

**Дисциплина**

**«Database Management System»**

**Final Work**

**«База данных услуг по аренде автомобилей»**

**Студент 2 курса**

**Специальность: «Digital Engineering»**

**Belov Anton**

**Проверено:**

**Анарбекова Т. М.**

**ALMATY**

**2024**

## Содержание

1. Введение
2. Основная часть
   1. Описание предметной области
   2. Этапы реализации проекта
      * Разработка базы данных
      * Реализация серверной части
      * Реализация клиентской части
   3. Тестирование функционала
   4. Анализ использования технологий
3. Заключение
4. Список использованных источников
5. Приложения

## Введение

### Краткая характеристика проекта

Проект представляет собой разработку веб-приложения — админ-панели для управления базой данных аренды автомобилей. Приложение позволяет администраторам:

* управлять записями об автомобилях (добавлять, обновлять, удалять);
* вести учет клиентов;
* отслеживать бронирования;
* управлять платежами;
* выполнять аналитические операции, такие как подсчет доходов и статистика по статусам автомобилей
* просматривать все данные в общей таблице

Данный проект направлен на автоматизацию процессов управления данными об аренде автомобилей и является примером использования современных технологий для упрощения рутинных задач

### Цели и задачи проекта

**Цель проекта:** создать функциональное и удобное приложение для управления базой данных, соответствующее современным требованиям.

**Задачи:**

1. Разработать структуру базы данных.
2. Реализовать серверную часть для обработки запросов.
3. Создать клиентский интерфейс с использованием HTML, CSS и JavaScript.
4. Обеспечить корректную интеграцию между клиентской и серверной частями.
5. Протестировать приложение на соответствие требованиям.
6. Выполнить анализ используемых технологий и их эффективности.
7. Рассмотреть возможные сценарии расширения функционала приложения.
8. Провести оценку удобства использования интерфейса конечными пользователями.
9. Убедиться в том, что приложение обладает необходимой производительностью при обработке больших объемов данных.

### Актуальность

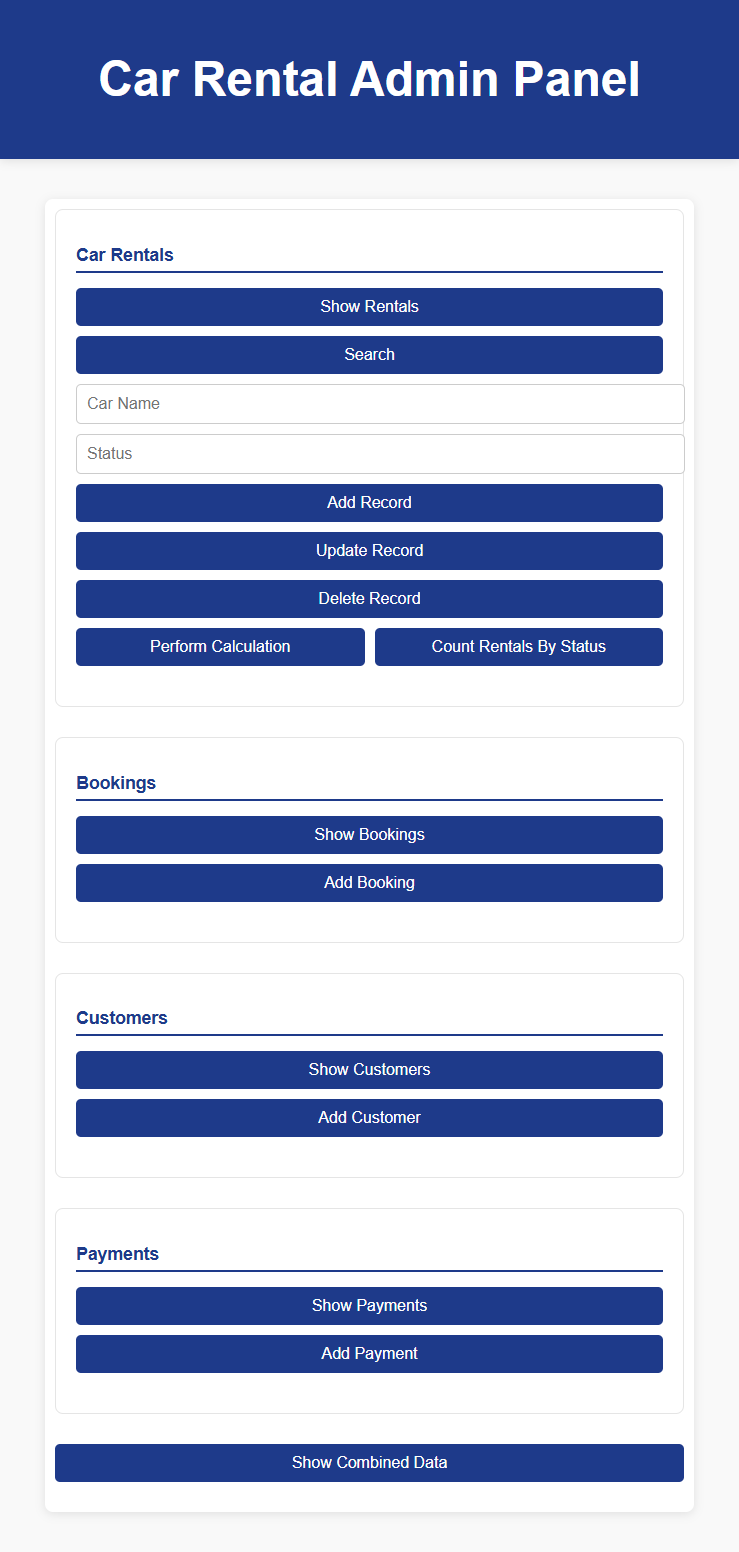
Приложение актуально для компаний, занимающихся арендой автомобилей, так как позволяет эффективно управлять данными и автоматизировать процессы. В современных условиях бизнеса эффективность управления данными напрямую влияет на конкурентоспособность компании

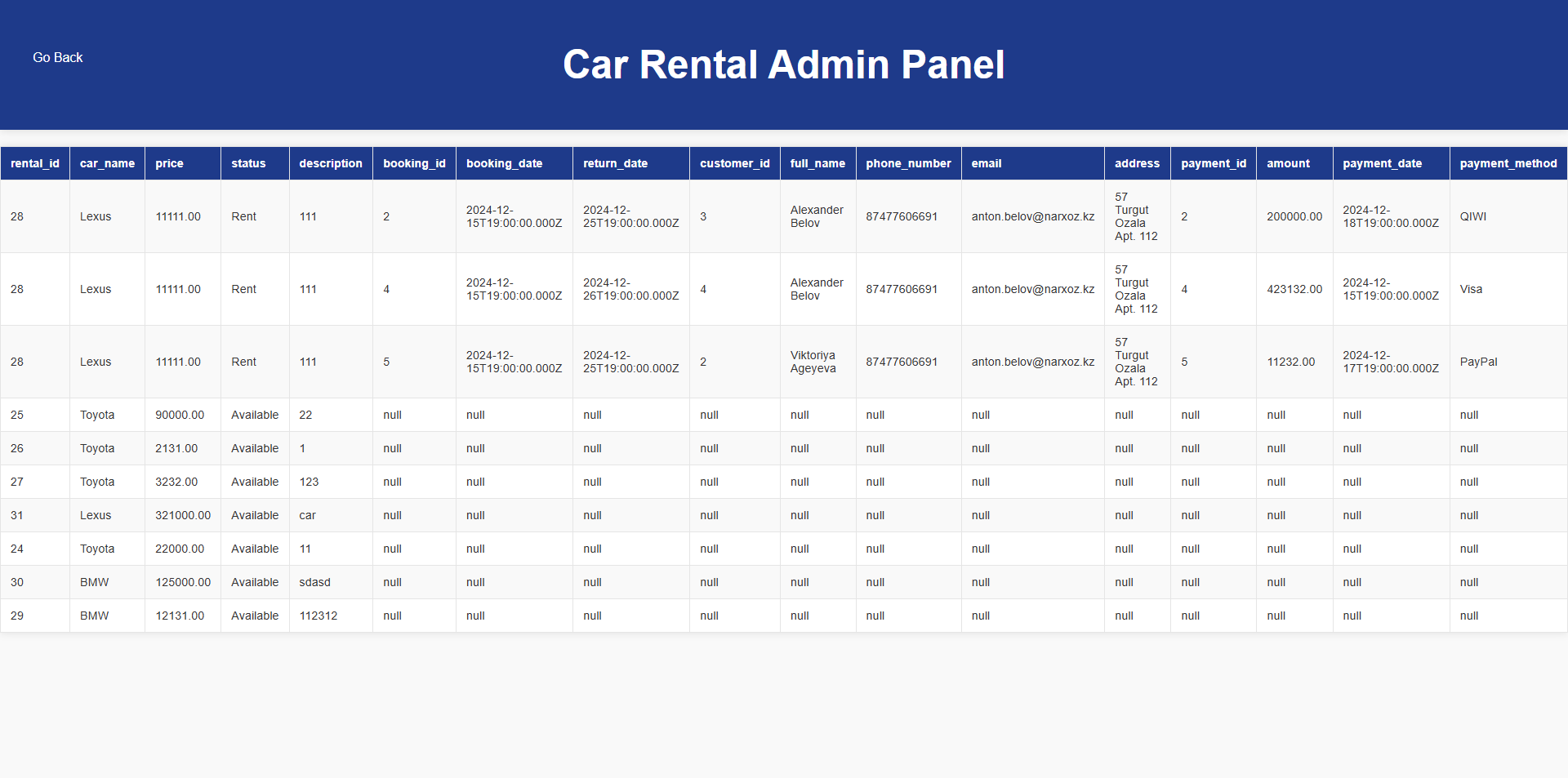
Использование цифровых решений позволяет:

