МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования

«Гродненский государственный университет имени Янки Купалы»

Факультет математики и информатики

Кафедра современных технологий программирования

ЗАНЕВСКИЙ ЕВГЕНИЙ КАЗИМИРОВИЧ

**Проектирование системы администрирования салона красоты**

Курсовая работа   
по дисциплине «Системы баз данных»

студента 3 курса специальности

1-26 03 01 «Управление информационными ресурсами»

дневной формы получения образования

|  |  |
| --- | --- |
|  | Научный руководитель  Гуща Юлия Вальдемаровна,  старший преподаватель кафедры современных технологий программирования |

Гродно 2023

**РЕЗЮМЕ**

Тема курсовой работы

**Проектирование системы администрирования салона красоты**

Работа содержит: 50 страниц, 9 рисунков, 13 листингов, 2 приложения, 7 использованных источников.

Ключевые слова: салон красоты, база данных, концептуальная модель, логическая модель, физическая модель, Power Designer, MySQL Workbench

Целью курсовой работы является проектирование системы администрирования для салона красоты.

Объектом исследования для данной работы является салон красоты.

Предмет исследования - автоматизация учета обслуживания клиентов салона красоты.

Программа написана на языке высокого уровня C# и предназначена для развития интеллектуальных способностей людей различных возрастных категорий. Так же целью работы является закрепление теоретических знаний, полученных при изучении курса «Системы баз данных», путём проектирования базы данных, представляющего собой автоматизированную систему контроля и управления салоном красоты.

Основные результаты работы над курсовой работой:

* Разработана эффективная база данных для салона красоты, которая содержит информацию о клиентах, мастерах, услугах и записях на обслуживание.
* В ходе выполнения курсовой работы были изучены и применены различные методы и инструменты проектирования баз данных, такие как ER-моделирование, нормализация и SQL-запросы.
* В результате выполнения курсовой работы были получены необходимые знания и навыки в области проектирования баз данных.

**RESUME**

The topic of the course work

**Designing a beauty salon administration system**

The work contains: 50 pages, 9 figures, 13 listings, 2 appendixes, 7 sources used.

Keywords: beauty salon, database, conceptual model, logical model, physical model, Power Designer, MySQL Workbench

The purpose of the course work is to design an administration system for a beauty salon.

The object of research for this work is a beauty salon.

The subject of the study is automation of accounting for beauty salon customer service.

The program is written in a high-level C# language and is designed to develop the intellectual abilities of people of various age categories. Also, the purpose of the work is to consolidate the theoretical knowledge gained during the study of the course "Database Systems" by designing a database, which is an automated control and management system for a beauty salon.

The main results of the work on the course work:

* An effective database for a beauty salon has been developed, which contains information about customers, masters, services and service records.
* During the course work, various methods and tools of database design were studied and applied, such as ER modeling, normalization and SQL queries.
* As a result of the course work, the necessary knowledge and skills in the field of database design were obtained.

СОДЕРЖАНИЕ

[**ВВЕДЕНИЕ** 5](#_Toc136389119)

[**1** **РЕАЛИЗАЦИЯ БАЗЫ ДАННЫХ** 7](#_Toc136389120)

[**1.1 Физическая модель базы данных** 7](#_Toc136389121)

[**2** **ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ** 9](#_Toc136389122)

[**2.1 Разработка структуры приложения** 10](#_Toc136389123)

[**2.2 Программная реализация серверной части приложения** 14](#_Toc136389124)

[**2.3 Реализация пользовательской части приложения** 21](#_Toc136389125)

[**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ** 29](#_Toc136389126)

[**ПРИЛОЖЕНИЯ** 30](#_Toc136389127)

# **ВВЕДЕНИЕ**

Индустрия красоты получила широкое распространение во всем мире и стала одной из самых быстрорастущих отраслей, привлекая множество потребителей и предпринимателей.

В настоящее время благодаря стремительному развитию ИТ-технологий в нашей жизни появилась возможность записаться на приём к врачу, оставить заявку на оказание некоторой услуги и многое другое, не выходя из дома. Наличие возможности онлайн-записи приводит к значительной экономии времени со стороны клиента. В таком случае клиент в любое время может просмотреть информацию о перечне услуг, их стоимости, времени работы и оформить заявку. По этой причине, со стороны салона красоты, существует потребность в онлайн-записи, которая может позволить привлечь большее количество клиентов.

На сегодняшний день имеется множество платных систем онлайн-записи. Однако не все они выполняют следующую функциональность:

* вся информация о всех заказах хранится в личном кабинете клиента;
* у клиента в личном кабинете должна быть возможность оформить заявку на одну или несколько услуг салона с возможностью выбора мастера;
* в качестве авторизованных пользователей предполагаются только клиенты и администратор салона;
* мастера могут посмотреть свое расписание с записями клиентов
* администратору предоставлена возможность по добавлению, удалению и редактированию информации о клиентах, мастерах, услугах, их стоимости, может просмотреть все записи на услуги по каждому клиенту и мастеру.

Для облегчения и удобства работы необходима система администрирования салона красоты, которая отвечала бы всем вышеперечисленным требованиям.

Целью курсовой работы является проектирование системы администрирования для салона красоты.

Объектом исследования для данной работы является салон красоты.

Предмет исследования - автоматизация учета обслуживания клиентов салона красоты.

Для достижения указанной цели необходимо решить такие задачи как:

1. Анализ предметной области и определение требований к базе данных.

2. Проектирование структуры базы данных, включая определение таблиц, полей и связей между таблицами.

3. Определение типов данных для каждого поля.

4. Нормализация базы данных до нужного уровня.

5. Создание ER-диаграммы, отображающей структуру базы данных.

6. Разработка SQL-запросов для создания таблиц и связей между ними.

7. Создание SQL-запросов для вставки, обновления и удаления данных из таблиц.

8. Создание форм и отчетов для ввода и вывода данных из базы данных.

9. Тестирование базы данных на наличие ошибок и соответствие требованиям.

База данных «Салон красоты» повышает эффективность контроля, учета и взаимодействия с клиентами, что упрощает задачи персонала и сокращает время, затрачиваемое на их выполнение.

# **1** **РЕАЛИЗАЦИЯ БАЗЫ ДАННЫХ**

Проектирование и реализация базы данных являются важными этапами разработки программного приложения. База данных представляет собой организованную коллекцию данных, которая используется для хранения, управления и извлечения информации.

## **1.1 Физическая модель базы данных**

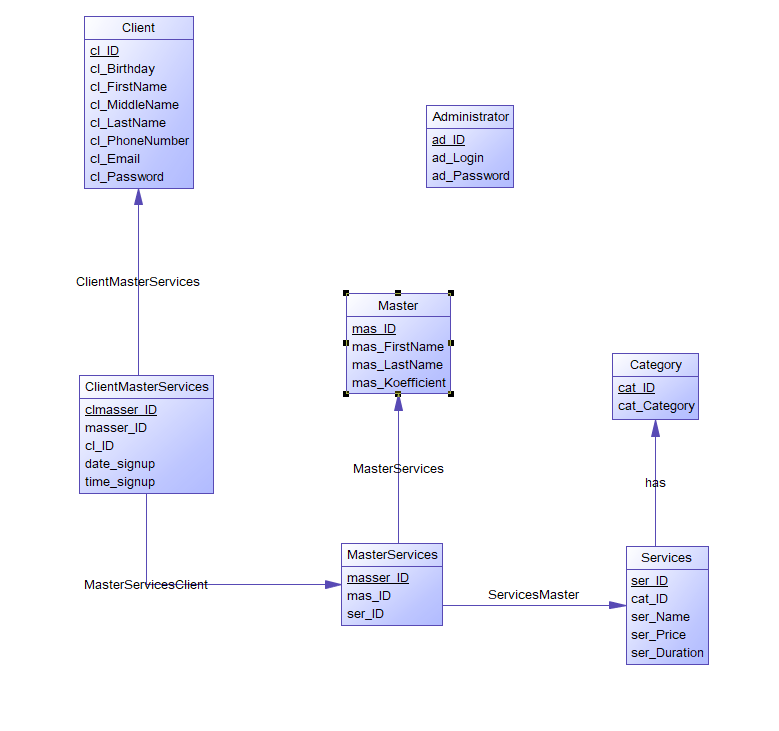
Физическая модель базы данных - это конкретная реализация логической модели базы данных, которая определяет способ хранения данных и методы доступа к ним на уровне конкретной СУБД (системы управления базами данных). Физическая модель базы данных учитывает особенности выбранной СУБД, аппаратное и программное обеспечение, а также требования к производительности и доступности данных.

Основная цель физической модели базы данных - определить, как данные будут физически организованы на диске и каким образом они будут доступны приложениям (модель представлена на рисунке 2.3).

