter = ({ userRole }) => {
  const navigate = useNavigate();

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

  return null;
};

export default AppRouter;
```

Листинг 3.4 – Содержимое файла `AppRouter.js`

Если у пользователя нет доступа к определенному ресурсу, либо в `url` был введен некорректный путь, то пользователя перенаправляет на страницу 404, на которой находится ссылка для перехода на главную страницу. Код файла `404.js` представлен в листинге 3.5.

```
import React from "react";
import { Link } from "react-router-dom";

const NotFoundPage = () => {
  return (
    <div>
      <h1>404 - Страница не найдена</h1>
      <p>Кажется, вы попали на несуществующую страницу.</p>
      <Link to="/main">Вернуться на главную</Link>
    </div>
  );
};

export default NotFoundPage;
```

Листинг 3.5 – Содержимое файла `404.js`

Компонент `AddService.js` представляет собой страницу,

предназначенную для добавления новой услуги в систему. Часть кода представлена в листинге 3.6.

```
useEffect(() => {
  const token = localStorage.getItem("authToken");
  if (token) {
    const decodedToken = jwtDecode(token);
    setUserRole(decodedToken.role);
  }
}, []);
const handleSubmit = async (serviceData, mode) => {
  try {
    const response = await axios.post("/serv/addservice",
serviceData, {
      headers: { Authorization: `Bearer
${localStorage.getItem("authToken")}` },
    });
    console.log("Услуга успешно добавлена:",
response.data.newService);
    navigate("/services");
    setError("");
  } catch (error) {
    console.error("Ошибка при добавлении услуги:", error);
    setError(error.response.data.message || "Ошибка при
добавлении услуги.");
  }
}
```

Листинг 3.5 – Содержимое файла AddService.js

Вначале извлекается роль пользователя из токена аутентификации, который хранится в локальном хранилище браузера. Это позволяет определить роль пользователя и соответствующим образом настроить интерфейс. Далее данные о новой услуге отправляются на сервер с использованием метода POST. В случае успешного ответа сервера происходит перенаправление на страницу услуг.

Для того, чтобы пользователь мог авторизоваться, ему нужно заполнить свои email и пароль. Код формы входа представлен в листинге 3.6.

```
function SignIn() {
  try {
    const response = await axios.post("/auth/login",
userData);
    const token = response.data.token;
    console.log("User logged in successfully:", token);
    localStorage.setItem("authToken", token);
    navigate("/main");
  } catch (error) {
    if (error.response && error.response.data) {
      setError(
        error.response.data.message || "Неизвестная ошибка
при входе."
      );
      console.error("Error logging in user:", error);
    }
  }
}
```