* сократить трудозатраты на обработку данных
* минимизировать вероятность ошибок в данных
* ускорить процесс принятия решений

### Объект исследования

База данных аренды автомобилей, включающая следующие сущности: автомобили, клиенты, бронирования и платежи





### Методы исследования

В процессе работы были использованы следующие методы:

* Анализ требований предметной области
* Проектирование базы данных на основе нормализации
* Разработка серверной и клиентской частей с использованием современных технологий
* Тестирование готового приложения для проверки функциональности
* Анализ производительности системы
* Оценка удобства использования интерфейса на основе обратной связи от пользователей

## Основная часть

### 1. Описание предметной области

База данных аренды автомобилей должна включать следующую информацию:

* **Автомобили:** название, цена аренды, статус (доступен, арендован), описание
* **Клиенты:** ФИО, контактные данные, адрес
* **Бронирования:** связь между клиентами и автомобилями, даты бронирования и возврата
* **Платежи:** информация о связанных бронированиях, суммах и методах оплаты

Эта информация хранится в реляционной базе данных PostgreSQL. Для обеспечения целостности данных применены ограничения целостности и внешние ключи

#### Анализ требований

Для построения системы были проанализированы следующие аспекты:

* Требования к удобству интерфейса
* Возможность выполнения аналитических операций
* Безопасность данных клиентов

### Требования к интерфейсу пользователя

* Простой и интуитивно понятный дизайн
* Наличие разделения по секциям (автомобили, клиенты, бронирования, платежи)
* Возможность выполнять операции CRUD с минимальными затратами времени
* Обратная связь для пользователя в случае ошибок
* Поддержка отображения данных в табличной форме для удобства анализа
* Гибкость интерфейса для добавления новых функций в будущем

### Ожидаемые результаты работы

В результате выполнения проекта:

1. Пользователь сможет выполнять основные операции управления данными через удобный интерфейс
2. Время обработки данных будет значительно сокращено за счет автоматизации
3. Будет достигнута высокая степень защиты данных благодаря внедрению проверок и ограничений на уровне базы данных

### 2. Этапы реализации проекта

#### 2.1. Разработка базы данных

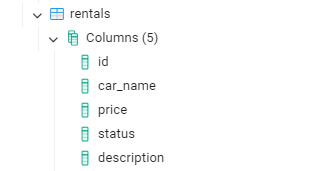
Для проекта была создана база данных с таблицами:

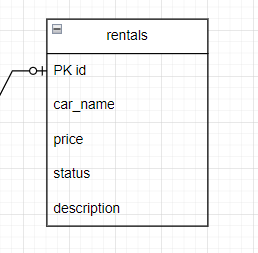
* **rentals** (id, car\_name, price, status, description)
* **customers** (id, full\_name, phone\_number, email, address)
* **bookings** (id, customer\_id, rental\_id, booking\_date, return\_date)
* **payments** (id, booking\_id, amount, payment\_date, payment\_method)

Связи между таблицами реализованы через внешние ключи. Для визуализации структуры базы данных была создана ER-диаграмма (см. Приложение)

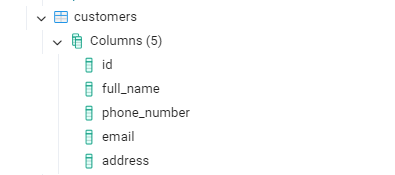
#### Подробное описание таблиц

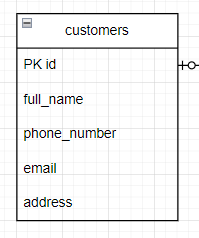
1. **Таблица rentals:**
   * Содержит информацию об автомобилях, доступных для аренды.
   * Поля: id (уникальный идентификатор), car\_name (название автомобиля), price (цена аренды), status (текущий статус автомобиля), description (описание)



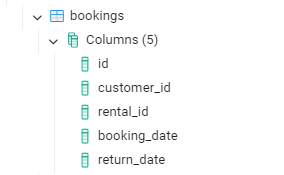


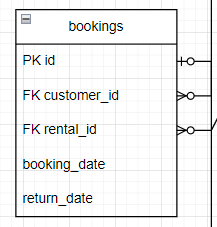
1. **Таблица customers:**
   * Хранит данные о клиентах компании.
   * Поля: id (уникальный идентификатор клиента), full\_name (ФИО клиента), phone\_number (номер телефона), email (электронная почта), address (адрес проживания)



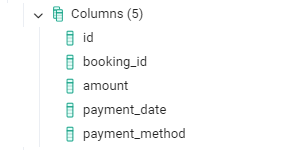
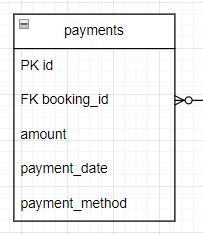


1. **Таблица bookings:**
   * Связывает клиентов и автомобили, предоставляя информацию о бронированиях.
   * Поля: id (уникальный идентификатор бронирования), customer\_id (ссылка на клиента), rental\_id (ссылка на автомобиль), booking\_date (дата бронирования), return\_date (дата возврата)





1. **Таблица payments:**
   * Отвечает за учет платежей
   * Поля: id (уникальный идентификатор платежа), booking\_id (ссылка на бронирование), amount (сумма платежа), payment\_date (дата платежа), payment\_method (метод оплаты)

#### SQL-запросы

Пример создания таблицы:

CREATE TABLE rentals (

id SERIAL PRIMARY KEY,

car\_name VARCHAR(255),

price NUMERIC,

status VARCHAR(50),

description TEXT

);

Пример добавления данных:

INSERT INTO rentals (car\_name, price, status, description)

VALUES ('Toyota Corolla', 1500, 'Available', 'Compact sedan');

Пример соединения всех таблиц в одну через JOIN таблицы:

SELECT

rentals.id AS rental\_id, rentals.car\_name, rentals.price, rentals.status, rentals.description,

bookings.id AS booking\_id, bookings.booking\_date, bookings.return\_date,

customers.id AS customer\_id, customers.full\_name, customers.phone\_number, customers.email, customers.address,

payments.id AS payment\_id, payments.amount, payments.payment\_date, payments.payment\_method

FROM rentals

LEFT JOIN bookings ON rentals.id = bookings.rental\_id

LEFT JOIN customers ON bookings.customer\_id = customers.id

LEFT JOIN payments ON bookings.id = payments.booking\_id

### 3. Тестирование функционала

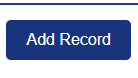
Приложение было протестировано на:

* корректность выполнения CRUD-операций (добавление, обновление, удаление);
* выполнение фильтрации и агрегатов (например, поиск автомобилей по статусу);
* корректное отображение данных в таблицах.