Каждое поле в таблице необходимо для хранения определенной информации, которая используется для работы с базой данных и обеспечения ее функциональности.

На следующем этапе необходимо сгенерировать SQL-код на основе физической модели, созданной в Power Designer, для дальнейшего переноса в MySQL Workbench.

Созданный SQL-код описан в приложении А.

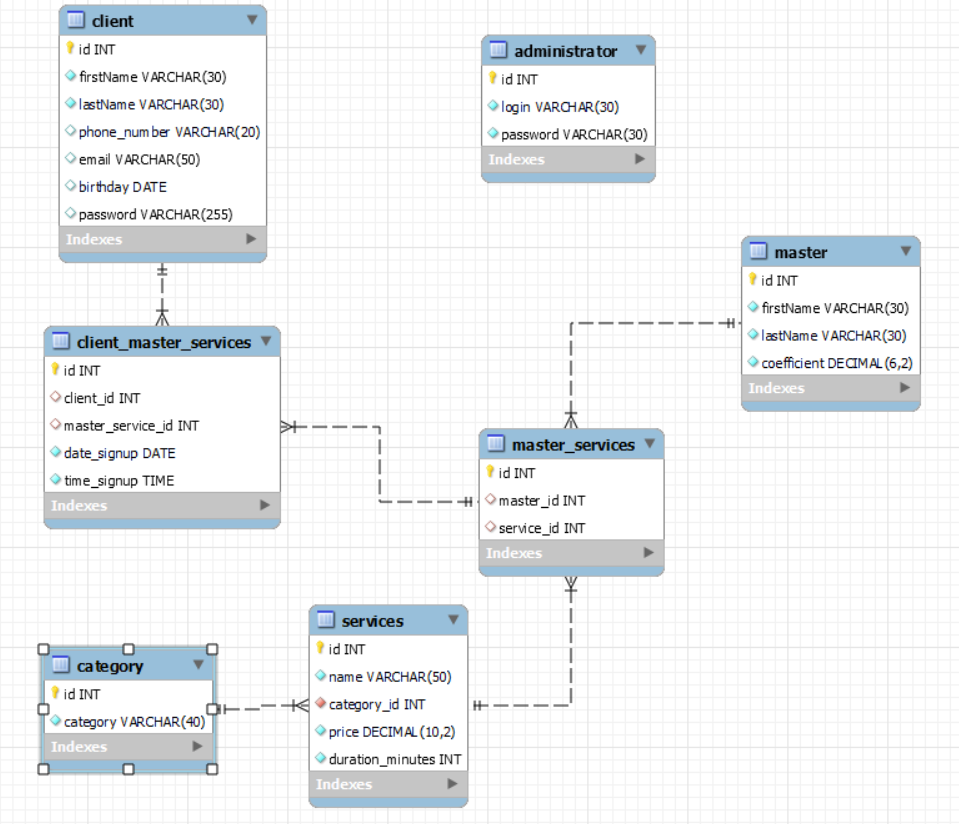


**Рисунок 1.1** – Физическая модель базы данных салона красоты, созданная в Power Designer

# **2** **ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ**

Проектирование и реализация приложения – это процесс создания программного приложения с учетом всех его функциональных требований, пользовательского интерфейса, архитектуры, базы данных и других компонентов.

В рамках данного этапа необходимо было спроектировать и реализовать информационную систему (приложение) администрирования для салона красоты. На рисунке 2.1 изображена схема базы данных вышеупомянутой ИС:



**Рисунок 2.1** – Схема базы данных

**2.1 Разработка структуры приложения**

Информационная система администрирования для салона красоты будет разрабатываться при помощи фреймворка React для клиентской части и NodeJS для серверной части.

1. Организация структуры сервера на основе NodeJS

Структура монолита в виде огромной кодовой базы с большим количеством зависимостей замедляет разработку и делает трудным внедрение новых функций. По этой причине рекомендуется разделять код на отдельные компоненты, причем для каждого компонента выделяется собственная папка для модулей компонента. Важно, чтобы каждый модуль остался маленьким и простым. Иначе разработчикам будет сложно развивать продукт: добавление нового функционала и внесение изменений в код будут производиться медленно и иметь высокий шанс поломки других зависимых компонент. В случаях, если бизнес-юниты не разделены, могут возникнуть проблемы с дальнейшим масштабированием приложения.

Примерная структура сервера на NodeJS:

1. Файл *server.js*:

В данном файле содержатся лишь основные функции промежуточной обработки, функция запуска самого Node.js приложения (сервера) и перенаправление на *routes*.

1. Конфигурации (*config*):

В директории *config* находятся конфигурационные файлы для запуска приложения.

1. Маршруты (*routes*):

Директория *routes* содержит описания всех маршрутов, причем любая взаимосвязанная совокупность маршрутов выносится в отдельный файл. *routes* определяют, как запросы от клиента должны быть направлены к соответствующим контроллерам и действиям. Они определяют соответствие URL-адресов сервера, действий и выполняемых контроллеров.

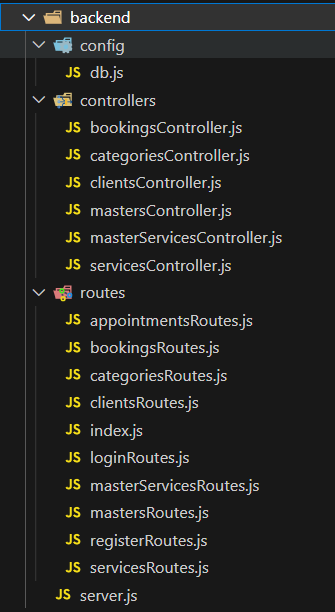
Также в *routes* присутствует файл index.js, который служит для структурированного построения маршрутов.

1. Контроллеры (*controllers*):

Директория *controllers* хранит классы, методы которого выступают обработчиками маршрутов. При этом контроллер создается для каждой взаимосвязанной совокупности маршрутов, например, для clientsRoutes создается clientsController (получить, добавить, удалить, изменить).

Структура NodeJS приложения обеспечивает организацию кода в логически связанные компоненты, упрощает разработку и обеспечивает согласованность в работе различных разработчиков.

На рисунке 2.2 изображена структура моего NodeJS приложения (сервер):



**Рисунок 2.2** – Структура NodeJS сервера

1. Организация структуры клиента на React

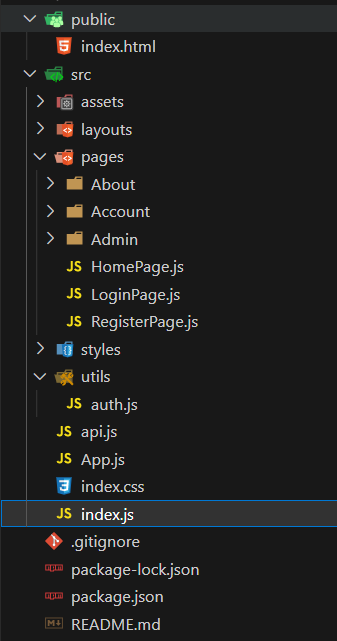
React позволяет легко создавать повторно используемые и масштабируемые приложения, используя компонентную архитектуру.

Единого мнения по структуре React-проектов не существует. Однако, есть несколько популярных подходов, которых можно придерживаться:

1. Группировка по функциональности или маршруту
2. Группировка по типу файла (группировка похожих файлов)

Для реализации приложения мною была выбрана «группировку по типу файла» для структуры приложения.

На рисунке 2.3 изображена структура моего клиентского приложения на React:



**Рисунок 2.3** – Структура клиентского приложения на React

Рассмотрим структуру приложения:

1. public/: Этот каталог содержит общедоступные ресурсы приложения, такие как index.html файл
2. src/: Основной каталог приложения, который содержит весь исходный код компонентов React, а также другие файлы и ресурсы JavaScript
3. assets/: содержит любые ресурсы, которые используются в приложении, такие как изображения или шрифты.
4. layouts/ и pages/: Компоненты макетов и страниц приложения, которые могли быть объединены в папку components/ , которая может содержать все компоненты React приложения. Причем pages содержит в своем составе папки, которые содержат компоненты для определенных страниц.
5. styles/: Каталог содержит многоразовые или глобальные стили
6. utils/: Содержит любые служебные функции или классы, которые используются во всем приложении.
7. api.js: Настраивает API соединение между клиентом и сервером
8. App.js: Это основной компонент приложения, отображающий основное содержимое приложения.
9. index.css: Cодержит глобальные стили
10. index.js: Это точка входа вашего приложения. Она инициализирует приложение React и отображает компонент приложения.

