Міністерство освіти і науки України
Харківський національний університет радіоелектроніки
Кафедра програмної інженерії

Лабораторна робота №2
з дисципліни: «Архітектура програмного забезпечення»
на тему: «РОЗРОБКА СЕРВЕРНОЇ ЧАСТИНИ ПРОГРАМНОЇ СИСТЕМИ»

Виконав
студент групи ПЗПІ-22-2
Верясов Владислав Олексійович
28 травня 2025 р.
Перевірив
Старший викладач кафедри ПІ
Сокорчук Ігор Петрович

Харків 2025

МЕТА

Метою цієї лабораторної роботи є розробити розробити серверну/бек-енд частину програмної системи для підтримки та регуляції прокату електротранспорту.

ЗАВДАННЯ

Робота передбачає створення будови програмної системи з описом її компонентів, UML діаграми прецедентів для серверної частини, ER діаграми даних, бази даних (БД) системи та її структури, функцій роботи з БД (ORM або CoRM), API (REST, GraphQL, gRPC) для клієнт-серверної взаємодії, а також специфікації API. Необхідно реалізувати програмний код API та функцій роботи з БД, перевірити його функціональність, завантажити код у GitHub репозиторій, створити відеодемонстрацію перевірки роботи серверної частини, завантажити відео на YouTube, додати хронологічний опис до відео, створити звіт, вказати у звіті посилання на відео, експортувати звіт у PDF для завантаження на dl.nure.ua та у текстовий файл (UTF-8) для запису в GitHub.

ХІД РОБОТИ

Назва: «Система для підтримки та регуляції прокату електротранспорту»
Назва англійською мовою: «Electric Transport Rental Management System»
Власна назва: «E-Transport».

На першому етапі було створено ER-діаграму даних, яка визначила структуру бази даних та її взаємозв’язки. Це стало основою для подальшого створення бази даних і забезпечило логічну організацію даних. ER-діаграма представлена у додатку А.

На другому етапі було розроблено базу даних на основі створеної ER-діаграми. Для цього використовувалася PostgreSQL як основна СУБД, а адміністрування бази виконувалося через PgAdmin4. Усі необхідні таблиці, зв’язки та обмеження було реалізовано.

На третьому етапі було створено діаграму структури бази даних, яка наочно демонструє зв’язки між таблицями та їх ключові атрибути. Це допомогло краще зрозуміти загальну архітектуру даних. Діаграма структури бази даних наведена у додатку Б.1.

На четвертому етапі було реалізовано кодування бази даних за допомогою ORM Prisma. Це спростило взаємодію з базою даних і забезпечило типізацію даних для backend-частини. Схема бази даних наведена у додатку Б.2.

На п’ятому етапі було створено UML-діаграму прецедентів для серверної частини системи. Вона визначила основні сценарії використання API клієнтами та основні функції системи. UML-діаграму наведено у додатку Б.3.

На шостому етапі, під час розробки API, було обрано реалізацію REST API з використанням Swagger для документування. Це забезпечило зручний інтерфейс для тестування та ознайомлення з ендпоїнтами API, дозволило автоматично генерувати документацію та значно полегшило взаємодію з клієнтами системи. Для цього було використано NestJS з мовою програмування TypeScript. Усі функції взаємодії з БД інтегрувалися через Prisma.

На сьомому етапі було створено специфікацію розробленого REST API з використанням Swagger, яка детально описує доступні ендпоїнти, методи, параметри запитів і відповіді. Логіку взаємодії серверної частини з БД наведено у додатку Д. Логіку взаємодії серверної частини з клієнтами наведено у додатку Д. Специфікацію розробленого REST API наведено у додатку Е.

На восьмому етапі було проведено тестування розробленого програмного коду серверної частини системи, включно з API та функціями роботи з БД, для перевірки їхньої працездатності та відповідності вимогам.

На дев’ятому етапі весь програмний код було завантажено в GitHub репозиторій у гілку main для забезпечення централізованого зберігання та доступу до нього.

На десятому етапі було створено відеозапис демонстрації перевірки функціональності серверної частини системи. Тривалість запису склала 7-9 хвилин, а відео завантажено на YouTube з українськими субтитрами.