```

return (
  <Container component="main" maxWidth="xs">
    <NavMenu />
    <CssBaseline />
    <div className={classes.paper}>
      <Typography component="h1" variant="h5">
        Войти
      </Typography>
      <form className={classes.form} noValidate
onSubmit={handleSubmit}>
        <TextField
          id="email"
          label="Email"
          name="email"
          autoComplete="email"
          value={email}
          onChange={(e) => setEmail(e.target.value)} />
//такой же элемент для пароля
        <Button
          type="submit"
          className={classes.submit}>
          Войти
        </Button>
      </form>
    </div>
  </Container>
); }
export default SignIn;

```

### Листинг 3.5 – Часть содержимого файла SignIn.js

В целом все компоненты и страницы схожи, запросы выполняются с помощью `axios`, результат возвращается с сервера, и если это ошибка, то выводится соответствующее сообщение. Разбиение на компоненты позволило многократно использовать различные формы и удобно передавать, и выводить данные.

## 4 Тестирование web-приложения

В этой главе будут рассмотрены основные элементы интерфейса и протестирован интерфейс веб-приложения.

Приложение во многих местах позволяет пользователю вводить данные. Разработанное приложение устойчиво к вводу неверной информации, и сообщает пользователю об ошибках.

### 4.1 Тестирование web-приложения

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

Регистрация

Имя \*  
Anastasiya

Номер телефона \*  
1

Email \*  
e@mail

Пароль \*  
....

Неверно введены данные

ЗАРЕГИСТРИРОВАТЬСЯ

[Уже есть аккаунт?](#)

Рисунок 4.1 – Форма регистрации при неправильных данных

При вводе правильного логина и пароля пользователь перенаправляется на главную страницу приложения, в случае неверного ввода данных, появляются сообщения внизу поля, сигнализируя о неправильном вводе email или пароля. Интерфейс формы входа можно увидеть на рисунке 4.2.

Войти

Email \*  
nastya@mail.ru

Пароль \*  
.....

Неверный логин или пароль

ВОЙТИ

[Нет аккаунта? Зарегистрироваться](#)

Рисунок 4.2 – Форма входа при неправильных данных

Существует также проверка на существование пользователя с введённым

адресом электронной почты. При попытке войти в приложение через пользователя с уже зарегистрированным email, также появляется ошибка.

Если данные указаны неверно, то пользователь не сможет сделать запрос для регистрации. Только после успешной регистрации пользователь будет перенаправлен на страницу логина.

Далее будет протестирована запись на услугу, если пользователь не авторизован. В случае, когда пользователь вошел в свой аккаунт, нажимая кнопку «Записаться», он попадает на страницу записи на услугу, представленную на рисунке 4.3.

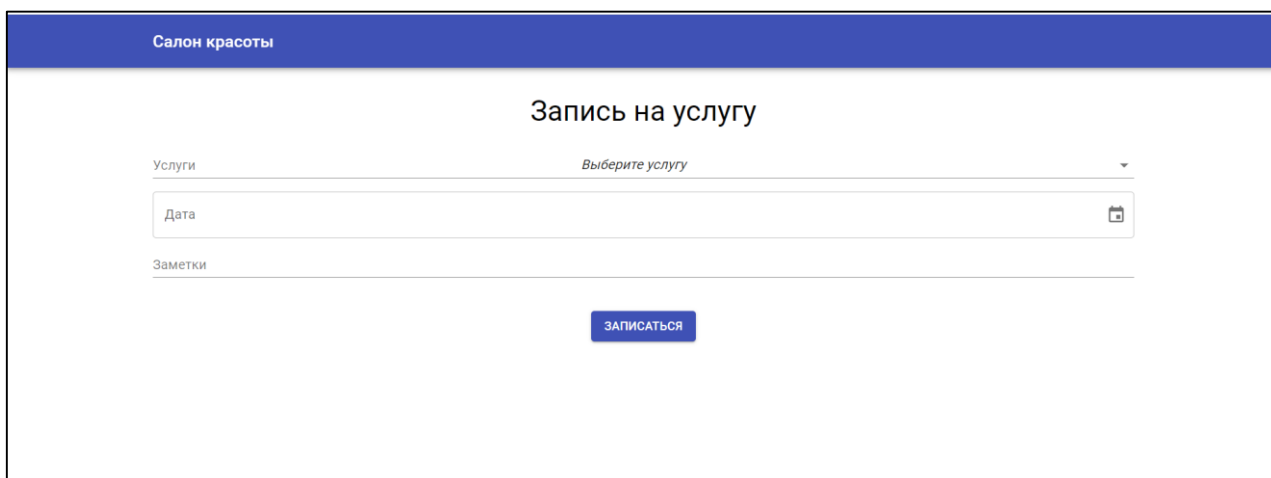


Рисунок 4.3 – Форма для записи на услугу

Но если пользователь не авторизован, то его перенаправляет на страницу входа.

Следующим этапом осуществлена проверка доступа к ресурсам, доступным только администратору. Если у пользователя нет прав администратора, то при вводе адреса в url отображается страница 404, пример приведен на рисунке 4.4.