## **2.2 Программная реализация серверной части приложения**

Как описывалось ранее, приложение реализовано при помощи NodeJS, который позволяет писать серверный код для веб-страниц и веб-приложений. Node.js — не отдельный язык программирования, а платформа для использования JavaScript на стороне сервера. Если говорить о языке, то как для фронденда, так и для бэкенда используется один и тот же JavaScript. Разница только в наборе API, которые используют фронтендеры и бэкендеры.

Выбор серверного JavaScript для бэкенда обеспечивает проекту ряд преимуществ:

* Рост эффективности разработки благодаря использованию одного языка для фронт- и бэкенда и возможности переиспользования кода
* Возможность использовать npm — самый большой пакетный менеджер

Node.js отлично подходит для разработки RTA — веб-приложений, реагирующих на действия пользователя в режиме реального времени, каковым данное приложение и является.

*server.js* представляет собой точку входа для запуска сервера (листинг 2.1). Он настраивает сервер, используя фреймворк Express.js. Он использует несколько пакетов промежуточного ПО, таких как body-parser и cors для разбора тел запросов и обработки механизма междоменных запросов (CORS). Затем он определяет маршрут для API и устанавливает обработку ошибок. Когда файл запускается, он запускает сервер и начинает слушать входящие запросы на указанном порту.

const express = require('express');

const app = express();

const bodyParser = require('body-parser');

const cors = require('cors');

const routes = require('./routes/index');

app.use(cors());

app.use(bodyParser.urlencoded({ extended: true }));

app.use(bodyParser.json());

app.use('/api', routes);

app.use((err, req, res, next) => {

console.error(err.stack);

res.status(500).json({ message: 'Internal Server Error' });

});

const port = process.env.PORT || 5000;

app.listen(port, () => {

console.log(`Server is running on port ${port}`);

});

**Листинг 2.1. –** Файл server.js

Файл *index.js* в *routes* создает роутер для структурированного построения маршрутов (листинг 2.2). Он импортирует различные файлы маршрутов и настраивает точки входа для каждого из них, используя метод router.use(). Наконец, он экспортирует роутер, чтобы его можно было использовать в основном файле приложения.

const express = require('express');

const router = express.Router();

const masterServicesRoutes = require('./masterServicesRoutes');

const bookingsRoutes = require('./bookingsRoutes');

const servicesRoutes = require('./servicesRoutes');

const categoriesRoutes = require('./categoriesRoutes');

const mastersRoutes = require('./mastersRoutes');

const registerRoutes = require('./registerRoutes');

const loginRoutes = require('./loginRoutes');

const clientsRoutes = require('./clientsRoutes');

const appointmentsRoutes = require('./appointmentsRoutes');

router.use('/login', loginRoutes);

router.use('/register', registerRoutes);

router.use('/clients', clientsRoutes);

router.use('/masters', mastersRoutes);

router.use('/categories', categoriesRoutes);

router.use('/services', servicesRoutes);

router.use('/master\_services', masterServicesRoutes);

router.use('/appointments', appointmentsRoutes);

router.use('/bookings', bookingsRoutes);

module.exports = router;

**Листинг 2.2. –** Файл index.js в папке routes

Рассмотрим один из таких роутеров – *clientsRoutes.js* (листинг 2.3).

*clientsRoutes.js* настраивает несколько конечных точек для обработки HTTP-запросов по текущему маршруту (‘/api/clients’), которые соответствуют различным операциям CRUD (Create, Read, Update, Delete). После того, как роутер определит, какой контроллер использовать для обработки запроса, контроллер ответственен за осмысление запроса и генерацию подходящего ответа. Данный объект роутера экспортируется для использования в других частях приложения.

const express = require('express');

const router = express.Router();

const clientController = require('../controllers/clientsController');

router.get('/', clientController.getAllClients);

router.get('/:email', clientController.getClientByEmail);

router.post('/', clientController.createClient);

router.put('/:id', clientController.updateClient);

router.delete('/:clientId', clientController.deleteClient);

module.exports = router;

**Листинг 2.3. –** Файл clientsRoutes.js в папке routes

Функции, ответственные за обработку запросов определены в модуле *clientsController.js* (листинг 2.4), который экспортирует функции, связанные с базой данных клиентов. Функции позволяют пользователю извлекать всех клиентов, создавать нового клиента, обновлять существующего клиента, удалять клиента и извлекать клиента по электронной почте.

Функции подключаются к базе данных с именем *db* с помощью метода *query* для выполнения SQL-запросов. Если во время запроса происходит ошибка, в ответ возвращается сообщение об ошибке вместе соответствующим HTTP-кодом состояния.

const db = require('../config/db');

const getAllClients = (req, res) => {

const clientsData = 'SELECT \* FROM client';

db.query(clientsData, (err, results) => {

if (err) {

console.error(err);

return res.status(500).json({ message: 'Error retrieving clients' });

}

res.json(results);

});

};

const createClient = (req, res) => {

const { firstName, lastName, phone\_number, email, birthday } = req.body;

const insertClientQuery = 'INSERT INTO client (firstName, lastName, phone\_number, email, birthday) VALUES (?, ?, ?, ?, ?)';

const values = [firstName, lastName, phone\_number, email, birthday];

db.query(insertClientQuery, values, (err, results) => {

if (err) {

console.error(err);

return res.status(500).json({ message: 'Error creating client' });

}

res.status(201).json({ message: 'Client created successfully' });

});

};

const updateClient = (req, res) => {

const id = req.params.id;

const { firstName, lastName, phone\_number, email, birthday } = req.body;

const updateClientQuery = 'UPDATE client SET firstName=?, lastName=?, phone\_number=?, email=?, birthday=? WHERE id=?';

const values = [firstName, lastName, phone\_number, email, birthday, id];

db.query(updateClientQuery, values, (err) => {

if (err) {

console.error(err);

return res.status(500).json({ message: 'Error updating client' });

}

res.status(200).json({ message: 'Client updated successfully' });

});

};

const deleteClient = (req, res) => {

const clientId = req.params.clientId;

const deleteClientQuery = 'DELETE FROM client WHERE id=?';

const values = [clientId]

db.query(deleteClientQuery, values, (err) => {

if (err) {

console.error(err);

return res.status(500).json({ message: 'Error deleting client' });

}

res.json({ message: `Client with ID ${clientId} deleted successfully` });

});

};

const getClientByEmail = (req, res) => {

const email = req.params.email;

db.query('SELECT \* FROM client WHERE email = ?', email, (err, result) => {

if (err) {

console.error('Error retrieving user:', err);

res.status(500).json({ error: 'Error retrieving user' });

} else if (result.length === 0) {

res.status(404).json({ message: 'User not found' });

} else {

const user = result[0];

res.json(user);

}

});

};

module.exports = {

getAllClients,

createClient,

updateClient,

deleteClient,

getClientByEmail,

};

**Листинг 2.4. –** Файл clientsController.js в папке controllers

db.js в config (листинг 2.5) создает пул подключений к базе данных MySQL с указанными учетными данными и экспортирует его как модуль, который можно использовать в других частях кодовой базы.

const mysql = require('mysql2');

const pool = mysql.createPool({

host: 'localhost',

user: 'root',

password: '29112002',

database: 'beauty\_salon2',

});

module.exports = pool;

**Листинг 2.5. –** Файл db.js в папке config

## **2.3 Реализация пользовательской части приложения**

Front-end часть приложения – это пользовательский интерфейс, который может быть открыт в web-браузере пользователя. В моем случае он реализован на фреймворке React.

React JS – это библиотека, позволяющая создавать интерактивные пользовательские интерфейсы просто. В основе лежит декларативный подход к программированию, когда достаточно описать, как части интерфейса выглядят в различных состояниях, а сама библиотека будет своевременно их обновлять, когда данные изменяются без дополнительных действий.

Пользовательская часть веб приложения способна получать данные с открытого API и выводить их для пользователя, а также предоставляет возможность взаимодействовать с ними.

Точкой входа моего приложения является файл index.js (листинг 2.6), который инициализирует приложение React и отображает компонент приложения App.

import React from 'react';

import ReactDOM from 'react-dom/client';

import App from './App';

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

<React.StrictMode>

<App />

</React.StrictMode>

);

**Листинг 2.6. –** Файл index.js

Рассмотрим App.js – основной компонент приложения, отображающий основное содержимое приложения (листинг Б.1). Он использует библиотеку react-router-dom, чтобы создавать маршруты для разных страниц. Он определяет соответствие маршрута с компонентом: макетом или страницей для “Home”, “About”, “Admin”, “Account”, “Login” и “Register”. Он создает router с помощью функций createBrowserRouter и createRoutesFromElements. Функция App возвращает компонент RouterProvider с router в качестве свойства.