#### Результаты тестов

**Функция**

Добавление автомобиля





**Фактический результат**

****

****

Запись успешно добавлена

**Ожидаемый результат**

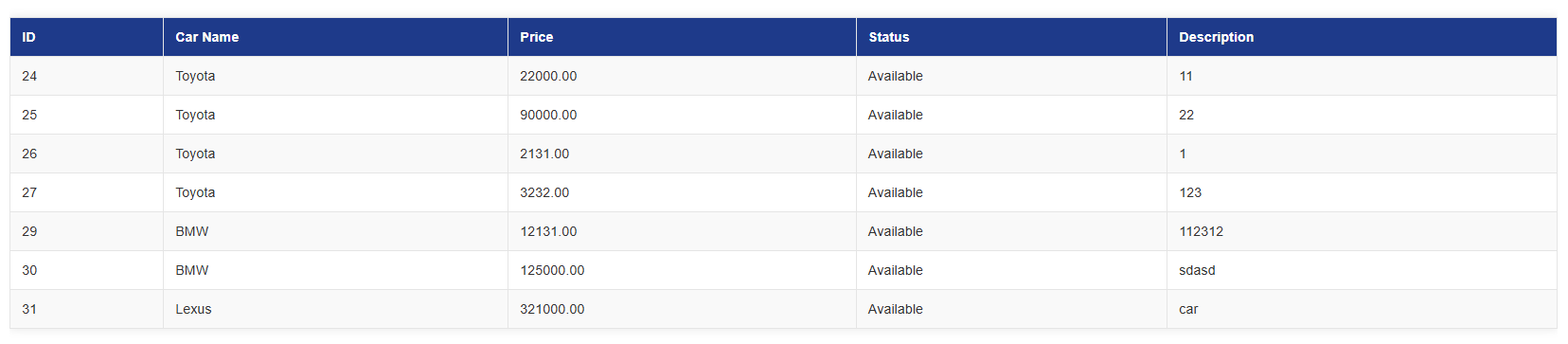
Успешно

**Функция**

Фильтрация по статусу



**Фактический результат**

****

Возвращаются только доступные авто

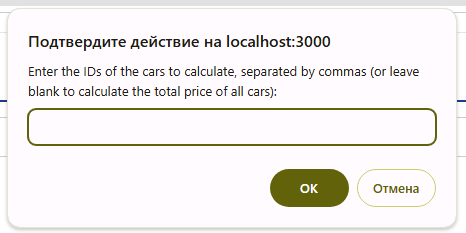
**Ожидаемый результат**

Успешно

**Функция**

Подсчет общей стоимости

****

****

**Фактический результат**

****

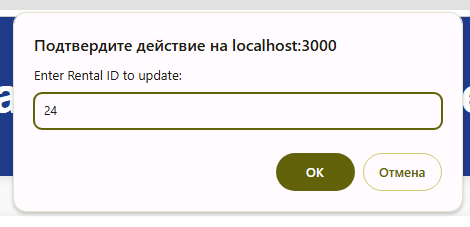
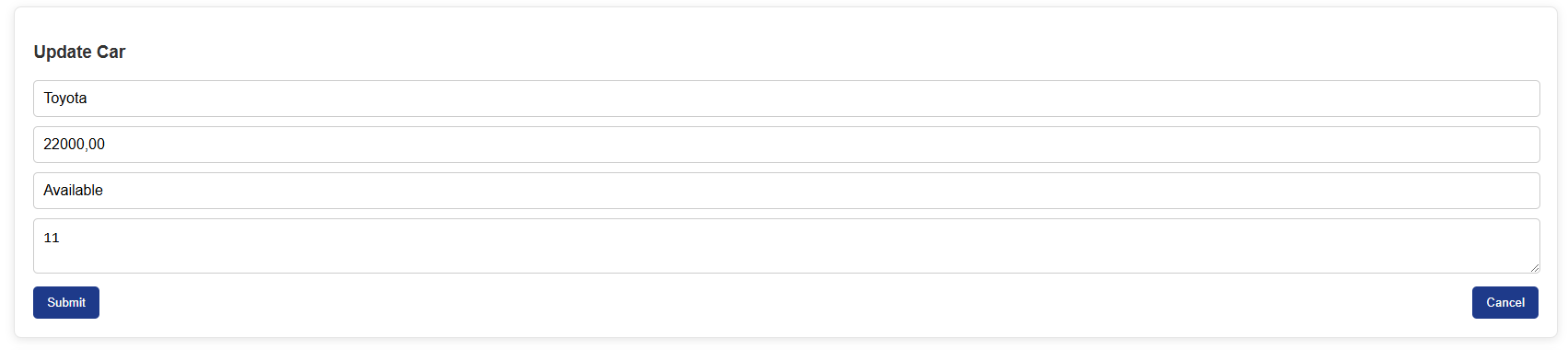
Корректный результат суммы

**Ожидаемый результат**

Успешно

**Функция**

Редактирование данных из готовой таблицы с выборкой по ID

**Фактический результат**

Изменим цену на +300$

****

****

Корректный результат изменения данных выбранной таблицы

****

****

**Ожидаемый результат**

Успешно

### 4. Анализ использования технологий

#### Серверная часть

Серверное приложение реализовано с использованием платформы **Node.js** и фреймворка **Express.js**. Эти технологии были выбраны за их гибкость, масштабируемость и широкую поддержку сообществом разработчиков.

**Основные преимущества:**

1. Быстрая обработка запросов благодаря асинхронной модели ввода-вывода.
2. Простота интеграции с базой данных PostgreSQL через библиотеку pg.
3. Возможность легкого расширения функционала за счет множества модулей.

Пример реализации эндпоинта на стороне сервера:

Javascript

app.post('/rentals', async (req, res) => {

const { car\_name, price, status, description } = req.body;

const result = await pool.query(

"INSERT INTO rentals (car\_name, price, status, description) VALUES ($1, $2, $3, $4) RETURNING \*",

[car\_name, price, status, description]

);

res.status(201).json(result.rows[0]);

});

#### Клиентская часть

Клиентская часть разработана с использованием HTML, CSS и JavaScript. Основной функционал построен на использовании **Fetch API** для отправки HTTP-запросов к серверу и динамического отображения данных на странице.

**Преимущества:**

1. Интуитивно понятный интерфейс с минималистичным дизайном
2. Динамическая загрузка данных, что повышает отзывчивость приложения
3. Простота интеграции с серверной частью

Пример вызова функции fetchData для получения данных о бронированиях:

Javascript

fetchData('/bookings', 'GET').then(data => {

displayTable(data, 'bookings');

});

#### База данных

**PostgreSQL** была выбрана в качестве СУБД благодаря ее возможностям для работы с большими объемами данных, поддержке транзакций и надежным механизмам обеспечения целостности данных.

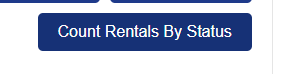
Пример использования агрегатных функций:

Sql

SELECT status, COUNT(\*) AS count

FROM rentals

GROUP BY status;





**Преимущества PostgreSQL:**

1. Поддержка сложных SQL-запросов, включая вложенные запросы.
2. Возможность масштабирования для работы с большими объемами данных.
3. Высокая производительность при обработке транзакций.

## Заключение

### Результаты проекта

В рамках проекта была разработана система для управления базой данных аренды автомобилей. Админ-панель обеспечивает полный функционал для работы с данными, включая добавление, обновление, удаление записей и выполнение аналитических операций.

**Основные достижения:**

1. Реализована структура базы данных с учетом нормализации и связей.
2. Разработан сервер, который поддерживает все основные CRUD-операции.
3. Создан удобный интерфейс для взаимодействия пользователей с данными.