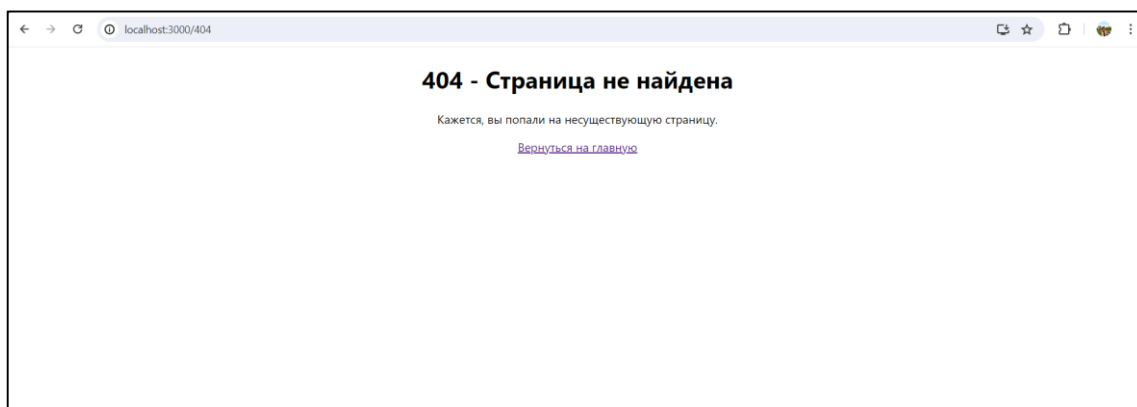


Рисунок 4.3 – Страница 404

Также если в url введен адрес, которого нет в списке, либо же доступ к нему у пользователя отсутствует, то происходит перенаправление на страницу 404.

Еще одним этапом является проверка возможных ошибок на стороне администратора. На рисунке 4.5 представлен результат добавления услуги, которая уже существует.

Рисунок 4.5 – Форма для записи на услугу

При добавлении сотрудника осуществляется проверка на корректность введенных данных, выбора услуг, а также правильность дат и времени. Если что-то неверно заполнено, выводится сообщение с ошибкой и форма сбрасывается.

Примером еще одной проверки является проверка на корректность даты, при добавлении расписания сотрудника. Результат введения неправильных даты и времени представлен на рисунке 4.6.

Рисунок 4.6 – Форма для редактирования сотрудника

На рисунке видно, что время начала рабочего дня больше, чем время окончания, чего быть не может. Точно такой же результат выведется в результате ввода даты, которая раньше текущей.

## 5 Руководство пользователя

В данном разделе находится ознакомление с остальным функционалом, доступным пользователям.

### 5.1 Гость

При первом переходе к приложению пользователь попадает на главную страницу, вид которой приведен на рисунке 5.1. На ней можно посмотреть список предоставляемых услуг и цены на них, список сотрудников их рейтинг и отзывы, чтобы выбрать подходящего.

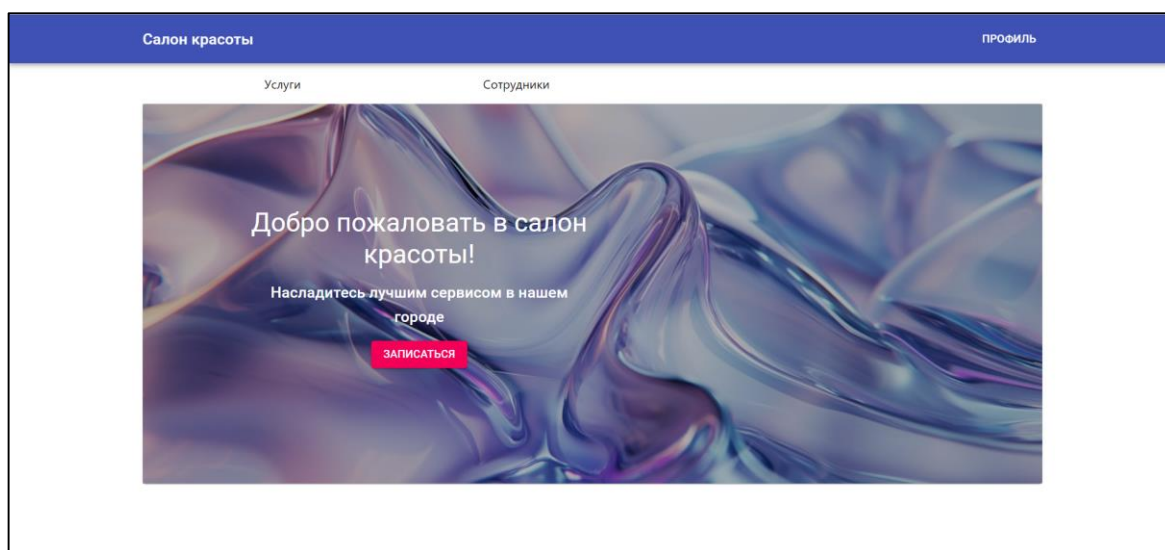


Рисунок 5.1 – Главная страница

Для просмотра рейтинга и отзывов сотрудников нужно перейти на страницу «Сотрудники» и кликнуть по интересующему, пример на рисунке 5.2

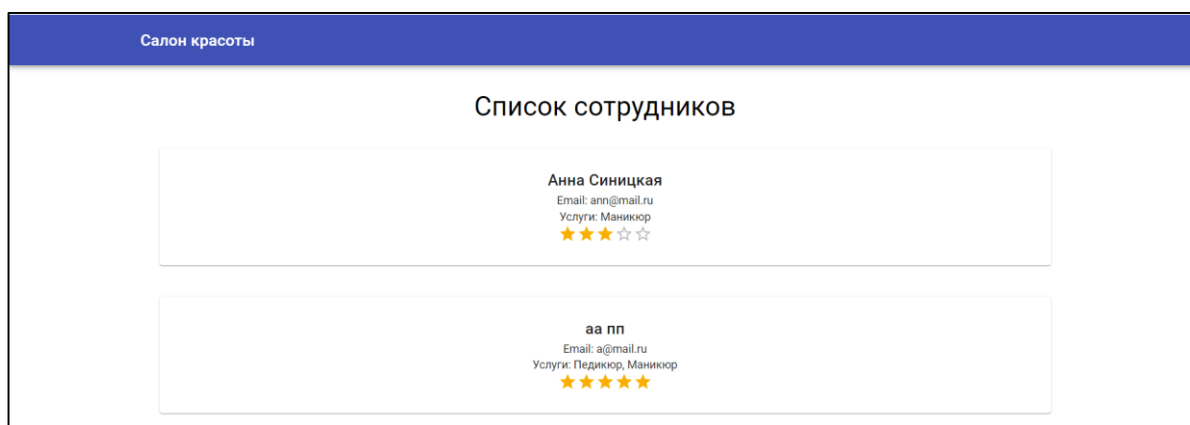


Рисунок 5.2 – Список сотрудников

Для доступа к записи на услугу, отправки отзыва и остальным функциям необходимо авторизоваться, для этого можно нажать кнопку «Войти». Также

при нажатии кнопки «Записаться» неавторизованного пользователя перенаправляет на страницу входа. Страница авторизации приведена на рисунке 5.3.

The form is titled "Войти" (Login) and is contained within a light gray box. It features two input fields: "Email \*" and "Пароль \*" (Password \*). Below these fields is a blue button labeled "войти" (login). At the bottom of the form, there is a link that reads "Нет аккаунта? Зарегистрироваться" (No account? Register).

Рисунок 5.3 – Форма авторизации

В случае, если аккаунта ещё нет, нужно перейти по ссылке и заполнить форму. После регистрации пользователь снова попадает на страницу входа.

Если гость или пользователь хочет получить совет по уходу, то ему достаточно нажать кнопку «Советы», появится модальное окно с советом, который может поддержать результат процедуры как можно дольше. Пример такого совета представлен на рисунке 5.4.

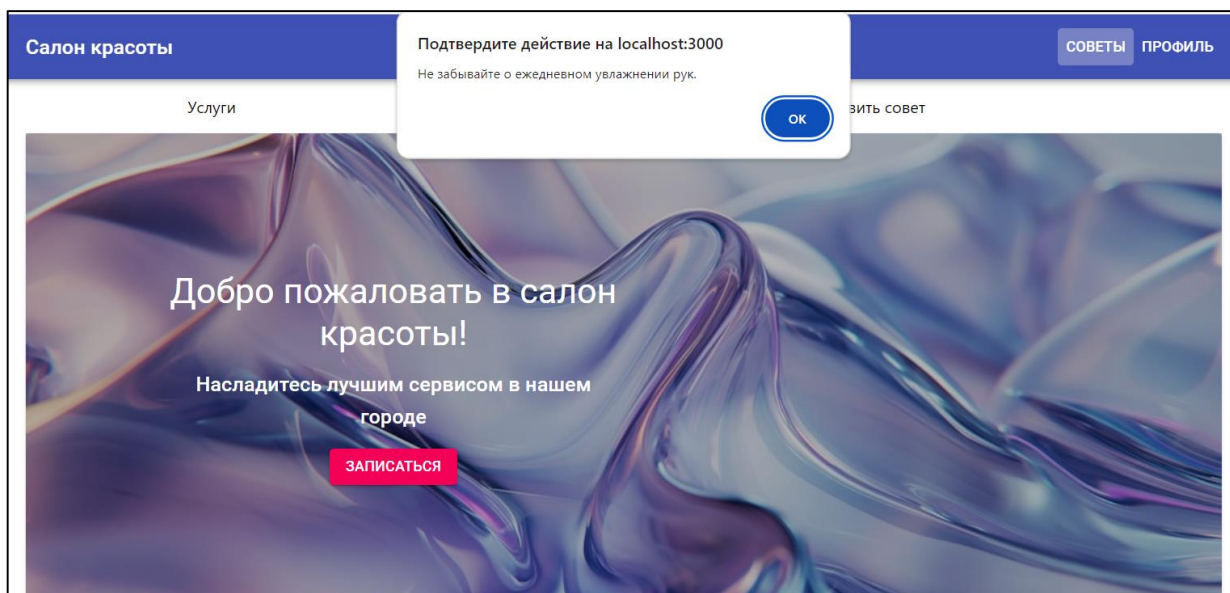


Рисунок 5.3 – Форма авторизации

Такой вариант предоставления советов выглядит очень интересным.