Файл api.js (листинг 2.7) настраивает соединение с API. Он импортирует библиотеку Axios (предоставляет удобный интерфейс для отправки запросов) для выполнения HTTP-запросов, создает экземпляр с базовым URL-адресом http://localhost:5000 и экспортирует его в качестве модуля по умолчанию.

import axios from 'axios';

const api = axios.create({

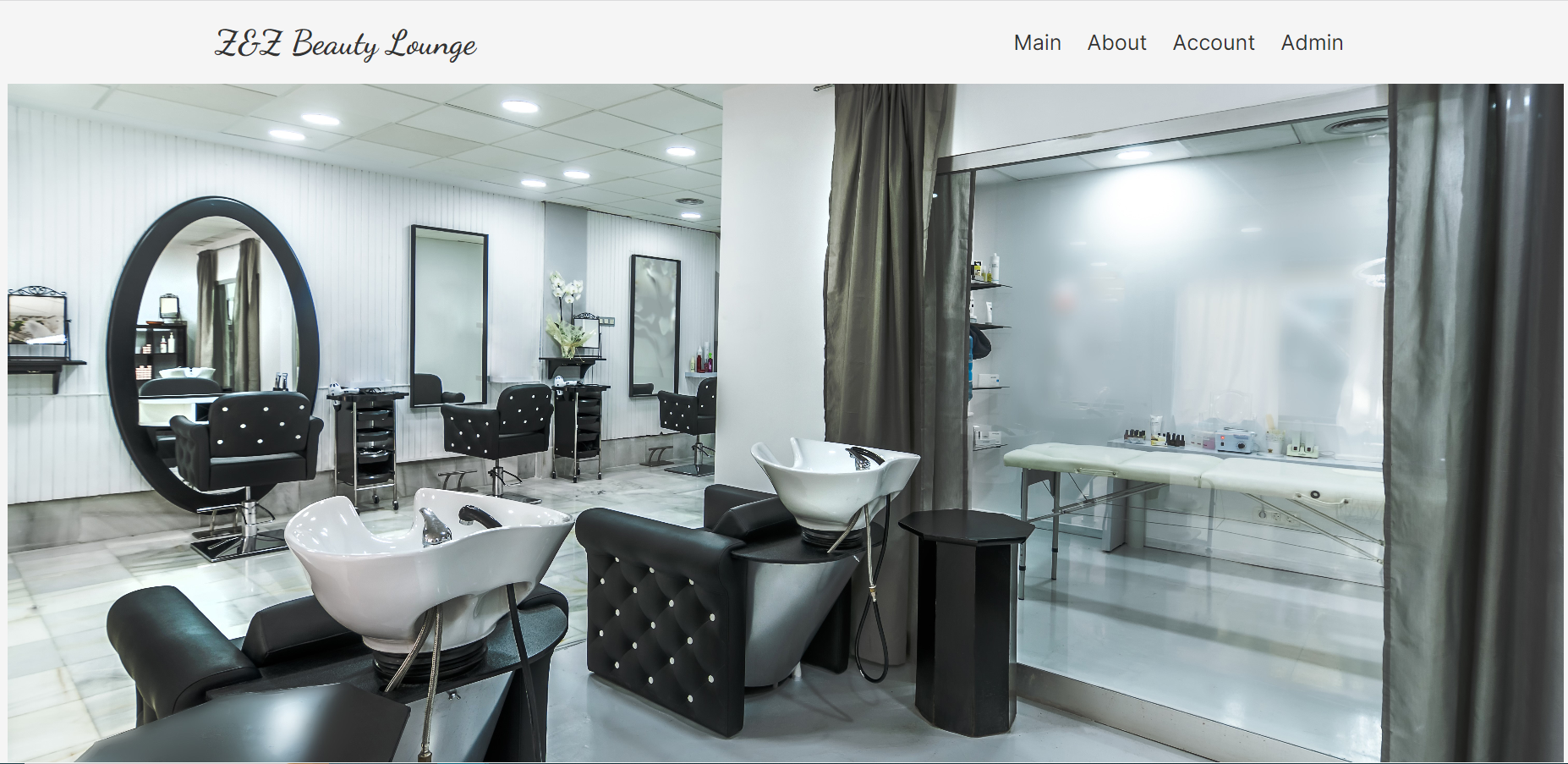
baseURL: 'http://localhost:5000',

});

export default api;

**Листинг 2.7. –** Файл api.js

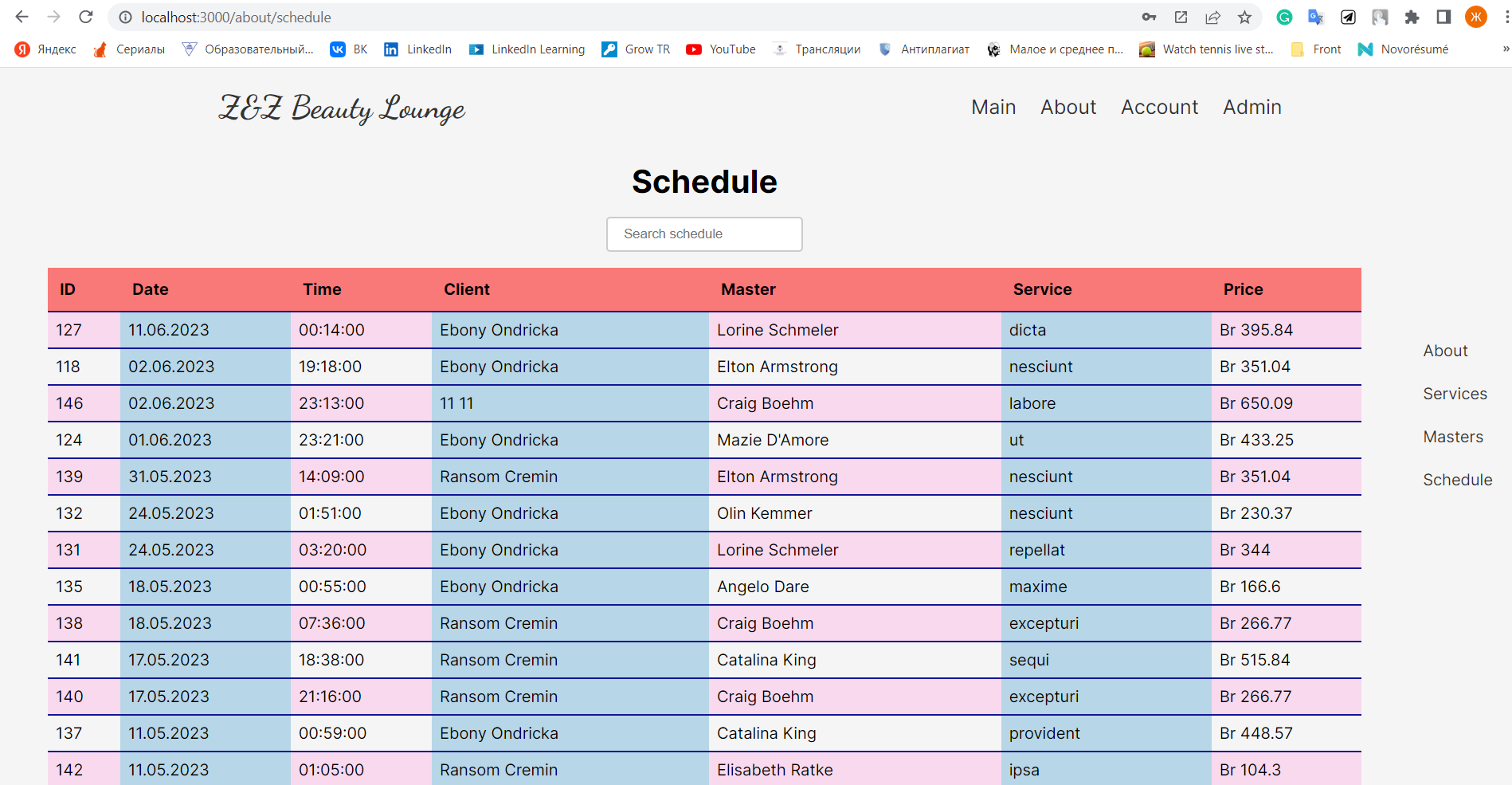
Компонент HomeLayout (листинг Б.2) будет рендериться сразу при загрузке веб-приложения, так как он является корневым компонентом приложения. Это функциональный компонент React, который рендерит шапку (header) с навигационными ссылками на разные страницы. Он также рендерит дочерние компоненты (<Outlet/>), которые располагаются по этому же маршруту (родительский компонент). Навигационные ссылки Account и Login в шапке рендерятся в зависимости от того, аутентифицирован ли пользователь или нет (рисунок 2.4).