### Перспективы развития

1. **Безопасность:** Реализация аутентификации и авторизации для защиты данных
2. **Визуализация данных:** Добавление графиков и диаграмм для анализа доходов и бронирований
3. **Поддержка многопользовательского режима:** Введение ролей и разграничения прав доступа
4. **Интеграция с внешними системами:** Подключение платежных систем или API для проверки автомобилей

## Список использованных источников

1. Официальная документация PostgreSQL: <https://www.postgresql.org/docs/>
2. Официальная документация Express: <https://expressjs.com/>
3. MDN Web Docs (JavaScript, Fetch API): <https://developer.mozilla.org/>

## Приложения

### Приложение 1: ER-диаграмма базы данных

### Приложение 2: Исходный код

#### 2.1. Серверная часть (server.js)

const express = require("express");

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

const { Pool } = require("pg");

const path = require("path");

const app = express();

const port = 3000;

app.use(bodyParser.json());

app.use(express.static(path.join(\_\_dirname, "frontend")));

const pool = new Pool({

    user: "postgres",

    host: "localhost",

    database: "car\_rental",

    password: "123",

    port: 5432,

});

app.get("/rentals/search", async (req, res) => {

    const { car\_name, status } = req.query;

    let query = "SELECT \* FROM rentals WHERE 1=1";

    const params = [];

    if (car\_name) {

        query += " AND car\_name ILIKE $1";

        params.push(`%${car\_name}%`);

    }

    if (status) {

        query += ` AND status = $${params.length + 1}`;

        params.push(status);

    }

    try {

        console.log('Executing query:', query);

        console.log('With parameters:', params);

        const result = await pool.query(query, params);

        console.log('Query result:', result.rows);

        if (result.rows.length > 0) {

            res.status(200).json(result.rows);

        } else {

            res.status(404).send("No rentals found");

        }

    } catch (err) {

        console.error('Error executing query:', err);

        res.status(500).send("Error searching rentals");

    }

});

app.get("/rentals", async (req, res) => {

    try {

        const result = await pool.query("SELECT \* FROM rentals");

        res.status(200).json(result.rows);

    } catch (err) {

        console.error(err);

        res.status(500).send("Error fetching rentals");

    }

});

app.get("/rentals/:id(\\d+)", async (req, res) => {

    const { id } = req.params;

    try {

        const result = await pool.query("SELECT \* FROM rentals WHERE id = $1", [id]);

        if (result.rows.length === 0) {

            return res.status(404).send("Rental not found");

        }

        res.status(200).json(result.rows[0]);

    } catch (err) {

        console.error(err);

        res.status(500).send("Error fetching rental");

    }

});

app.post("/rentals", async (req, res) => {

    const { car\_name, price, status, description } = req.body;

    try {

        const result = await pool.query(

            "INSERT INTO rentals (car\_name, price, status, description) VALUES ($1, $2, $3, $4) RETURNING \*",

            [car\_name, price, status, description]

        );

        res.status(201).json(result.rows[0]);

    } catch (err) {

        console.error(err);

        res.status(500).send("Error adding rental");

    }

});

app.put("/rentals/:id(\\d+)", async (req, res) => {

    const { id } = req.params;

    const { car\_name, price, status, description } = req.body;

    try {

        const result = await pool.query(

            "UPDATE rentals SET car\_name = $1, price = $2, status = $3, description = $4 WHERE id = $5 RETURNING \*",

            [car\_name, price, status, description, id]

        );

        if (result.rows.length === 0) {

            return res.status(404).send("Rental not found");

        }

        res.status(200).json(result.rows[0]);

    } catch (err) {

        console.error(err);

        res.status(500).send("Error updating rental");

    }

});

app.delete("/rentals/:id(\\d+)", async (req, res) => {

    const { id } = req.params;

    try {

        const result = await pool.query(

            "DELETE FROM rentals WHERE id = $1 RETURNING \*",

            [id]

        );

        if (result.rows.length === 0) {

            return res.status(404).send("Rental not found");

        }

        res.status(200).json({ message: "Rental deleted successfully" });

    } catch (err) {

        console.error(err);

        res.status(500).send("Error deleting rental");

    }

});

app.get("/rentals/calculate", async (req, res) => {

    const { ids } = req.query;

    let query = "SELECT SUM(price) AS total\_price FROM rentals";

    const params = [];

    if (ids) {

        const idArray = ids.split(',').map(id => parseInt(id, 10));

        query += " WHERE id = ANY($1)";

        params.push(idArray);

    }

    try {

        const result = await pool.query(query, params);

        if (result.rows.length === 0 || result.rows[0].total\_price === null) {

            return res.status(404).send("No rentals found for calculation");

        }

        res.status(200).json({ totalPrice: result.rows[0].total\_price });

    } catch (err) {

        console.error(err);

        res.status(500).send("Error calculating total price");

    }

});

app.get("/customers", async (req, res) => {

    try {

        const result = await pool.query("SELECT \* FROM customers");

        console.log(result.rows);

        res.status(200).json(result.rows);

    } catch (err) {

        console.error(err);

        res.status(500).send("Error fetching customers");

    }

});

app.get("/customers/:id(\\d+)", async (req, res) => {

    const { id } = req.params;

    try {

        const result = await pool.query("SELECT \* FROM customers WHERE id = $1", [id]);

        if (result.rows.length === 0) {

            return res.status(404).send("Customer not found");

        }

        res.status(200).json(result.rows[0]);

    } catch (err) {

        console.error(err);

        res.status(500).send("Error fetching customer");

    }

});

app.post("/customers", async (req, res) => {

    console.log('Received POST request to /customers with body:', req.body);

    const { full\_name, phone\_number, email, address } = req.body;

    try {

        const result = await pool.query(

            "INSERT INTO customers (full\_name, phone\_number, email, address) VALUES ($1, $2, $3, $4) RETURNING \*",

            [full\_name, phone\_number, email, address]

        );

        res.status(201).json(result.rows[0]);

    } catch (err) {

        console.error('Error adding customer:', err);

        res.status(500).send("Error adding customer");

    }

});

app.put("/customers/:id(\\d+)", async (req, res) => {

    const { id } = req.params;

    const { full\_name, phone\_number, email, address } = req.body;

    try {

        const result = await pool.query(

            "UPDATE customers SET full\_name = $1, phone\_number = $2, email = $3, address = $4 WHERE id = $5 RETURNING \*",

            [full\_name, phone\_number, email, address, id]

        );

        if (result.rows.length === 0) {

            return res.status(404).send("Customer not found");

        }

        res.status(200).json(result.rows[0]);

    } catch (err) {

        console.error(err);

        res.status(500).send("Error updating customer");

    }

});

app.delete("/customers/:id(\\d+)", async (req, res) => {

    const { id } = req.params;

    try {

        const result = await pool.query(

            "DELETE FROM customers WHERE id = $1 RETURNING \*",

            [id]

        );

        if (result.rows.length === 0) {

            return res.status(404).send("Customer not found");

        }

        res.status(200).json({ message: "Customer deleted successfully" });

    } catch (err) {

        console.error(err);

        res.status(500).send("Error deleting customer");

    }

});

app.get("/bookings", async (req, res) => {

    try {

        const result = await pool.query("SELECT \* FROM bookings");

        res.status(200).json(result.rows);

    } catch (err) {

        console.error(err);

        res.status(500).send("Error fetching bookings");

    }

});

app.get("/bookings/:id(\\d+)", async (req, res) => {

    const { id } = req.params;

    try {

        const result = await pool.query("SELECT \* FROM bookings WHERE id = $1", [id]);

        if (result.rows.length === 0) {

            return res.status(404).send("Booking not found");

        }

        res.status(200).json(result.rows[0]);

    } catch (err) {

        console.error(err);

        res.status(500).send("Error fetching booking");

    }

});

app.post("/bookings", async (req, res) => {

    const { customer\_id, rental\_id, booking\_date, return\_date } = req.body;

    try {

        const result = await pool.query(

            "INSERT INTO bookings (customer\_id, rental\_id, booking\_date, return\_date) VALUES ($1, $2, $3, $4) RETURNING \*",

            [customer\_id, rental\_id, booking\_date, return\_date]

        );

        res.status(201).json(result.rows[0]);

    } catch (err) {

        console.error(err);

        res.status(500).send("Error adding booking");