На одинадцятому етапі було створено хронологічний опис демонстрації відео (хвилина:секунда) і додано цей опис до опису відео на YouTube.

На дванадцятому етапі було створено звіт до лабораторної роботи, у якому вказано посилання на відеозапис на YouTube та описано основні етапи виконання завдання.

На тринадцятому етапі звіт було експортовано у формат PDF і завантажено на платформу dl.nure.ua, а також у текстовий файл з кодуванням UTF-8, який було записано до GitHub репозиторію.

У серверній частині було реалізовано наступне:

Аунтефікація в системі через JWT токени

Шифрування паролів на бекенд частині

Доступи до функцій системи відповідно до ролі

Методи Get, Post, Put, Delete для усіх таблиць

Логіку оренди та її завершення зі зміною стану самоката та його батареї, платежом при закінченні поїздки.

Отримання персоналізованої інформації для користувачів, такої як:

- усі поїздки
- усі платежі
- найближчі вільні самокати
- вільні самокати

Отримання персоналізованої інформації для адміністраторів, такої як:

- інформація по конкретному самокату
- середній час використання самокату
- усі оренди самокату
- усі користувачі та персонал системи
- усі оренди та платежі користувача
- усі самокати та їх стан
- найприбутковіші зони та їх дохід

Отримання персоналізованої інформації для технічного персоналу, такої як:

- інформація та стан самокату
- інформація про батареї
- інформація про середню швидкість
- інформація про енергозатрати

Резервне копіювання користувацьких даних

Середній час оренди

кількість поїздок на одиницю транспорту за одиницю часу

Розподіл активності за годинами/днями

Відстеження простоїв транспорту (час між орендами)

Найпопулярніші зони старту/фінішу

Розрахунок ефективності використання батарей

ВИСНОВОК

У ході виконання роботи було створено програмну систему, яка включає серверну частину з функціональним REST API, інтегрованим з базою даних через ORM Prisma. Було виконано повний цикл розробки: від проєктування архітектури системи та створення діаграм (ER, UML) до програмної реалізації та тестування. Завдяки цьому вдалося побудувати ефективну та зрозумілу систему для взаємодії

клієнтів із сервером.

Для розробки серверної частини використовувалася мова програмування TypeScript у фреймворку NestJS. У ролі системи управління базами даних було використано PostgreSQL, адміністровану через PgAdmin4, а для ORM – Prisma, яка забезпечила зручну взаємодію з базою даних і строгий контроль типів. Документацію для REST API було реалізовано через Swagger, що зробило API доступним для тестування та інтеграції.

Виконання лабораторної роботи сприяло набуттю практичних навичок у створенні програмних систем, проєктуванні архітектури серверної частини та баз даних, а також реалізації REST API з використанням сучасних інструментів. Я навчився ефективно інтегрувати різні технології, такі як Docker для контейнеризації системи, Prisma для роботи з базами даних та Swagger для документування API.

Ця лабораторна робота навчила мене організовувати процес розробки від планування до завершення, включаючи тестування та підготовку звітності. Крім того, я набув досвіду використання GitHub для управління кодом і публікації результатів, а також створення відеодемонстрацій, що допомогло розвинути як технічні, так і презентаційні навички.

Відеозапис доповіді на YouTube: https://youtu.be/Pg0JZRWXuoc

ДОДАТОК А
Відеозапис

Відеозапис презентації результатів лабораторної роботи:
https://youtu.be/Pg0JZRWXuoc
Хронологічний опис роботи:

00:00 - початок
00:18 – отримання усіх оренд
00:30 – оренда транспорту
01:18 – перевірка статусу оренди
01:35 – завершення оренди
03:03 – середня тривалість оренди
03:16 – активніть по годинам
03:37 – активніть по дням
04:16 – часи простою траспорта
05:02 – популярні зони початку оренди
05:48 – популярні зони завершення оренди
06:06 – прибуток з зони
06:44 – інформація та статистика щодо транспортного засобу
07:14 – дохід з транспортного засобу
07:30 – усі вільні самокати
07:47 – середній час використання самокату
08:03 – кількість оренд на самокат
09:02 – пройдена дистанція самоката
09:07 – найближчі самокати до користувача
09:50 – середня кількість поїздок на одиниці транспорту
10:15 – використана енергія батареєю
10:42 – ефективність викиростання батареї
11:07 – інщі функції системи(логін, розподілення ролей, шифрування)