**Рисунок 2.4** – Главная страница

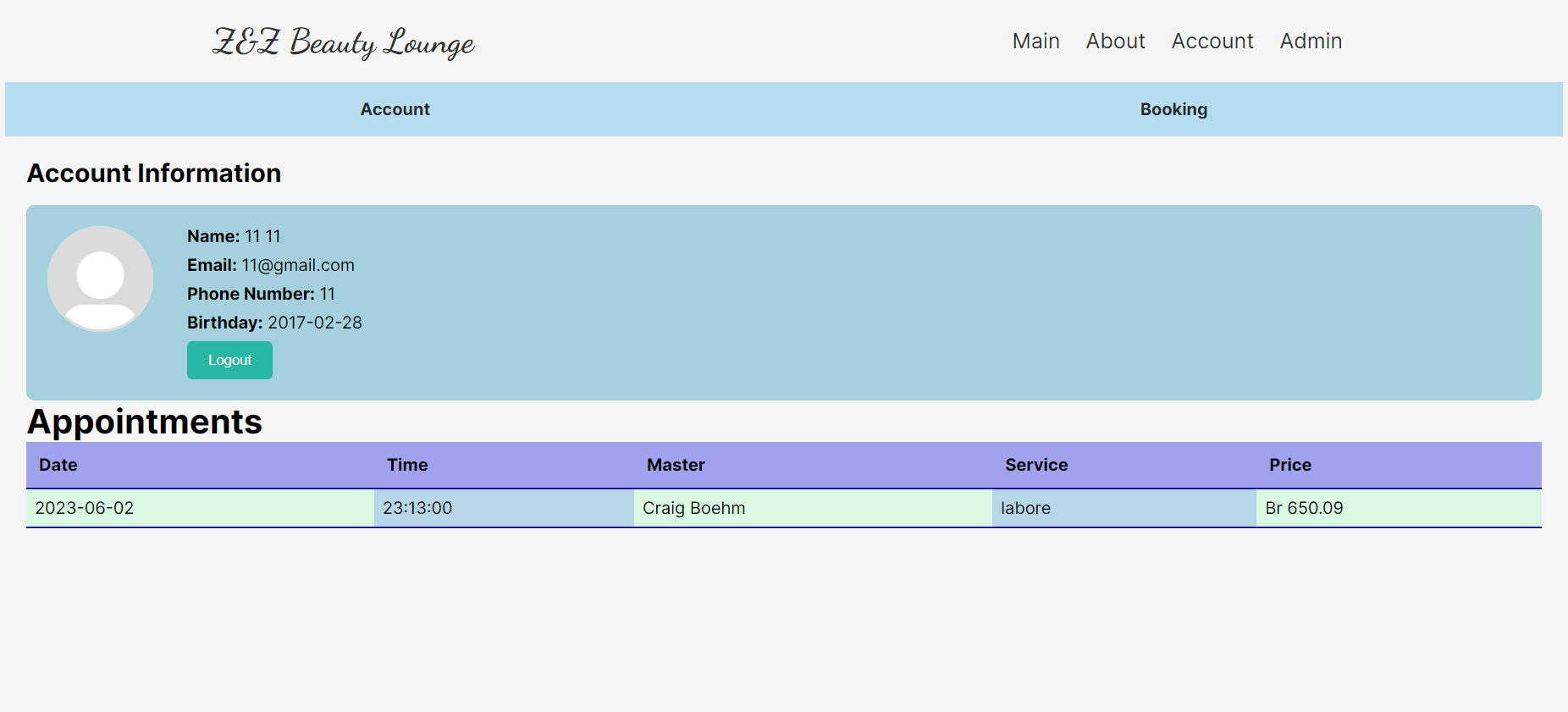
Компонент AboutLayout (листинг Б.3) будет рендериться при переходе на ‘/about’, так как он является корневым компонентом данной директории наряду с AboutPage. Он добавляет на страницу боковую панель с навигационными ссылками (компоненты Link) на различные маршруты. Также присутствует компонент Outlet – заполнитель, который отображает дочерний компонент, соответствующий текущему маршруту.

Рассмотрим Schedule (листинг Б.4) – один из дочерних компонентов AboutLayout. Schedule – функциональный компонент React, который отображает расписание записей. Он получает записи из API с помощью асинхронной функции loadAppointments(), которая получает ответ, используя api.get('/api/appointments') и затем сохраняет эти данные в appointments благодаря setAppointments(), форматирует их и отображает в таблице. Записи можно фильтровать с помощью строки поиска. Компонент использует хуки React, такие как useState() и useEffect(), для управления состоянием и выполнения побочных эффектов (при загрузке страницы).



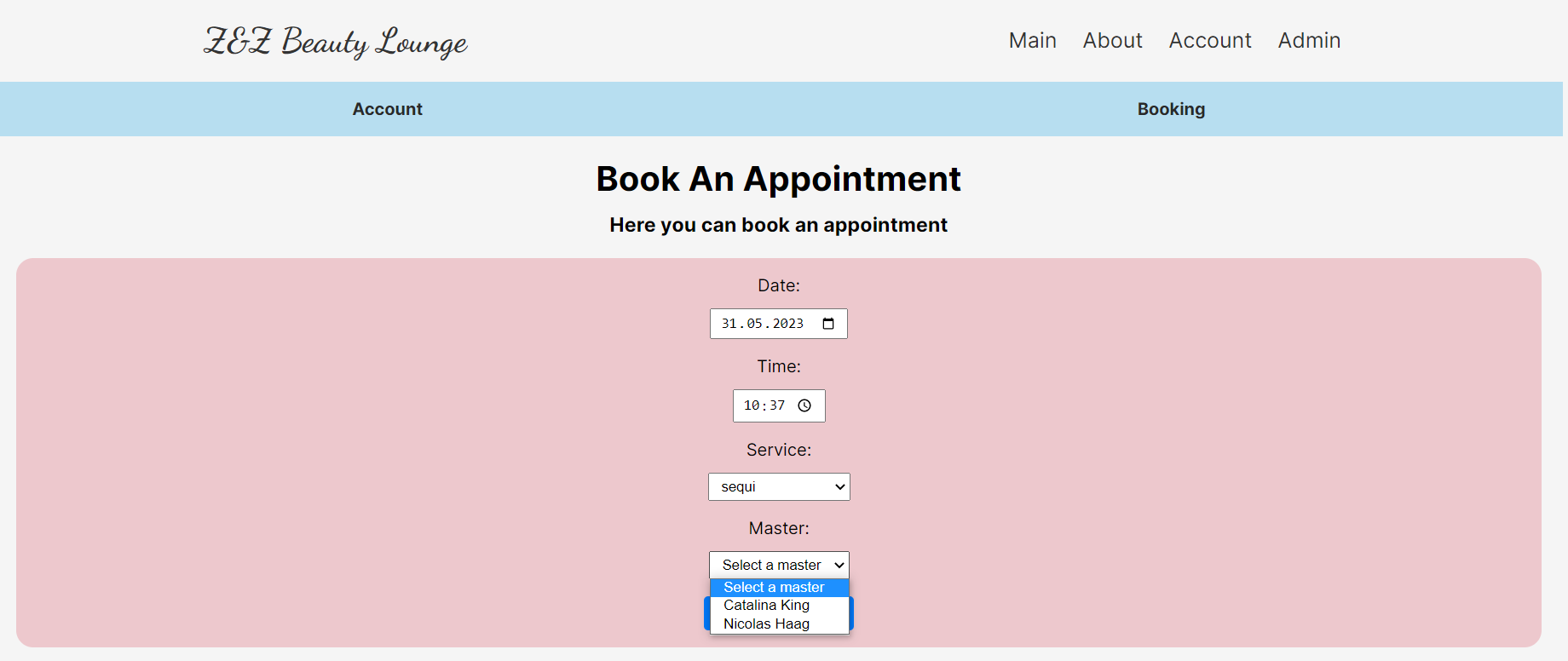
**Рисунок 2.5** – Расписание

В случае, когда клиент залогинился в системе, ему доступен личный аккаунт, в который можно перейти, кликнув на “Account” в навигации. Данная страница (рисунок 2.6) – результат работы компонентов AccountLayout (создает навигацию) и AccountPage (выводит контактные данные клиента и список всех записей клиента).



**Рисунок 2.6** – Личный аккаунт клиента

В личном аккаунте есть также и страница для записи на услуги – BookingPage, который содержит в себе компонент BookingForm. Данная форма (рисунок 2.7) содержит поля для ввода даты, времени, выбора услуги и мастера, способного оказать данную услугу клиенту. После ввода всех данных необходимо нажать на кнопку – все данные впишутся в БД, данная запись уже отображается в личном кабинете.