## 5.2 Пользователь

После успешной авторизации пользователю доступны такие опции как запись на услугу, добавление отзывов и просмотр личной информации. Для того, чтобы осуществить запись на услугу нужно нажать на кнопку «Записаться» на главной странице. Перейдя на страницу записи необходимо выбрать услугу, день и время у доступных мастеров, а также, по желанию, можно оставить заметки, пример представлен на рисунке 5.4.

The screenshot shows a web form titled 'Запись на услугу' (Service Booking) under the header 'Салон красоты' (Beauty Salon). The form is for booking a 'Маникюр' (Manicure) service. It includes a date selection field set to '05/16/2024'. Below the date, there are two sections for selecting a master: 'Анна Сеницкая' with time slots 10:00, 11:00, 12:00, 13:00, and 14:00; and 'Инна Голубева' with time slots 09:30, 10:30, 11:30, 12:30, 13:30, 14:30, 15:30, and 16:30. A 'Заметки' (Notes) field contains the text 'яркий маникюр' (bright manicure). At the bottom of the form is a blue button labeled 'ЗАПИСАТЬСЯ' (BOOK).

Рисунок 5.4 – Форма записи на услугу

Когда пользователь нажал кнопку «Записаться», при успешной записи происходит перенаправление на страницу пользователя, где отображается информация о всех его записях, и кнопка «Отменить», которая отменяет запись. Пример страницы пользователя приведен на рисунке 5.5.

The screenshot shows a user profile page titled 'Информация о пользователе' (User Information) under the header 'Салон красоты' (Beauty Salon). The page displays user details: Name 'Анастасия', Phone '+375445959897', and Email 'nastya@mail.ru'. Below the details are two buttons: a blue 'СОХРАНИТЬ' (SAVE) button and a pink 'ВЫЙТИ' (LOG OUT) button. At the bottom, there is a section titled 'Ваши записи' (Your bookings) showing a booking for 'Педикюр' (Pedicure) by master 'Анна Сеницкая' on '2024-05-16' at '11:00'. A pink 'ОТМЕНИТЬ' (CANCEL) button is located next to the booking details.

Рисунок 5.5 – Форма записи на услугу

На странице пользователя можно изменить информацию о себе, после этого необходимо нажать кнопку «Сохранить», при обнаружении ошибок, появится соответствующее сообщение. Для выхода из системы достаточно нажать на кнопку «Выйти».

Для того чтобы оставить отзыв, необходимо перейти на страницу «Сотрудники», выбрать нужного сотрудника поставить оценку, написать комментарий и нажать кнопку «Отправить», интерфейс данной операции представлен на рисунке 5.6.

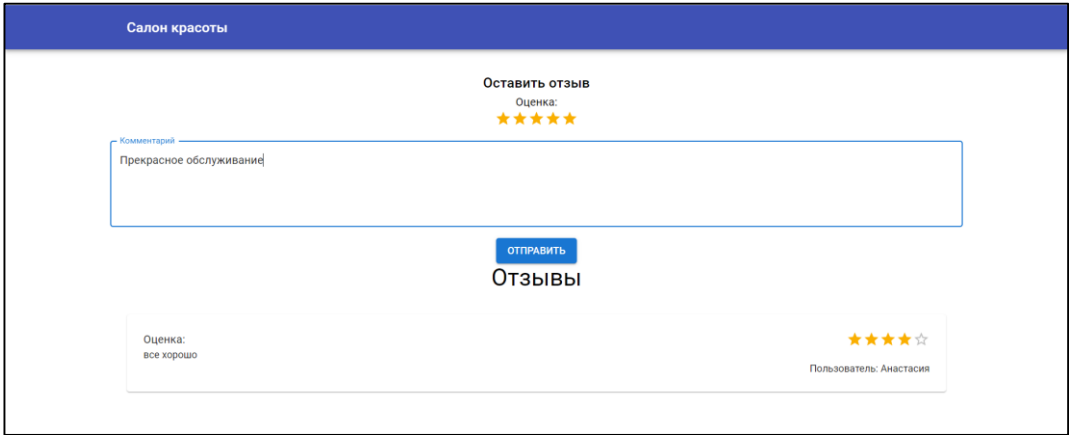
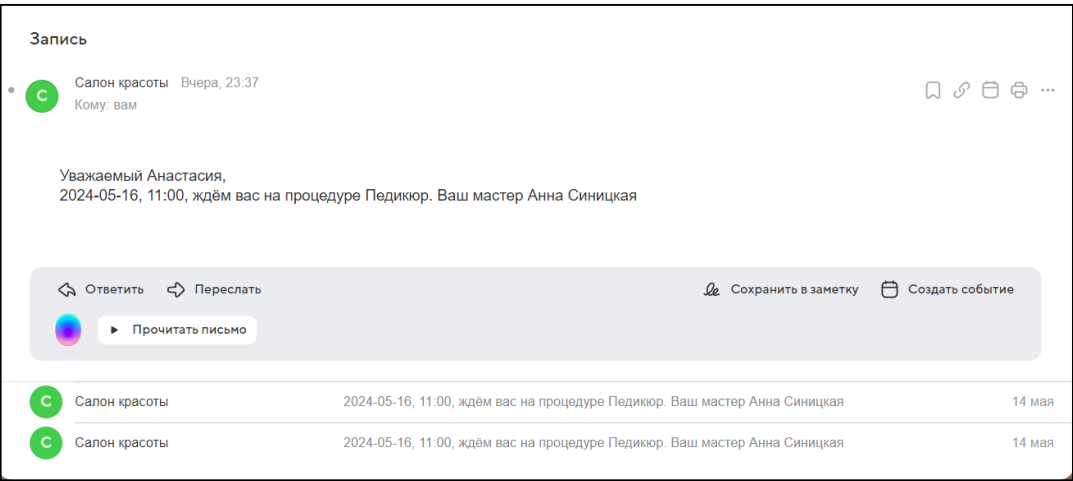


Рисунок 5.6 – Страница отзывов

В последствии у сотрудника отображается средний рейтинг, что позволяет сразу оценить примерную работу, не переходя к отзывам.

При записи на услугу, данные клиента заносятся в базу данных, впоследствии эти данные используются для того, чтобы напомнить пользователю о том, что он записан в салон красоты. Такого рода письма приходят клиенту на почту (рисунок 5.7).



От кого	Тема	Дата
Салон красоты	2024-05-16, 11:00, ждём вас на процедуре Педикюр. Ваш мастер Анна Сеницкая	14 мая
Салон красоты	2024-05-16, 11:00, ждём вас на процедуре Педикюр. Ваш мастер Анна Сеницкая	14 мая

Рисунок 5.7 – Напоминание о записи