ДОДАТОК Б
Графічні матеріали

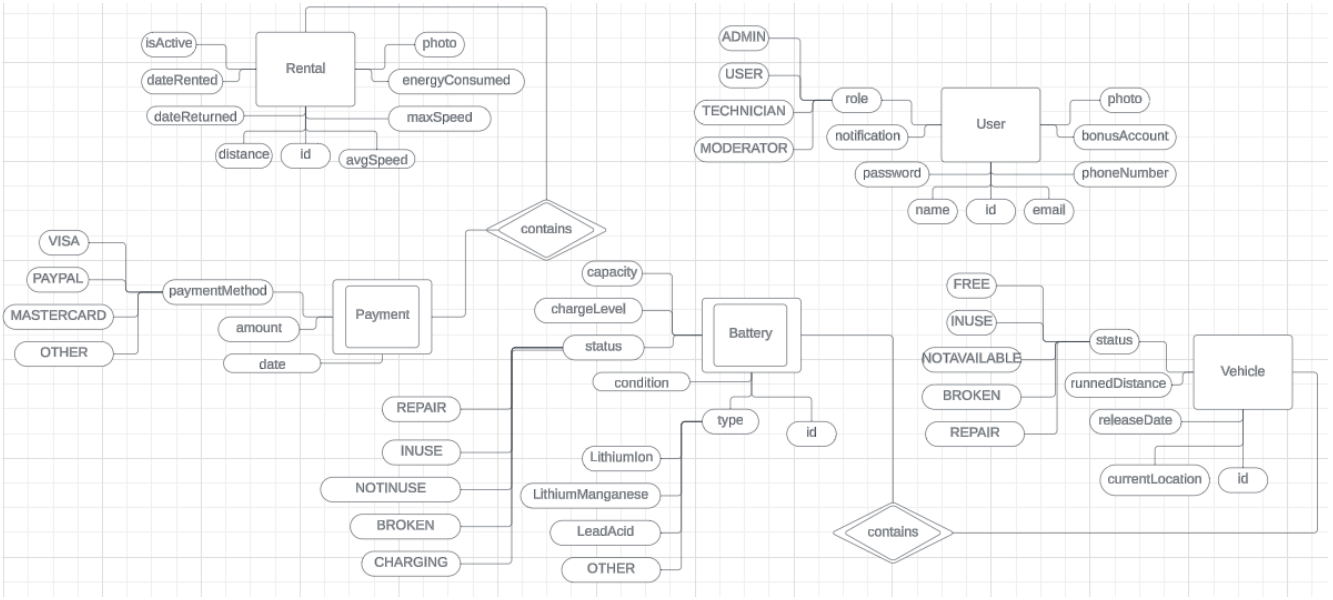

Рисунок Б.1 — ER-діаграма даних (нотація Чена)