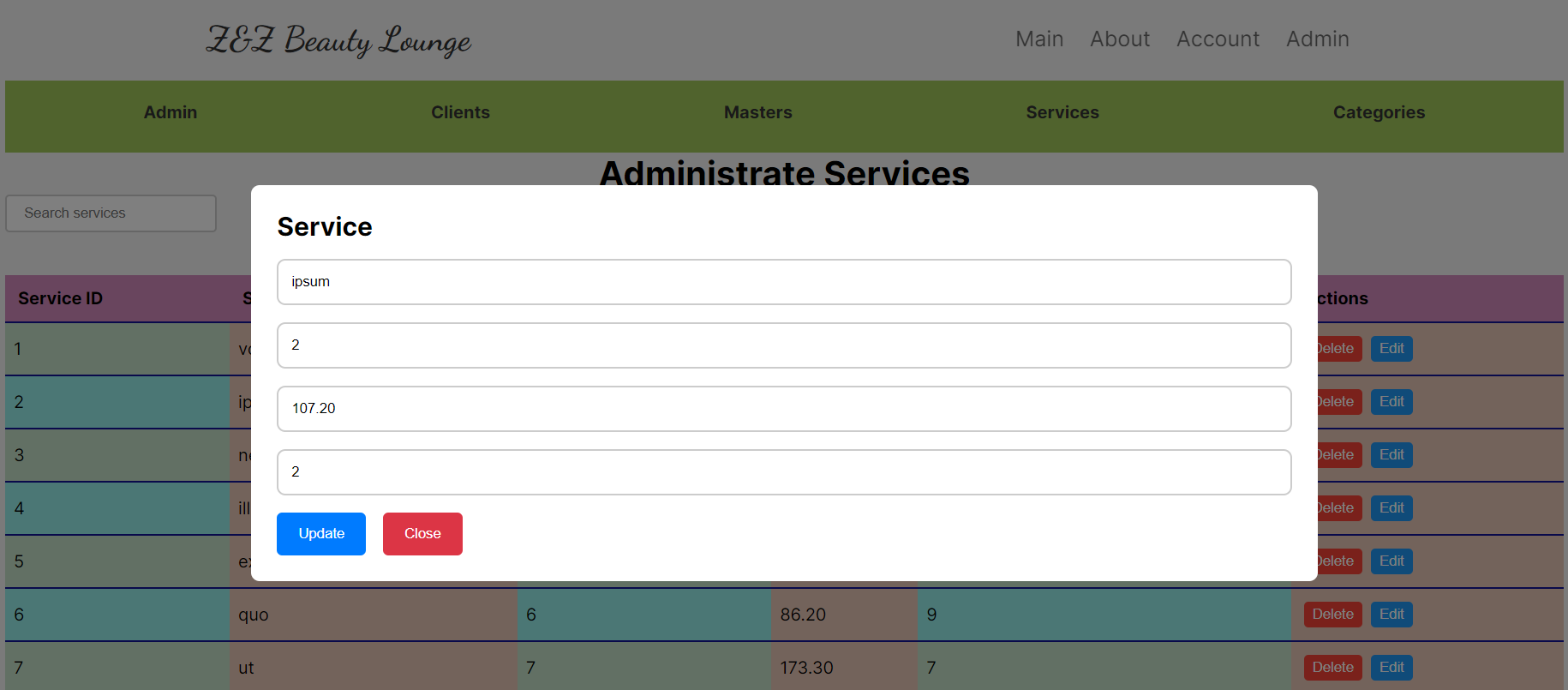
**Рисунок 2.7** – Форма для записи на услугу

BookingForm (листинг Б.5) – компонент React, который отображает форму для записи на прием. Он получает доступные услуги и соответствующих мастеров из API и заполняет выпадающие списки. Когда пользователь выбирает услугу (handleServiceChange), он получает соответствующих мастеров и выбирает мастера из списка. Когда пользователь отправляет форму (handleSubmit), он отправляет POST-запрос на конечную точку API для создания записи, предварительно получив идентификатор клиента и идентификатор мастера услуги для создания записи.

Для администратора есть отдельный раздел на сайте, в котором он может:

* получать информацию о каждом клиенте, мастере, о каждой услуге;
* добавлять, удалять и редактировать информацию о категориях, клиентах, услугах, их стоимости, мастерах;
* смотреть информацию о всех записях клиентов на услуги;
* смотреть информацию о всех записях к мастерам.

Так на рисунке 2.8 показано, что после нажатия кнопки “Edit” появляется модальная форма для редактирования данных.



**Рисунок 2.8** – Редактирование данных администратором

Компонент ServicesAdmin (листинг Б.6) отображает таблицу услуг с возможностью поиска, добавления, редактирования и удаления услуг. Он получает услуги из API при помощи fetchServices, которая сохраняет полученный список в состоянии компонента с помощью setServices. Пользователь может фильтровать их по имени.

При нажатии пользователем кнопки "Добавить" (функция handleAddButton), появляется модальное окно с полями для ввода деталей услуги.

При нажатии пользователем кнопки "Редактировать" услуги (функция handleEdit), открывается то же модальное окно с предварительно заполненными полями введенных деталей. Пользователь может изменить детали и сохранить их, нажав кнопку "Обновить".

Пользователь также может удалить услугу, нажав на кнопку "Удалить".

Пользователь может также фильтровать услуги по имени. filteredServices - это переменная, которая содержит отфильтрованный список услуг на основе поискового запроса пользователя. Она используется для отображения только соответствующих услуг в таблице.

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения курсовой работы была разработана база данных для салона красоты. База данных состоит из нескольких таблиц, которые хранят информацию о клиентах, мастерах, услугах и администраторах.

База данных позволяет хранить информацию о клиентах, мастерах, услугах и записях на услуги салона красоты, а также обеспечивает возможность управления данными для администраторов салона красоты.

Выполнены следующие задачи проекта: спроектирована и разработана база данных приложения, описаны алгоритмы и использование необходимых средств для реализации приложения, изучены основные средства и функции для работы с пользовательским интерфейсом.

Дальнейшую модификацию проекта вижу в дальнейшем улучшении веб-приложения, добавления различного функционала, внедрения с другими инструментами и сервисами, такими как электронная почта, календари, системы контроля версий и др. Это позволит создать единую экосистему рабочих инструментов и упростит процессы совместной работы и обмена информацией.

В заключении хочется отметить, что все поставленные цели и задачи курсовой работы достигнуты и решены. Реализованная система администрирования салона красоты может быть использована и дополнена для организации эффективного рабочего процесса.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Реализация масштабируемого React-приложения [Электронный ресурс] / АПНИ ‒ Режим доступа: https://apni.ru/article/4574-realizatsiya-masshtabiruemoe-react-prilozhen ‒ Дата доступа 29.05.2023

2. Лучшие практики Node.js — советы по структуре проектов [Электронный ресурс] / Хабр ‒ Режим доступа: https://habr.com/ru/articles/454476/ ‒ Дата доступа 29.05.2023

3. Организация структуры приложения [Электронный ресурс] / Nodejsdev ‒ Режим доступа: https://nodejsdev.ru/guides/webdraftt/app-structure/ ‒ Дата доступа 29.05.2023

4. React router [Электронный ресурс] / React router ‒ Режим доступа: https://reactrouter.com/en/main ‒ Дата доступа 29.05.2023

5. Built-in React Hooks [Электронный ресурс] /React ‒ Режим доступа: https://react.dev/reference/react ‒ Дата доступа 29.05.2023

6. Node.js v18.16.0 documentation [Электронный ресурс] ‒ Режим доступа: https://nodejs.org/dist/latest-v18.x/docs/api/synopsis.html ‒ Дата доступа 29.05.2023

7. Routing [Электронный ресурс] / Express ‒ Режим доступа: https://expressjs.com/en/guide/routing.html ‒ Дата доступа 29.05.2023

**ПРИЛОЖЕНИЯ**

**ПРИЛОЖЕНИЕ А**

**SQL скрипт для создания БД**

CREATE DATABASE beauty\_salon;

USE beauty\_salon;

CREATE TABLE administrator (

id INT AUTO\_INCREMENT PRIMARY KEY,

login VARCHAR(30) NOT NULL,

password VARCHAR(30) NOT NULL

);

CREATE TABLE client (

id INT AUTO\_INCREMENT PRIMARY KEY,

firstName VARCHAR(30) NOT NULL,

lastName VARCHAR(30) NOT NULL,

phone\_number VARCHAR(20),

email VARCHAR(50),

birthday DATE,

password VARCHAR(255)

);

CREATE TABLE master (

id INT AUTO\_INCREMENT PRIMARY KEY,

firstName VARCHAR(30) NOT NULL,

lastName VARCHAR(30) NOT NULL,

coefficient DECIMAL(6, 2) NOT NULL,

CHECK (coefficient>=1)

);

CREATE TABLE category (

id INT AUTO\_INCREMENT PRIMARY KEY,

category VARCHAR(40) NOT NULL

);