В письме указывается дата, время и услуга, на которую записан пользователь. Все это делает посещение салона еще более привлекательным.

### 5.3 Администратор

После входа в систему главная страница администратора такая же, как и у обычного пользователя, дополнительные возможности отображаются на страницах сотрудников и услуг. К примеру, перейдя на страницу услуг, администратор может добавить новую услугу, нажав кнопку «Добавить услугу», либо изменить существующую, нажав непосредственно на нее. Пример изменения услуги услуги представлен на рисунке 5.7.

The screenshot shows a web interface for editing a service. At the top is a blue header with the text 'Салон красоты'. Below the header, the page title 'Редактирование услуги' is centered. The form contains the following fields: 'Название услуги' (Service Name) with the value 'Маникюр', 'Описание услуги' (Service Description) with the text 'Уход за руками и ногтями, который включает в себя не только нанесение лака, но и гигиеническую обработку, уход за кожей, придание формы ногтям и их украшение.', and 'Цена услуги' (Service Price) with the value '50'. At the bottom of the form are two buttons: a blue button labeled 'ИЗМЕНИТЬ УСЛУГУ' (Edit Service) and a red button labeled 'УДАЛИТЬ УСЛУГУ' (Delete Service).

Рисунок 5.7 – Страница редактирования услуги

Для удаления услуги нужно нажать кнопку «Удалить услугу». Такой же функционал и при работе с сотрудниками.

На рисунке 5.8 приведена страница добавления сотрудника.

The screenshot shows a web interface for adding a new employee. At the top is a blue header with the text 'Салон красоты' on the left and 'СОВЕТЫ' on the right. Below the header, the page title 'Добавление сотрудника' is centered. The form contains the following fields: 'Имя \*' (Name) with the value 'Екатерина', 'Фамилия \*' (Surname) with the value 'Булай', 'Должность' (Position) with the value 'бровист', and 'Email \*' (Email) with the value 'kat@mail.ru'. Below these fields is a dropdown menu for 'Услуги' (Services) with the selected value 'Коррекция и окрашивание бровей'. At the bottom of the form are two buttons: a blue button labeled 'ДОБАВИТЬ РАСПИСАНИЕ' (Add Schedule) and a blue button labeled 'ДОБАВИТЬ СОТРУДНИКА' (Add Employee).

Рисунок 5.7 – Страница добавление сотрудника

Также администратор может добавлять советы, которые будут отображаться пользователю. Алгоритм добавления советов такой же, как и при добавлении услуг.

## 5.4 Установка приложения

Для запуска приложения необходимо выполнить следующие шаги:

1. Для запуска серверной части, которая будет соединять базу данных и сторону клиента, обрабатывать запросы и выполнять различные операции, необходимо выполнить команду «node app» в директории «backend».

2. Запустить клиентскую часть приложения, которая будет взаимодействовать с сервером и предоставлять клиенту пользовательский интерфейс можно с помощью команды «npm start». Клиентская часть приложения находится в директории «frontend».

Необходимые https сертификат и приватный ключ находятся в главной директории проекта.

После выполнения всех действий приложение будет готово к работе. Сайт станет доступен в браузере по url <https://localhost:3000>.