    }

});

app.put("/bookings/:id(\\d+)", async (req, res) => {

    const { id } = req.params;

    const { customer\_id, rental\_id, booking\_date, return\_date } = req.body;

    try {

        const rentalCheck = await pool.query("SELECT status FROM rentals WHERE id = $1", [rental\_id]);

        if (rentalCheck.rows.length === 0) {

            return res.status(404).send("Rental not found");

        }

        if (rentalCheck.rows[0].status === 'Rent') {

            console.log(`Car with ID ${rental\_id} is already rented`);

            return res.status(400).send("Car is already rented");

        }

        const result = await pool.query(

            "UPDATE bookings SET customer\_id = $1, rental\_id = $2, booking\_date = $3, return\_date = $4 WHERE id = $5 RETURNING \*",

            [customer\_id, rental\_id, booking\_date, return\_date, id]

        );

        res.status(200).json(result.rows[0]);

    } catch (err) {

        console.error(err);

        res.status(500).send("Error updating booking");

    }

});

app.delete("/bookings/:id(\\d+)", async (req, res) => {

    const { id } = req.params;

    try {

        const result = await pool.query(

            "DELETE FROM bookings WHERE id = $1 RETURNING \*",

            [id]

        );

        if (result.rows.length === 0) {

            return res.status(404).send("Booking not found");

        }

        res.status(200).json({ message: "Booking deleted successfully", booking: result.rows[0] });

    } catch (err) {

        console.error('Error deleting booking:', err);

        res.status(500).send("Error deleting booking");

    }

});

app.get("/payments", async (req, res) => {

    try {

        const result = await pool.query("SELECT \* FROM payments");

        res.status(200).json(result.rows);

    } catch (err) {

        console.error(err);

        res.status(500).send("Error fetching payments");

    }

});

app.get("/rentals/status-count", async (req, res) => {

    try {

        const result = await pool.query(`

            SELECT status, COUNT(\*) AS count

            FROM rentals

            GROUP BY status

        `);

        res.status(200).json(result.rows);

    } catch (err) {

        console.error('Error fetching rental counts by status:', err);

        res.status(500).send("Error fetching rental counts by status");

    }

});

app.get("/payments/:id(\\d+)", async (req, res) => {

    const { id } = req.params;

    try {

        const result = await pool.query("SELECT \* FROM payments WHERE id = $1", [id]);

        if (result.rows.length === 0) {

            return res.status(404).send("Payment not found");

        }

        res.status(200).json(result.rows[0]);

    } catch (err) {

        console.error(err);

        res.status(500).send("Error fetching payment");

    }

});

app.post("/payments", async (req, res) => {

    const { booking\_id, amount, payment\_date, payment\_method } = req.body;

    try {

        const result = await pool.query(

            "INSERT INTO payments (booking\_id, amount, payment\_date, payment\_method) VALUES ($1, $2, $3, $4) RETURNING \*",

            [booking\_id, amount, payment\_date, payment\_method]

        );

        res.status(201).json(result.rows[0]);

    } catch (err) {

        console.error(err);

        res.status(500).send("Error adding payment");

    }

});

app.post("/trigger-delete-booking/:id(\\d+)", async (req, res) => {

    const { id } = req.params;

    try {

        const result = await pool.query(

            "DELETE FROM bookings WHERE id = $1 RETURNING \*",

            [id]

        );

        if (result.rows.length === 0) {

            return res.status(404).send("Booking not found");

        }

        res.status(200).json({ message: "Booking deleted successfully, rental status updated", booking: result.rows[0] });

    } catch (err) {

        console.error('Error triggering delete booking:', err);

        res.status(500).send("Error triggering delete booking");

    }

});

app.put("/payments/:id(\\d+)", async (req, res) => {

    const { id } = req.params;

    const { booking\_id, amount, payment\_date, payment\_method } = req.body;

    try {

        const result = await pool.query(

            "UPDATE payments SET booking\_id = $1, amount = $2, payment\_date = $3, payment\_method = $4 WHERE id = $5 RETURNING \*",

            [booking\_id, amount, payment\_date, payment\_method, id]

        );

        if (result.rows.length === 0) {

            return res.status(404).send("Payment not found");

        }

        res.status(200).json(result.rows[0]);

    } catch (err) {

        console.error(err);

        res.status(500).send("Error updating payment");

    }

});

app.delete("/payments/:id(\\d+)", async (req, res) => {

    const { id } = req.params;

    try {

        const result = await pool.query(

            "DELETE FROM payments WHERE id = $1 RETURNING \*",

            [id]

        );

        if (result.rows.length === 0) {

            return res.status(404).send("Payment not found");

        }

        res.status(200).json({ message: "Payment deleted successfully" });

    } catch (err) {

        console.error(err);

        res.status(500).send("Error deleting payment");

    }

});

app.get("/combined-data", async (req, res) => {

    try {

        const result = await pool.query(`

            SELECT

                rentals.id AS rental\_id, rentals.car\_name, rentals.price, rentals.status, rentals.description,

                bookings.id AS booking\_id, bookings.booking\_date, bookings.return\_date,

                customers.id AS customer\_id, customers.full\_name, customers.phone\_number, customers.email, customers.address,

                payments.id AS payment\_id, payments.amount, payments.payment\_date, payments.payment\_method

            FROM rentals

            LEFT JOIN bookings ON rentals.id = bookings.rental\_id

            LEFT JOIN customers ON bookings.customer\_id = customers.id

            LEFT JOIN payments ON bookings.id = payments.booking\_id

        `);

        res.status(200).json(result.rows);

    } catch (err) {

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

        res.status(500).send("Error fetching combined data");

    }

});

app.post("/bookings", async (req, res) => {

    const { customer\_id, rental\_id, booking\_date, return\_date } = req.body;

    try {

        const rentalCheck = await pool.query("SELECT status FROM rentals WHERE id = $1", [rental\_id]);

        if (rentalCheck.rows.length === 0) {

            return res.status(404).send("Rental not found");

        }

        if (rentalCheck.rows[0].status === 'Rent') {

            console.log(`Car with ID ${rental\_id} is already rented`);

            return res.status(400).send("Car is already rented");

        }

        const result = await pool.query(

            "INSERT INTO bookings (customer\_id, rental\_id, booking\_date, return\_date) VALUES ($1, $2, $3, $4) RETURNING \*",

            [customer\_id, rental\_id, booking\_date, return\_date]

        );

        await pool.query("UPDATE rentals SET status = 'Rent' WHERE id = $1", [rental\_id]);

        res.status(201).json(result.rows[0]);

    } catch (err) {

        console.error(err);

        res.status(500).send("Error adding booking");

    }

});

app.get("/", (req, res) => {

    res.sendFile(path.join(\_\_dirname, "frontend", "index.html"));

});

app.listen(port, () => {

  console.log(`Server is running on http://localhost:${port}`);

});