CREATE TABLE services (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(50) NOT NULL,

category\_id INT NOT NULL,

price DECIMAL(10, 2) NOT NULL,

duration\_minutes INT NOT NULL,

FOREIGN KEY (category\_id) REFERENCES category(id) ON DELETE RESTRICT ON UPDATE RESTRICT

);

CREATE TABLE master\_services (

id INT AUTO\_INCREMENT PRIMARY KEY,

master\_id INT,

service\_id INT,

FOREIGN KEY (master\_id) REFERENCES master(id) ON DELETE RESTRICT ON UPDATE RESTRICT,

FOREIGN KEY (service\_id) REFERENCES services(id) ON DELETE RESTRICT ON UPDATE RESTRICT

);

CREATE TABLE client\_master\_services (

id INT AUTO\_INCREMENT PRIMARY KEY,

client\_id INT,

master\_service\_id INT,

date\_signup DATE NOT NULL,

time\_signup TIME NOT NULL,

FOREIGN KEY (client\_id) REFERENCES client(id) ON DELETE RESTRICT ON UPDATE RESTRICT,

FOREIGN KEY (master\_service\_id) REFERENCES master\_services(id) ON DELETE RESTRICT ON UPDATE RESTRICT

);

**ПРИЛОЖЕНИЕ Б**

import React from 'react';

import { Route, createBrowserRouter, createRoutesFromElements, RouterProvider } from 'react-router-dom';

import './index.css';

import HomeLayout from './layouts/HomeLayout';

import AboutLayout from './layouts/AboutLayout';

import HomePage from './pages/HomePage';

import Services from './pages/About/Services';

import Masters from './pages/About/Masters';

import Schedule from './pages/About/Schedule';

import AboutPage from './pages/About/AboutPage';

import AdminLayout from './layouts/AdminLayout';

import AdminPage from './pages/Admin/AdminPage';

import ClientsAdmin from './pages/Admin/ClientsAdmin';

import MastersAdmin from './pages/Admin/MastersAdmin';

import ServicesAdmin from './pages/Admin/ServicesAdmin';

import CategoriesAdmin from './pages/Admin/CategoriesAdmin';

import AccountLayout from './layouts/AccountLayout';

import AccountPage from './pages/Account/AccountPage';

import BookingPage from './pages/Account/BookingPage';

import LoginPage from './pages/LoginPage';

import RegisterPage from './pages/RegisterPage';

const router = createBrowserRouter(

createRoutesFromElements(

<Route path="/" element={<HomeLayout />} >

<Route index element={<HomePage />} />

<Route path="about" element={<AboutLayout />} >

<Route index element={<AboutPage />} />

<Route path="services" element={<Services />} />

<Route path="masters" element={<Masters />} />

<Route path="schedule" element={<Schedule />} />

</Route>

<Route path='admin' element={<AdminLayout />}>

<Route index element={<AdminPage />} />

<Route path="clients" element={<ClientsAdmin />} />

<Route path="masters" element={<MastersAdmin />} />

<Route path="services" element={<ServicesAdmin />} />

<Route path="categories" element={<CategoriesAdmin />} />

</Route>

<Route path='account' element={<AccountLayout />}>

<Route index element={<AccountPage />} />

<Route path='booking' element={<BookingPage />} />

</Route>

<Route path='login' element={<LoginPage />} />

<Route path='register' element={<RegisterPage />} />

</Route>

)

)

function App() {

return (

<RouterProvider router={router} />

);

}

export default App;

**Листинг Б.1** – Файл App.js

import React from 'react';

import { Route, createBrowserRouter, createRoutesFromElements, RouterProvider } from 'react-router-dom';

import './index.css';

import HomeLayout from './layouts/HomeLayout';

import AboutLayout from './layouts/AboutLayout';

import HomePage from './pages/HomePage';

import Services from './pages/About/Services';

import Masters from './pages/About/Masters';

import Schedule from './pages/About/Schedule';

import AboutPage from './pages/About/AboutPage';

import AdminLayout from './layouts/AdminLayout';

import AdminPage from './pages/Admin/AdminPage';

import ClientsAdmin from './pages/Admin/ClientsAdmin';

import MastersAdmin from './pages/Admin/MastersAdmin';

import ServicesAdmin from './pages/Admin/ServicesAdmin';

import CategoriesAdmin from './pages/Admin/CategoriesAdmin';

import AccountLayout from './layouts/AccountLayout';

import AccountPage from './pages/Account/AccountPage';

import BookingPage from './pages/Account/BookingPage';

import LoginPage from './pages/LoginPage';

import RegisterPage from './pages/RegisterPage';

const router = createBrowserRouter(

createRoutesFromElements(

<Route path="/" element={<HomeLayout />} >

<Route index element={<HomePage />} />

<Route path="about" element={<AboutLayout />} >

<Route index element={<AboutPage />} />

<Route path="services" element={<Services />} />

<Route path="masters" element={<Masters />} />

<Route path="schedule" element={<Schedule />} />

</Route>

<Route path='admin' element={<AdminLayout />}>

<Route index element={<AdminPage />} />

<Route path="clients" element={<ClientsAdmin />} />

<Route path="masters" element={<MastersAdmin />} />

<Route path="services" element={<ServicesAdmin />} />

<Route path="categories" element={<CategoriesAdmin />} />

</Route>

<Route path='account' element={<AccountLayout />}>

<Route index element={<AccountPage />} />

<Route path='booking' element={<BookingPage />} />

</Route>

<Route path='login' element={<LoginPage />} />

<Route path='register' element={<RegisterPage />} />

</Route>

)

)

function App() {

return (

<RouterProvider router={router} />

);

}

export default App;

**Листинг Б.2** – Файл HomeLayout.js

import { Link, Outlet } from 'react-router-dom';

import '../styles/AboutLayout.css';

function AboutLayout() {

return (

<>

<main className="about">

<Outlet />

</main>

<aside className='about\_\_sidebar'>

<nav className='about\_\_sidebar-nav'>

<ul className='about\_\_sidebar-list'>

<li className='about\_\_sidebar-item'>

<Link to="" className='about\_\_sidebar-link'>About</Link>

</li>

<li className='about\_\_sidebar-item'>

<Link to="services" className='about\_\_sidebar-link'>Services</Link>

</li>

<li className='about\_\_sidebar-item'>

<Link to="masters" className='about\_\_sidebar-link'>Masters</Link>

</li>

<li className='about\_\_sidebar-item'>

<Link to="schedule" className='about\_\_sidebar-link'>Schedule</Link>

</li>

</ul>

</nav>

</aside>

</>

);

}

export default AboutLayout;

**Листинг Б.3** – Файл AboutLayout.js

import React, { useState, useEffect } from 'react';

import '../../styles/Schedule.css';

import api from '../../api';

const Schedule = () => {

const [appointments, setAppointments] = useState([]);

const [searchQuery, setSearchQuery] = useState('');

const loadAppointments = async () => {

const response = await api.get('/api/appointments');

setAppointments(response.data);

};

const formatDate = (date) => {

const dateObject = new Date(date);

const formattedDate = dateObject.toLocaleDateString();

return formattedDate;

}

const handleSearchChange = (e) => {

setSearchQuery(e.target.value);

};

const filteredSchedule = appointments.filter((appointment) => {

const clientFullName = `${appointment.firstName} ${appointment.lastName}`;

const clientMatch = clientFullName.toLowerCase().includes(searchQuery.toLowerCase());

const masterFullName = `${appointment.masterFirstName} ${appointment.masterLastName}`;

const masterMatch = masterFullName.toLowerCase().includes(searchQuery.toLowerCase());

const serviceMatch = appointment.serviceName.toLowerCase().includes(searchQuery.toLowerCase());

return clientMatch || masterMatch || serviceMatch;

});

useEffect(() => {

loadAppointments();

}, []);

return (

<div className="about-schedule">

<h1 className="about-schedule\_\_title">Schedule</h1>

<input

className="about-schedule\_\_search"

type="text"

value={searchQuery}

onChange={handleSearchChange}

placeholder="Search schedule"

/>

<table className="about-schedule\_\_table table">

<thead>

<tr className="table\_\_row">

<th className="table\_\_header">ID</th>

<th className="table\_\_header">Date</th>

<th className="table\_\_header">Time</th>

<th className="table\_\_header">Client</th>

<th className="table\_\_header">Master</th>

<th className="table\_\_header">Service</th>

<th className="table\_\_header">Price</th>

</tr>

</thead>

<tbody>

{filteredSchedule.map((appointment) => (

<tr className="table\_\_row" key={appointment.id}>

<td className="table\_\_data">{appointment.id}</td>

<td className="table\_\_data">{formatDate(appointment.date\_signup)}</td>

<td className="table\_\_data">{appointment.time\_signup}</td>

<td className="table\_\_data">{appointment.firstName} {appointment.lastName}</td>