## Заключение

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

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

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

Использование WebSocket позволяет моментально отправлять клиентам советы по уходу за собой, поддерживая постоянное соединение между клиентом и сервером. Для обеспечения безопасности и конфиденциальности данных применяется протокол HTTPS. Это повышает доверие пользователей к приложению и обеспечивает защиту данных.

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

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

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

## **Список используемых источников**

1 Node.js [Электронный ресурс] – Режим доступа: <https://nodejs.org/en/about/> – Дата доступа: 10.03.2024.

2 Документация на фреймворк Express [Электронный ресурс] / Режим доступа: <https://expressjs.com> – Дата доступа: 12.03.2024.

3 PostgreSQL Сайт о программировании [Электронный ресурс] / Режим доступа: <https://postgrespro.ru/docs/postgresql.com> – Дата доступа: 20.03.2024.

4 React [Электронный ресурс] – Режим доступа: <https://react.dev/> – Дата доступа: 23.03.2023.

5 WebSocket API [Электронный ресурс] – Режим доступа: <https://learn.javascript.ru/websocket> – Дата доступа: 22.04.2023.

## Приложение А

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

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

model Users {
  userID      Int      @id @default(autoincrement())
  name        String
  phone       String?
  email       String   @unique
  password    String
  role        Roles    @default(USER)

  reviews    Reviews[]
  registrations Registration[]
}

enum Roles {
  ADMIN
  USER
}

model Services {
  serviceID   Int      @id @default(autoincrement())
  name        String   @unique
  description  String?
  price       Decimal  @default(0.00)

  employees   EmployeesServices[]
}

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

  services    EmployeesServices[]
  reviews    Reviews[]
  registrations Registration[]
  schedules   Schedule[]
}

model EmployeesServices {
  employeeID  Int
  serviceID   Int
```

```

    employee Employees @relation(fields: [employeeID], references:
[employeeID], onDelete: Cascade)
    service Services @relation(fields: [serviceID], references:
[serviceID], onDelete: Cascade)

    @@id([employeeID, serviceID])
}

model Registration {
  registrationID Int @id @default(autoincrement())
  userID Int
  employeeID Int
  dateTime DateTime
  notes String?

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

model Reviews {
  reviewID Int @id @default(autoincrement())
  userID Int
  employeeID Int
  rating Int
  comm String?

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

model Schedule {
  scheduleID Int @id @default(autoincrement())
  employeeID Int
  date DateTime
  startTime DateTime
  endTime DateTime

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

model Tips {
  id Int @id @default(autoincrement())
  tip String?
}