#### 2.2. Клиентская часть

* index.html
* <!DOCTYPE html>
* <html lang="en">
* <head>
* <meta charset="UTF-8" />
* <meta name="viewport" content="width=device-width, initial-scale=1.0" />
* <title>Car Rental Admin Panel</title>
* <link rel="stylesheet" href="style.css" />
* </head>
* <body>
* <header>
* <div class="header-container">
* <h1>Car Rental Admin Panel</h1>
* </div>
* </header>
* <main>
* <div class="dashboard">
* <section id="car-rentals" class="section">
* <h2>Car Rentals</h2>
* <div class="buttons">
* <button id="showData">Show Rentals</button>
* <button id="searchData">Search</button>
* <input type="text" id="carNameInput" placeholder="Car Name" />
* <input type="text" id="statusInput" placeholder="Status" />
* <button id="addRecord">Add Record</button>
* <button id="updateRecord">Update Record</button>
* <button id="deleteRecord">Delete Record</button>
* <div class="group-tworow">
* <button id="calculate">Perform Calculation</button>
* <button id="countByStatus">Count Rentals By Status</button>
* </div>
* </div>
* <div id="results"></div>
* </section>
* <div id="formContainer" class="form-container">
* <h3>Add New Car</h3>
* <form id="addCarForm">
* <input type="text" id="newCarName" placeholder="Car Name" required />
* <input type="number" id="newCarPrice" placeholder="Price" required />
* <input type="text" id="newCarStatus" placeholder="Status" required />
* <textarea id="newCarDescription" placeholder="Description" required></textarea>
* <button type="submit">Submit</button>
* <button type="button" id="cancelAdd">Cancel</button>
* </form>
* </div>
* <div id="updateFormContainer" class="form-container">
* <h3>Update Car</h3>
* <form id="updateCarForm">
* <input type="hidden" id="updateCarId" />
* <input type="text" id="updateCarName" placeholder="Car Name" required />
* <input type="number" id="updateCarPrice" placeholder="Price" required />
* <input type="text" id="updateCarStatus" placeholder="Status" required />
* <textarea id="updateCarDescription" placeholder="Description" required></textarea>
* <div class="group-tworow">
* <button type="submit">Submit</button>
* <button type="button" id="cancelUpdate">Cancel</button>
* </div>
* </form>
* </div>
* <div id="statusCountResults"></div>
* <section id="bookings" class="section">
* <h2>Bookings</h2>
* <div class="buttons">
* <button id="showBookings">Show Bookings</button>
* <button id="addBooking">Add Booking</button>
* </div>
* <div id="bookingsResults"></div>
* </section>
* <div id="bookingFormContainer" class="form-container">
* <h3 id="bookingFormTitle">Add New Booking</h3>
* <form id="bookingForm">
* <input type="hidden" id="bookingId" />
* <input type="number" id="bookingCustomerId" placeholder="Customer ID" required />
* <input type="number" id="bookingRentalId" placeholder="Rental ID" required />
* <input type="date" id="bookingDate" placeholder="Booking Date" required />
* <input type="date" id="returnDate" placeholder="Return Date" />
* <button type="submit">Submit</button>
* <button type="button" id="cancelBooking">Cancel</button>
* </form>
* </div>
* <section id="customers" class="section">
* <h2>Customers</h2>
* <div class="buttons">
* <button id="showCustomers">Show Customers</button>
* <button id="addCustomer">Add Customer</button>
* </div>
* <div id="customersResults"></div>
* </section>
* <div id="customerFormContainer" class="form-container">
* <h3 id="customerFormTitle">Add New Customer</h3>
* <form id="customerForm">
* <input type="hidden" id="customerId" />
* <input type="text" id="customerName" placeholder="Full Name" required />
* <input type="text" id="customerPhone" placeholder="Phone Number" required />
* <input type="email" id="customerEmail" placeholder="Email" />
* <textarea id="customerAddress" placeholder="Address" required></textarea>
* <button type="submit">Submit</button>
* <button type="button" id="cancelCustomer">Cancel</button>
* </form>
* </div>
* <section id="payments" class="section">
* <h2>Payments</h2>
* <div class="buttons">
* <button id="showPayments">Show Payments</button>
* <button id="addPayment">Add Payment</button>
* </div>
* <div id="paymentsResults"></div>
* </section>
* <div id="paymentFormContainer" class="form-container">
* <h3 id="paymentFormTitle">Add New Payment</h3>
* <form id="paymentForm">
* <input type="hidden" id="paymentId" />
* <input type="number" id="paymentBookingId" placeholder="Booking ID" required />
* <input type="number" id="paymentAmount" placeholder="Amount" required />
* <input type="date" id="paymentDate" placeholder="Payment Date" required />
* <input type="text" id="paymentMethod" placeholder="Payment Method" required />
* <button type="submit">Submit</button>
* <button type="button" id="cancelPayment">Cancel</button>
* </form>
* </div>
* <section>
* <div class="buttons btn-ctr">
* <button onclick="location.href='table.html?endpoint=/combined-data'">
* Show Combined Data
* </button>
* </div>
* </section>
* </div>
* </main>
* <div id="combinedDataResults"></div>
* <script src="script.js"></script>
* </body>
* </html>
* Style.css
* body {
* font-family: 'Arial', sans-serif;
* background-color: #f9f9f9;
* margin: 0;
* padding: 0;
* color: #333;
* }
* header {
* background-color: #1e3a8a;
* color: #fff;
* padding: 20px;
* text-align: center;
* font-size: 1.5rem;
* box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
* }
* main {
* padding: 20px;
* }
* .dashboard {
* width: 90%;
* margin: 20px auto;
* background: #ffffff;
* border-radius: 8px;
* box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
* padding: 20px;
* }
* .section {
* margin-bottom: 30px;
* padding: 20px;
* border: 1px solid #e5e5e5;
* border-radius: 8px;
* }
* .section h2 {
* margin-bottom: 15px;
* font-size: 1.3rem;
* color: #1e3a8a;
* border-bottom: 2px solid #1e3a8a;
* padding-bottom: 5px;
* }
* .buttons {
* display: flex;
* flex-wrap: wrap;
* gap: 10px;
* margin-bottom: 20px;
* }
* .buttons button {
* background-color: #1e3a8a;
* color: white;
* padding: 10px 20px;
* border: none;
* border-radius: 5px;
* font-size: 1rem;
* cursor: pointer;
* transition: background-color 0.3s ease;
* }
* .buttons button:hover {
* background-color: #163176;
* }
* .buttons input {
* flex-grow: 1;
* padding: 10px;
* border: 1px solid #ccc;
* border-radius: 5px;
* font-size: 1rem;
* }
* #results, #bookingsResults, #customersResults, #paymentsResults {
* margin-top: 20px;
* overflow-x: auto;
* }
* table {
* width: 100%;
* border-collapse: collapse;
* margin: 20px 0;
* font-size: 0.9rem;
* background: #fff;
* box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
* }
* table th, table td {
* padding: 12px;
* border: 1px solid #e5e5e5;
* text-align: left;
* }
* table th {
* background-color: #1e3a8a;
* color: #fff;
* }
* table tr:nth-child(even) {
* background-color: #f9f9f9;
* }
* table tr:hover {
* background-color: #f1f1f1;
* }
* .form-container {
* display: none;
* padding: 20px;
* background-color: #fff;
* border: 1px solid #e5e5e5;
* border-radius: 8px;
* box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
* margin-top: 20px;
* }
* form input, form textarea {
* width: calc(100% - 20px);
* padding: 10px;
* margin-bottom: 10px;
* border: 1px solid #ccc;
* border-radius: 5px;
* font-size: 1rem;
* }
* form button[type="submit"],
* form button[type="button"] {
* background-color: #1e3a8a;
* color: white;
* padding: 10px 15px;
* border: none;
* border-radius: 5px;
* cursor: pointer;
* transition: background-color 0.3s ease;
* }
* form button[type="submit"]:hover,
* form button[type="button"]:hover {
* background-color: #163176;
* }
* @media (max-width: 768px) {
* .dashboard {
* padding: 10px;
* }
* .section h2 {
* font-size: 1.1rem;
* }
* .buttons {
* flex-direction: column;
* gap: 10px;
* }
* .buttons button,
* .buttons input {
* width: 100%;
* }
* }
* footer {
* background-color: #1e3a8a;
* color: #fff;
* text-align: center;
* padding: 10px;
* position: fixed;
* bottom: 0;
* width: 100%;
* font-size: 0.9rem;
* }
* .group-tworow {
* display: flex;
* justify-content: space-between;
* width: 100%;
* gap: 10px;
* }
* .showfulldata {
* text-align: center;
* }
* .btn-ctr {
* display: flex;
* justify-content: center;
* margin-top: 20px;
* }
* script.js
* async function fetchData(endpoint, method = 'GET', body = null) {
* const options = { method, headers: { 'Content-Type': 'application/json' } };
* if (body) options.body = JSON.stringify(body);
* try {