<td className="table\_\_data">{appointment.masterFirstName} {appointment.masterLastName}</td>

<td className="table\_\_data">{appointment.serviceName}</td>

<td className="table\_\_data">Br {+appointment.price}</td>

</tr>

))}

</tbody>

</table>

</div>

);

};

export default Schedule;

**Листинг Б.4** – Файл Schedule.js

import React, { useState, useEffect } from 'react';

import api from '../../api';

import '../../styles/BookingForm.css';

const BookingForm = () => {

const [bookingData, setBookingData] = useState({

date: '',

time: '',

master\_id: '',

service\_id: '',

});

const [masterServicesId, setMasterServicesId] = useState();

const [clientId, setClientId] = useState();

const [masters, setMasters] = useState([]);

const [services, setServices] = useState([]);

useEffect(() => {

fetchServices();

}, []);

const fetchServices = async () => {

api.get('/api/services')

.then((response) => {

setServices(response.data);

})

.catch((error) => {

console.error('Error fetching services:', error);

});

};

const fetchMasters = async (e) => {

api.post("/api/bookings/correspondingMasters", {

service\_id: e.target.value

})

.then((response) => {

setMasters(response.data);

})

};

const handleChange = (e) => {

const { name, value } = e.target;

setBookingData((prevData) => ({

...prevData,

[name]: value,

}));

};

const handleServiceChange = (e) => {

handleChange(e);

fetchMasters(e);

}

const handleSubmit = async (e) => {

e.preventDefault();

await fetchMasterServicesId(e);

await fetchClientId();

if (masterServicesId && clientId) {

const booking = {

client\_id: clientId,

master\_service\_id: masterServicesId,

date\_signup: bookingData.date,

time\_signup: bookingData.time,

};

try {

const response = await api.post('/api/bookings/bookings', booking);

console.log('Booking submitted:', response.data);

setBookingData({

date: '',

time: '',

master\_id: '',

service\_id: '',

});

setClientId(null);

setMasterServicesId(null);

setMasters([]);

} catch (error) {

console.error('Error submitting booking:', error);

}

} else {

console.log('Click once again');

}

};

const fetchClientId = async () => {

try {

const user = JSON.parse(localStorage.getItem('user'));

if (user && user.email) {

const { email } = user;

const response = await api.post('/api/bookings/find\_client', { email });

const client = response.data;

if (client && client.id) {

setClientId(client.id);

} else {

console.error('Client not found.');

}

} else {

console.error('User email not found in localStorage');

}

} catch (error) {

console.error('Error fetching client data:', error);

}

};

const fetchMasterServicesId = async (e) => {

try {

const requestData = {

master\_id: bookingData.master\_id,

service\_id: bookingData.service\_id,

};

const response = await api.post('/api/bookings/find\_ms', requestData);

const connection = response.data;

if (connection) {

setMasterServicesId(connection.id);

} else {

console.error('No matching connection found.');

}

} catch (error) {

console.error('Error fetching services:', error);

}

};

return (

<form className="booking-form" onSubmit={handleSubmit}>

<label className="booking-form\_\_label">Date:</label>

<input

className="booking-form\_\_input"

type="date"

name="date"

value={bookingData.date}

onChange={handleChange}

/>

<label className="booking-form\_\_label">Time:</label>

<input

className="booking-form\_\_input"

type="time"

name="time"

value={bookingData.time}

onChange={handleChange}

/>

<label className="booking-form\_\_label">Service:</label>

<select

className="booking-form\_\_select"

name="service\_id"

value={bookingData.service\_id}

onChange={handleServiceChange}

>

<option value="">Select a service</option>

{services.map((service) => (

<option value={service.id} key={service.id}>

{service.name}

</option>

))}

</select>

<label className="booking-form\_\_label">Master:</label>

<select

className="booking-form\_\_select"

name="master\_id"

value={bookingData.master\_id}

onChange={handleChange}

>

<option value="">Select a master</option>

{masters.map((master) => (

<option value={master.id} key={master.id}>

{master.firstName} {master.lastName}

</option>

))}

</select>

<button className="booking-form\_\_button" type="submit">Book Appointment</button>

</form>

);

};

export { BookingForm };

**Листинг Б.5** – Файл BookingForm.js

import React, { useEffect, useState } from 'react';

import api from '../../api';

import '../../styles/ServicesAdmin.css';

const ServicesAdmin = () => {

const [services, setServices] = useState([]);

const [showModal, setShowModal] = useState(false);

const [service, setService] = useState({

id: null,

name: '',

category\_id: '',

price: '',

duration\_minutes: '',

});

const [searchQuery, setSearchQuery] = useState('');

const fetchServices = async () => {

try {

const response = await api.get('/api/services');

setServices(response.data);

} catch (error) {

console.error('Error fetching services:', error);

}

};

useEffect(() => {

fetchServices();

}, []);

const handleChange = (e) => {

setService({

...service,

[e.target.name]: e.target.value,

});

};

const toggleModal = () => {

setShowModal(!showModal);

};

const handleAddButton = async (e) => {

e.preventDefault();

try {

await api.post('/api/services', service);

fetchServices();

} catch (error) {

console.error('Error creating service:', error);

}

setService({

id: null,

name: '',

category\_id: '',

price: '',

duration\_minutes: '',

});

toggleModal();

};

const handleCloseButton = async (e) => {

setService({

id: null,

name: '',

category\_id: '',

price: '',

duration\_minutes: '',

});

toggleModal();

};

const handleDelete = async (serviceId) => {

try {

await api.delete(`/api/services/${serviceId}`);

fetchServices();

} catch (error) {

console.error('Error deleting service:', error);

}

};

const handleEdit = (service) => {

setService(service);

toggleModal();

};

const handleUpdateButton = async (e) => {

e.preventDefault();

try {

await api.put(`/api/services/${service.id}`, service);

fetchServices();

} catch (error) {

console.error('Error updating service:', error);

}

setService({

id: null,

name: '',

category\_id: '',

price: '',

duration\_minutes: '',

});

toggleModal();

};

const handleSearchChange = (e) => {

setSearchQuery(e.target.value);

};

const filteredServices = services.filter((service) =>

service.name.toLowerCase().includes(searchQuery.toLowerCase())

);

return (

<div className='admin-services'>

<h1 className='admin-services\_\_title'>Administrate Services</h1>

<input

className='admin-services\_\_search'

type='text'

value={searchQuery}

onChange={handleSearchChange}

placeholder='Search services'

/>

<button className='table\_\_button table\_\_button--add' onClick={toggleModal}>Add Service</button>

<table className="table admin-services\_\_table">

<thead>

<tr className="table\_\_row">

<th className="table\_\_header">Service ID</th>

<th className="table\_\_header">Service Name</th>

<th className="table\_\_header">Category ID</th>

<th className="table\_\_header">Price</th>

<th className="table\_\_header">Duration (Minutes)</th>

<th className="table\_\_header">Actions</th>

</tr>

</thead>

<tbody>

{filteredServices.map((service) => (

<tr className="table\_\_row" key={service.id}>

<td className='table\_\_data'>{service.id}</td>

<td className='table\_\_data'>{service.name}</td>

<td className='table\_\_data'>{service.category\_id}</td>

<td className='table\_\_data'>{service.price}</td>

<td className='table\_\_data'>{service.duration\_minutes}</td>

<td className='table\_\_data'>

<button className="table\_\_button table\_\_button--delete" onClick={() => handleDelete(service.id)}>Delete</button>

<button className="table\_\_button table\_\_button--edit" onClick={() => handleEdit(service)}>Edit</button>

</td>

</tr>

))}

</tbody>

</table>

{showModal && (

<div className="modal">

<div className="modal\_\_content">

<h2 className="modal\_\_title">Service</h2>

<input

className="modal\_\_input"

type="text"

placeholder="Service Name"

onChange={handleChange}

value={service.name}

name="name"

/>

<input

className="modal\_\_input"

type="text"

placeholder="Category ID"

onChange={handleChange}

value={service.category\_id}

name="category\_id"

/>

<input

className="modal\_\_input"

type="text"

placeholder="Price"

onChange={handleChange}

value={service.price}

name="price"

/>

<input

className="modal\_\_input"

type="text"

placeholder="Duration (Minutes)"

onChange={handleChange}

value={service.duration\_minutes}

name="duration\_minutes"

/>

{service.id ? (

<button className="modal\_\_button modal\_\_button--update" onClick={handleUpdateButton}>Update</button>

) : (

<button className="modal\_\_button modal\_\_button--add" onClick={handleAddButton}>Add</button>

)}

<button className="modal\_\_button modal\_\_button--close" onClick={handleCloseButton}>Close</button>

</div>

</div>

)}

</div>

)

}

export default ServicesAdmin;

**Листинг Б.6** – Файл ServicesAdmin.js