```

Листинг 1 – Модель базы данных



## Приложение Б

```
import React, { useState, useEffect } from "react";
import {
  TextField,
  Button,
  Typography,
  MenuItem,
  Select,
  FormControl,
  InputLabel,
} from "@material-ui/core";
import { makeStyles } from "@material-ui/core/styles";
import DateTimePicker from "../DateTimePicker";
import axios from "axios";

const useStyles = makeStyles((theme) => ({
  formControl: {
    marginBottom: theme.spacing(2),
  },
  button: {
    marginTop: theme.spacing(3),
  },
}));

function EmployeeForm({ initialData = {}, mode = "add", onSubmit }) {
  const classes = useStyles();
  // Состояния для данных формы
  const [name, setName] = useState(initialData.name || "");
  const [surname, setSurname] = useState(initialData.surname || "");
  const [positions, setPositions] =
    useState(initialData.positions || "");
  const [email, setEmail] = useState(initialData.email || "");
  const [services, setServices] = useState(initialData.services || []);
  const [schedules, setSchedules] = useState(
    initialData.mode === "add" ? [{ date: "", startTime: "", endTime: "" }] : []
  );
  const [availableServices, setAvailableServices] =
    useState([]);
  const [error, setError] = useState("");

  // Загружаем список доступных услуг из сервера
  useEffect(() => {
    const fetchServices = async () => {
      try {
        const response = await axios.get("/serv/getservices");
        setAvailableServices(response.data);
      } catch (error) {
```

```

        console.error("Ошибка при загрузке списка услуг:",
error);
        setError("Ошибка при загрузке списка услуг.");
    }
};
fetchServices();
}, []);

useEffect(() => {
    if (mode === "edit") {
        setServices(initialData.services.map((service) =>
service.id));
    }
}, [mode, initialData.services]);

// Обработчик отправки формы
const handleSubmit = async (e) => {
    e.preventDefault();

    // Валидация данных формы
    if (!name || !surname || !email) {
        setError("Заполните все обязательные поля.");
        return;
    }

    // Формирование данных для отправки
    const employeeData = {
        name,
        surname,
        positions,
        email,
        services,
        schedules,
    };
    try {
        // Передаем данные формы в обработчик onSubmit
        await onSubmit(employeeData, mode);
        setError("");
    } catch (error) {
        console.error("Ошибка при обработке формы:", error);
        setError(error.response?.data?.message || "Ошибка при
обработке формы");
    }
};

const handleServiceChange = (e) => {
    setServices(e.target.value);
};

const handleAddSchedule = () => {
    setSchedules([...schedules, { date: "", startTime: "",
endTime: "" }]);
};