* console.log(`Requesting ${endpoint} with method ${method} and body ${JSON.stringify(body)}`);
* const response = await fetch(endpoint, options);
* if (!response.ok) {
* throw new Error(`HTTP error! status: ${response.status}`);
* }
* const data = await response.json();
* return data;
* } catch (error) {
* console.error('Error:', error);
* document.getElementById('results').textContent = 'An error occurred';
* }
* }
* function displayTable(data, type) {
* const resultsDiv = document.getElementById('results');
* if (!data || data.length === 0) {
* resultsDiv.innerHTML = 'No data found';
* return;
* }
* let table = '<table><tr>';
* if (type === 'customers') {
* table += '<th>ID</th><th>Full Name</th><th>Phone Number</th><th>Email</th><th>Address</th>';
* } else if (type === 'bookings') {
* table += '<th>ID</th><th>Customer ID</th><th>Rental ID</th><th>Booking Date</th><th>Return Date</th>';
* } else if (type === 'payments') {
* table += '<th>ID</th><th>Booking ID</th><th>Amount</th><th>Payment Date</th><th>Payment Method</th>';
* } else {
* table += '<th>ID</th><th>Car Name</th><th>Price</th><th>Status</th><th>Description</th>';
* }
* table += '</tr>';
* data.forEach(item => {
* table += '<tr>';
* if (type === 'customers') {
* table += `<td>${item.id}</td><td>${item.full\_name}</td><td>${item.phone\_number}</td><td>${item.email}</td><td>${item.address}</td>`;
* } else if (type === 'bookings') {
* table += `<td>${item.id}</td><td>${item.customer\_id}</td><td>${item.rental\_id}</td><td>${item.booking\_date}</td><td>${item.return\_date}</td>`;
* } else if (type === 'payments') {
* table += `<td>${item.id}</td><td>${item.booking\_id}</td><td>${item.amount}</td><td>${item.payment\_date}</td><td>${item.payment\_method}</td>`;
* } else {
* table += `<td>${item.id}</td><td>${item.car\_name}</td><td>${item.price}</td><td>${item.status}</td><td>${item.description}</td>`;
* }
* table += '</tr>';
* });
* table += '</table>';
* resultsDiv.innerHTML = table;
* }
* document.getElementById('showCustomers').addEventListener('click', () => {
* fetchData('/customers').then(data => {
* displayTable(data, 'customers');
* });
* });
* document.getElementById('showData').addEventListener('click', () => {
* fetchData('/rentals').then(data => {
* displayTable(data);
* });
* });
* document.getElementById('searchData').addEventListener('click', () => {
* const carName = document.getElementById('carNameInput').value.trim();
* const status = document.getElementById('statusInput').value.trim();
* let queryParams = `?car\_name=${encodeURIComponent(carName)}`;
* if (status) {
* queryParams += `&status=${encodeURIComponent(status)}`;
* }
* fetchData(`/rentals/search${queryParams}`).then(data => {
* displayTable(data);
* }).catch(error => {
* console.error('Error:', error);
* document.getElementById('results').textContent = 'An error occurred';
* });
* });
* document.getElementById('addRecord').addEventListener('click', () => {
* document.getElementById('formContainer').style.display = 'block';
* });
* document.getElementById('addCarForm').addEventListener('submit', (e) => {
* e.preventDefault();
* const newRental = {
* car\_name: document.getElementById('newCarName').value,
* price: parseFloat(document.getElementById('newCarPrice').value),
* status: document.getElementById('newCarStatus').value,
* description: document.getElementById('newCarDescription').value
* };
* fetchData('/rentals', 'POST', newRental).then(data => {
* if (data) {
* document.getElementById('results').textContent = JSON.stringify(data, null, 2);
* }
* });
* document.getElementById('formContainer').style.display = 'none';
* document.getElementById('addCarForm').reset();
* });
* document.getElementById('cancelAdd').addEventListener('click', () => {
* document.getElementById('formContainer').style.display = 'none';
* document.getElementById('addCarForm').reset();
* });
* document.getElementById('updateRecord').addEventListener('click', async () => {
* const rentalId = prompt('Enter Rental ID to update:');
* const rental = await fetchData(`/rentals/${rentalId}`);
* if (rental) {
* document.getElementById('updateCarId').value = rental.id;
* document.getElementById('updateCarName').value = rental.car\_name;
* document.getElementById('updateCarPrice').value = rental.price;
* document.getElementById('updateCarStatus').value = rental.status;
* document.getElementById('updateCarDescription').value = rental.description;
* document.getElementById('updateFormContainer').style.display = 'block';
* } else {
* document.getElementById('results').textContent = 'Rental not found';
* }
* });
* document.getElementById('updateCarForm').addEventListener('submit', (e) => {
* e.preventDefault();
* const updatedRental = {
* car\_name: document.getElementById('updateCarName').value,
* price: parseFloat(document.getElementById('updateCarPrice').value),
* status: document.getElementById('updateCarStatus').value,
* description: document.getElementById('updateCarDescription').value
* };
* const rentalId = document.getElementById('updateCarId').value;
* fetchData(`/rentals/${rentalId}`, 'PUT', updatedRental).then(data => {
* if (data) {
* document.getElementById('results').textContent = JSON.stringify(data, null, 2);
* }
* });
* document.getElementById('updateFormContainer').style.display = 'none';
* document.getElementById('updateCarForm').reset();
* });
* document.getElementById('cancelUpdate').addEventListener('click', () => {
* document.getElementById('updateFormContainer').style.display = 'none';
* document.getElementById('updateCarForm').reset();
* });
* document.getElementById('deleteRecord').addEventListener('click', () => {
* const rentalId = prompt('Enter Rental ID to delete:');
* fetchData(`/rentals/${rentalId}`, 'DELETE').then(data => {
* if (data) {
* document.getElementById('results').textContent = JSON.stringify(data, null, 2);
* }
* });
* });
* document.getElementById('calculate').addEventListener('click', () => {
* const selectedCars = prompt('Enter the IDs of the cars to calculate, separated by commas (or leave blank to calculate the total price of all cars):');
* let queryParams = '';
* if (selectedCars) {
* const ids = selectedCars.split(',').map(id => id.trim()).join(',');
* queryParams = `?ids=${ids}`;
* }
* fetchData(`/rentals/calculate${queryParams}`).then(data => {
* if (data) {
* document.getElementById('results').textContent = `Total Price: $${data.totalPrice}`;
* } else {
* document.getElementById('results').textContent = 'No rentals found for calculation';
* }
* }).catch(error => {
* console.error('Error:', error);
* document.getElementById('results').textContent = 'An error occurred';
* });
* });
* document.getElementById('showBookings').addEventListener('click', () => {
* fetchData('/bookings').then(data => {
* displayTable(data, 'bookings');
* });
* });
* document.getElementById('addBooking').addEventListener('click', () => {
* document.getElementById('bookingFormContainer').style.display = 'block';
* document.getElementById('bookingFormTitle').innerText = 'Add New Booking';
* });
* document.addEventListener('DOMContentLoaded', function() {
* document.getElementById('bookingForm').addEventListener('submit', (e) => {
* e.preventDefault();
* const newBooking = {
* customer\_id: document.getElementById('bookingCustomerId').value,
* rental\_id: document.getElementById('bookingRentalId').value,
* booking\_date: document.getElementById('bookingDate').value,
* return\_date: document.getElementById('returnDate').value
* };
* fetchData('/bookings', 'POST', newBooking).then(data => {
* displayTable([data], 'bookings');
* }).catch(error => {
* console.error('Error adding booking:', error);
* if (error.message.includes('400')) {
* document.getElementById('resultsBookings').textContent = 'Error adding booking: Car is already rented';
* } else {
* document.getElementById('resultsBookings').textContent = 'Error adding booking';
* }
* });
* document.getElementById('bookingFormContainer').style.display = 'none';
* document.getElementById('bookingForm').reset();
* });
* async function fetchData(endpoint, method = 'GET', body = null) {
* const options = { method, headers: { 'Content-Type': 'application/json' } };
* if (body) options.body = JSON.stringify(body);
* try {
* const response = await fetch(endpoint, options);
* if (!response.ok) {
* throw new Error(`HTTP error! status: ${response.status}`);
* }
* const data = await response.json();
* return data;
* } catch (error) {