```

```

const handleScheduleChange = (index, field, value) => {
  const updatedSchedules = [...schedules];
  updatedSchedules[index][field] = value;
  setSchedules(updatedSchedules);
};

return (
  <form onSubmit={handleSubmit}>
    <TextField
      fullWidth
      label="Имя"
      value={name}
      onChange={(e) => setName(e.target.value)}
      className={classes.formControl}
      required
    />
    <TextField
      fullWidth
      label="Фамилия"
      value={surname}
      onChange={(e) => setSurname(e.target.value)}
      className={classes.formControl}
      required
    />
    <TextField
      fullWidth
      label="Должность"
      value={positions}
      onChange={(e) => setPositions(e.target.value)}
      className={classes.formControl}
    />
    <TextField
      fullWidth
      label="Email"
      type="email"
      value={email}
      onChange={(e) => setEmail(e.target.value)}
      className={classes.formControl}
      required
    />
    <FormControl fullWidth className={classes.formControl}>
      <InputLabel id="service-label">Услуги</InputLabel>
      <Select
        labelId="service-label"
        multiple
        value={services}
        onChange={handleServiceChange}
        renderValue={(selected) =>
          selected
            .map((serv) => {
              const service = availableServices.find((s) => {
                return s.id == serv;
              });
            })
        }
      />
    </FormControl>
  </form>
);

```

```

        });
        return service ? service.name : "";
    })
    .join(" ")
  } >
  {availableServices.map((service) => (
    <MenuItem key={service.id} value={service.id}>
      {service.name}
    </MenuItem>
  ))}
</Select>
</FormControl>

{/* Поля для выбора расписаний */}
{schedules.map((schedule, index) => (
  <div key={index}>
    <DateTimePicker
      index={index}
      schedule={schedule}
      onUpdateSchedule={handleScheduleChange}
    />
  </div>
))}
<Button
  variant="contained"
  color="primary"
  onClick={handleAddSchedule}
  className={classes.button}
>
  Добавить расписание
</Button>
{error && <Typography color="error">{error}</Typography>}
<Button
  variant="contained"
  color="primary"
  type="submit"
  className={classes.button}
>
  {mode === "add" ? "Добавить сотрудника" : "Изменить
сотрудника"}
</Button>
</form>
);
}
export default EmployeeForm;

```

## Приложение В

```
import React, { useState } from "react";
import {
  LocalizationProvider,
  DatePicker,
  TimeField,
} from "@mui/x-date-pickers";
import { AdapterDayjs } from "@mui/x-date-pickers/AdapterDayjs";
import { Grid, TextField } from "@mui/material";

function DateTimePicker({ index, onUpdateSchedule }) {
  const [selectedDate, setSelectedDate] = useState(null);
  const [selectedStartTime, setSelectedStartTime] =
    useState(null);
  const [selectedEndTime, setSelectedEndTime] = useState(null);

  const handleDateChange = (newDate) => {
    setSelectedDate(newDate);
    onUpdateSchedule(index, "date", newDate);
  };

  const handleStartTimeChange = (newTime) => {
    setSelectedStartTime(newTime);
    onUpdateSchedule(index, "startTime", newTime);
  };

  const handleEndTimeChange = (newTime) => {
    setSelectedEndTime(newTime);
    onUpdateSchedule(index, "endTime", newTime);
  };

  return (
    <LocalizationProvider dateAdapter={AdapterDayjs}>
      <Grid container spacing={2}>
        {/* Выбор даты */}
        <Grid item xs={4}>
          <DatePicker
            label="Выберите день"
            value={selectedDate}
            onChange={handleDateChange}
            renderInput={(props) => <TextField {...props} />}
          />
        </Grid>

        {/* Выбор времени начала */}
        <Grid item xs={4}>
          <TimeField
            label="Время начала"
            value={selectedStartTime}
            format="HH:mm"
            onChange={handleStartTimeChange}
          />
        </Grid>
      </Grid>
    </LocalizationProvider>
  );
}
```

```
        />
      </Grid>

      { /* Выбор времени окончания */ }
      <Grid item xs={4}>
        <TimeField
          label="Время окончания"
          value={selectedEndTime}
          format="HH:mm"
          onChange={handleEndTimeChange}
        />
      </Grid>
    </Grid>
  </LocalizationProvider>
);
}
export default DateTimePicker;
```