* console.error('Error:', error);
* throw error;
* }
* }
* });
* document.getElementById('cancelBooking').addEventListener('click', () => {
* document.getElementById('bookingFormContainer').style.display = 'none';
* document.getElementById('bookingForm').reset();
* });
* document.getElementById('addCustomer').addEventListener('click', () => {
* document.getElementById('customerFormContainer').style.display = 'block';
* document.getElementById('customerFormTitle').innerText = 'Add New Customer';
* });
* document.getElementById('customerForm').addEventListener('submit', (e) => {
* e.preventDefault();
* const newCustomer = {
* full\_name: document.getElementById('customerName').value,
* phone\_number: document.getElementById('customerPhone').value,
* email: document.getElementById('customerEmail').value,
* address: document.getElementById('customerAddress').value
* };
* fetchData('/customers', 'POST', newCustomer).then(data => {
* if (data) {
* displayTable([data], 'customers');
* }
* });
* document.getElementById('customerFormContainer').style.display = 'none';
* document.getElementById('customerForm').reset();
* });
* document.getElementById('cancelCustomer').addEventListener('click', () => {
* document.getElementById('customerFormContainer').style.display = 'none';
* document.getElementById('customerForm').reset();
* });
* document.getElementById('showPayments').addEventListener('click', () => {
* fetchData('/payments').then(data => {
* displayTable(data, 'payments');
* });
* });
* document.getElementById('addPayment').addEventListener('click', () => {
* document.getElementById('paymentFormContainer').style.display = 'block';
* document.getElementById('paymentFormTitle').innerText = 'Add New Payment';
* });
* document.getElementById('paymentForm').addEventListener('submit', (e) => {
* e.preventDefault();
* const newPayment = {
* booking\_id: document.getElementById('paymentBookingId').value,
* amount: document.getElementById('paymentAmount').value,
* payment\_date: document.getElementById('paymentDate').value,
* payment\_method: document.getElementById('paymentMethod').value
* };
* fetchData('/payments', 'POST', newPayment).then(data => {
* if (data) {
* displayTable([data], 'payments');
* }
* });
* document.getElementById('paymentFormContainer').style.display = 'none';
* document.getElementById('paymentForm').reset();
* });
* document.getElementById('cancelPayment').addEventListener('click', () => {
* document.getElementById('paymentFormContainer').style.display = 'none';
* document.getElementById('paymentForm').reset();
* });
* function displayStatusCount(data) {
* const resultsDiv = document.getElementById('statusCountResults');
* if (!data || data.length === 0) {
* resultsDiv.innerHTML = 'No data found';
* return;
* }
* let table = '<table><tr><th>Status</th><th>Count</th></tr>';
* data.forEach(item => {
* table += `<tr><td>${item.status}</td><td>${item.count}</td></tr>`;
* });
* table += '</table>';
* resultsDiv.innerHTML = table;
* }
* document.getElementById('countByStatus').addEventListener('click', () => {
* fetchData('/rentals/status-count').then(data => {
* displayStatusCount(data);
* }).catch(error => {
* console.error('Error:', error);
* document.getElementById('statusCountResults').textContent = 'An error occurred';
* });
* });
* function displayCombinedData(data) {
* const resultsDiv = document.getElementById('combinedDataResults');
* if (!data || data.length === 0) {
* resultsDiv.innerHTML = 'No data found';
* return;
* }
* let table = '<table><tr><th>Rental ID</th><th>Car Name</th><th>Price</th><th>Status</th><th>Description</th><th>Booking ID</th><th>Booking Date</th><th>Return Date</th><th>Customer ID</th><th>Full Name</th><th>Phone Number</th><th>Email</th><th>Address</th><th>Payment ID</th><th>Amount</th><th>Payment Date</th><th>Payment Method</th></tr>';
* data.forEach(item => {
* table += `<tr>
* <td>${item.rental\_id}</td>
* <td>${item.car\_name}</td>
* <td>${item.price}</td>
* <td>${item.status}</td>
* <td>${item.description}</td>
* <td>${item.booking\_id}</td>
* <td>${item.booking\_date}</td>
* <td>${item.return\_date}</td>
* <td>${item.customer\_id}</td>
* <td>${item.full\_name}</td>
* <td>${item.phone\_number}</td>
* <td>${item.email}</td>
* <td>${item.address}</td>
* <td>${item.payment\_id}</td>
* <td>${item.amount}</td>
* <td>${item.payment\_date}</td>
* <td>${item.payment\_method}</td>
* </tr>`;
* });
* table += '</table>';
* resultsDiv.innerHTML = table;
* }
* document.getElementById('showCombinedData').addEventListener('click', () => {
* fetchData('/combined-data').then(data => {
* displayCombinedData(data);
* }).catch(error => {
* console.error('Error:', error);
* document.getElementById('combinedDataResults').textContent = 'An error occurred';
* });
* });
* document.addEventListener('DOMContentLoaded', function() {
* document.getElementById('triggerDeleteBooking').addEventListener('click', () => {
* const bookingId = prompt('Enter Booking ID to delete:');
* if (bookingId) {
* fetchData(`/trigger-delete-booking/${bookingId}`, 'POST').then(data => {
* document.getElementById('triggerResults').textContent = JSON.stringify(data, null, 2);
* }).catch(error => {
* console.error('Error triggering delete booking:', error);
* document.getElementById('triggerResults').textContent = 'An error occurred';
* });
* } else {
* document.getElementById('triggerResults').textContent = 'Booking ID is required';
* }
* });
* });
* document.addEventListener('DOMContentLoaded', function() {
* function deleteBooking() {
* const bookingId = prompt('Enter Booking ID to delete:');
* if (bookingId) {
* fetchData(`/bookings/${bookingId}`, 'DELETE').then(data => {
* document.getElementById('resultsBookings').textContent = JSON.stringify(data, null, 2);
* }).catch(error => {
* console.error('Error deleting booking:', error);
* document.getElementById('resultsBookings').textContent = 'An error occurred';
* });
* } else {
* document.getElementById('resultsBookings').textContent = 'Booking ID is required';
* }
* }
* document.getElementById('deleteBooking').addEventListener('click', deleteBooking);
* });
* async function fetchData(endpoint, method = 'GET', body = null) {
* const options = { method, headers: { 'Content-Type': 'application/json' } };
* if (body) options.body = JSON.stringify(body);
* try {
* const response = await fetch(endpoint, options);
* if (!response.ok) {
* throw new Error(`HTTP error! status: ${response.status}`);
* }
* const data = await response.json();
* return data;
* } catch (error) {
* console.error('Error:', error);
* throw error;
* }
* }
* table.html и table.js
* <!DOCTYPE html>
* <html lang="en">
* <head>
* <meta charset="UTF-8" />
* <meta name="viewport" content="width=device-width, initial-scale=1.0" />
* <title>Combined Data</title>
* <link rel="stylesheet" href="style.css" />
* </head>
* <body>
* <div class="container">
* <header>
* <h1 style="position: relative;">
* <div style="position: absolute; left: 0; top: 0;" class="buttons">
* <button onclick="window.history.back()">Go Back</button>
* </div>
* Car Rental Admin Panel
* </h1>
* </header>
* <div id="tableContainer">
* </div>
* </div>
* <script src="table.js"></script>
* </body>
* </html>
* document.addEventListener('DOMContentLoaded', function() {
* const urlParams = new URLSearchParams(window.location.search);
* const endpoint = urlParams.get('endpoint');
* async function loadData(endpoint) {
* const response = await fetch(endpoint);
* if (!response.ok) {
* throw new Error(`HTTP error! status: ${response.status}`);
* }
* const data = await response.json();
* displayTable(data);
* }
* function displayTable(data) {
* const tableContainer = document.getElementById('tableContainer');
* if (!data || data.length === 0) {
* tableContainer.innerHTML = 'No data found';
* return;
* }
* let table = '<table><tr>';
* const keys = Object.keys(data[0]);
* keys.forEach(key => {
* table += `<th>${key}</th>`;
* });
* table += '</tr>';
* data.forEach(item => {
* table += '<tr>';
* keys.forEach(key => {
* table += `<td>${item[key]}</td>`;
* });
* table += '</tr>';
* });
* table += '</table>';
* tableContainer.innerHTML = table;
* }
* if (endpoint) {
* loadData(endpoint).catch(error => {
* console.error('Error loading data:', error);
* document.getElementById('tableContainer').textContent = 'An error occurred';
* });
* } else {
* document.getElementById('tableContainer').textContent = 'No endpoint specified';
* }
* });