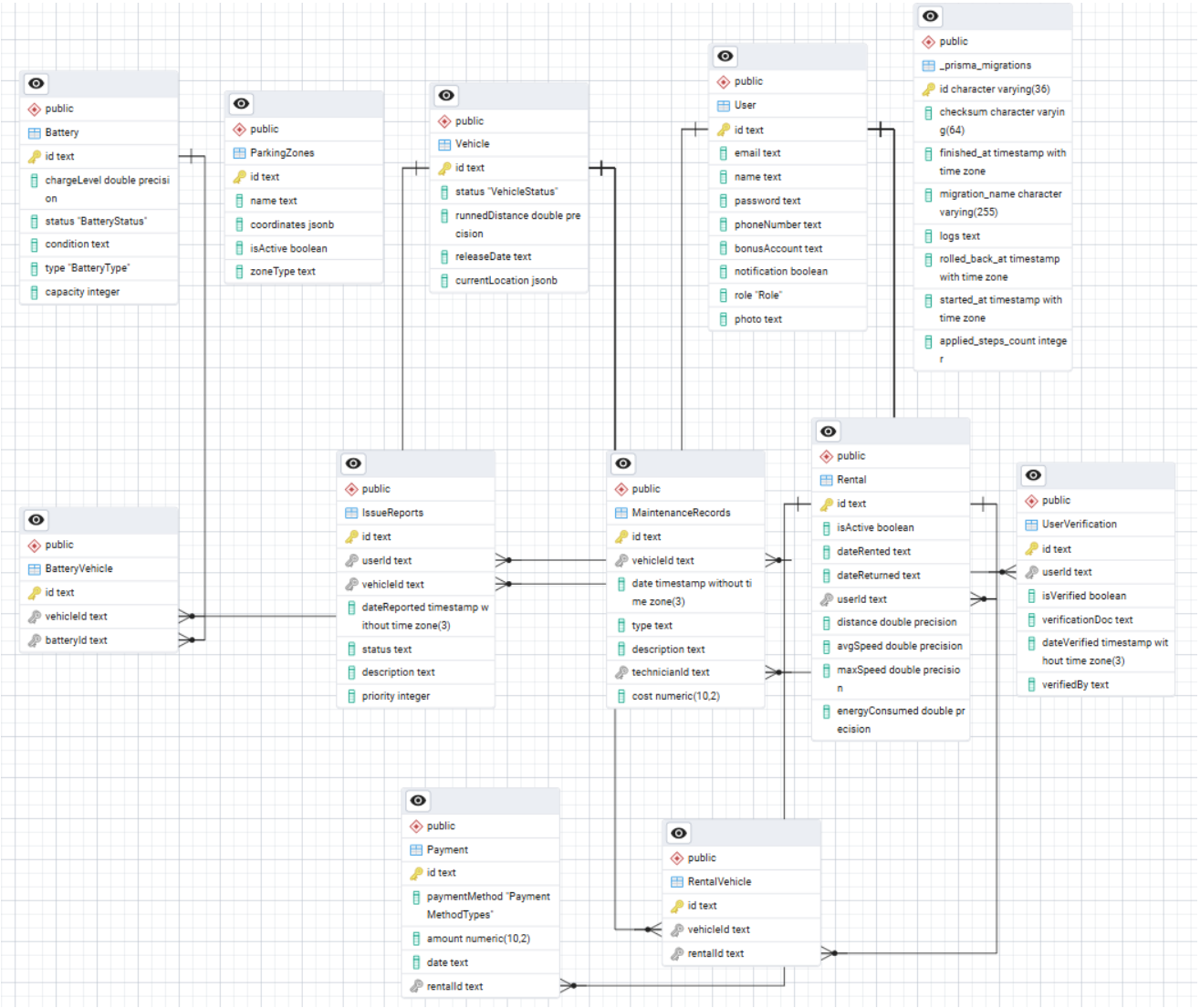

Рисунок Б.2 — Структура бази даних

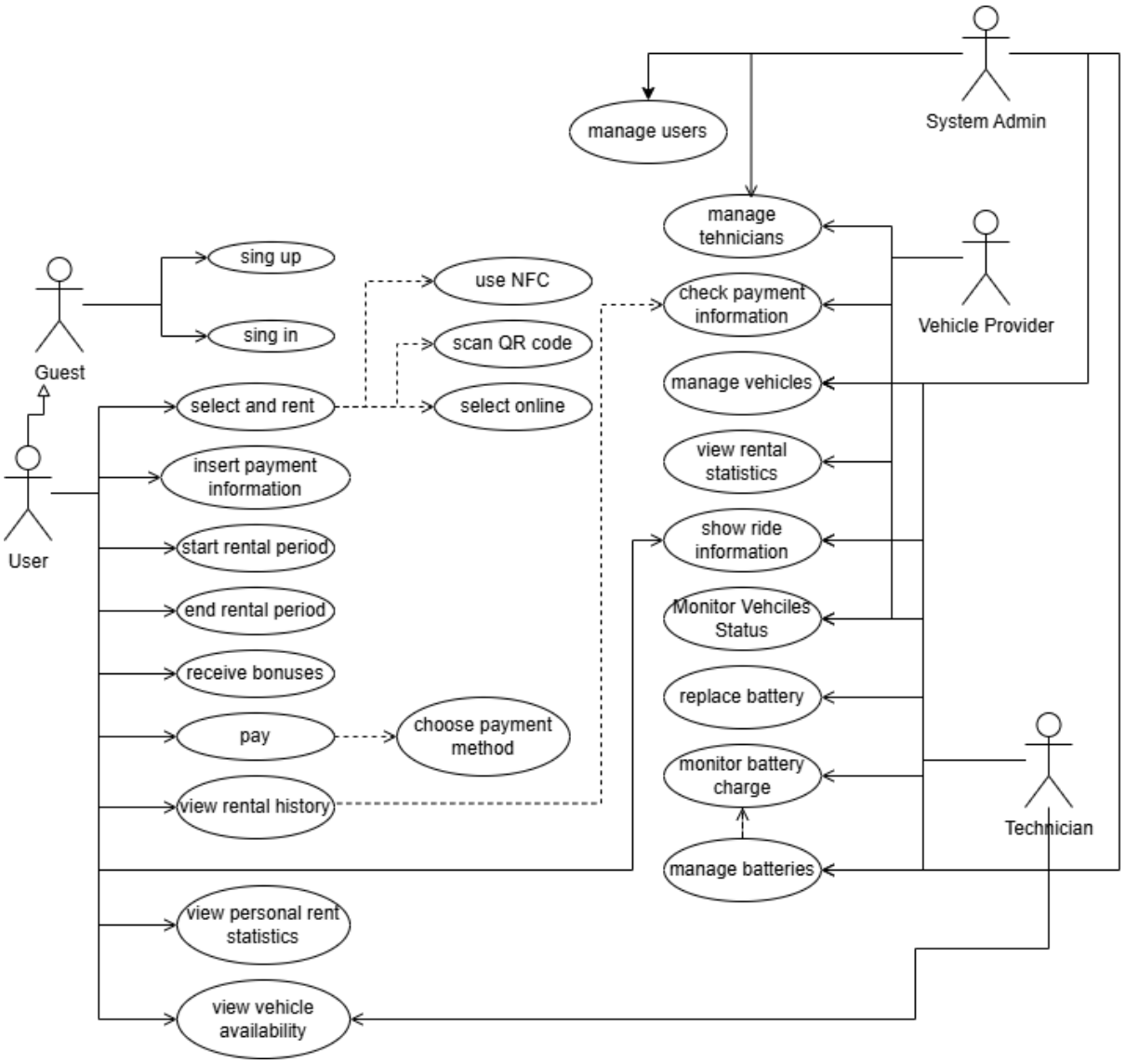

Рисунок Б.3 — UML-діаграма прецедентів

ДОДАТОК В

Фрагменти коду серверу

В.1 Математична обробка прикладних даних

GitHub репозиторій: https://github.com/NureVeriasovVladyslav/apz-pzpi-22-2-veriasov-vladyslav/blob/main/Lab2/apz-pzpi-22-2-veriasov-vladyslav-lab2/apz-pzpi-22-2-veriasov-vladyslav-lab2-code/src/vehicle/vehicle.service.ts

Та GitHub репозиторій: https://github.com/NureVeriasovVladyslav/apz-pzpi-22-2-veriasov-vladyslav/blob/main/Lab2/apz-pzpi-22-2-veriasov-vladyslav-lab2/apz-pzpi-22-2-veriasov-vladyslav-lab2-code/src/rental/rental.service.ts

Лістинг 1 – Функції з елементами математичної обробки інформації з IoT (фрагмент сервісу для таблиці Vehicle)

```
async calculateTotalDistance(vehicleId: string): Promise<number> {

        // Отримуємо всі оренди для вказаного транспортного засобу
        const rentals = await this.prisma.rentalVehicle.findMany({
            where: { vehicleId },
            include: { rental: true },
        });

        // Обчислюємо загальну дистанцію
        const totalDistance = rentals.reduce(
            (total, rentalVehicle) => total +
rentalVehicle.rental.distance,
            0,
        );

        // Оновлюємо поле `runnedDistance` у транспортному засобі
        await this.prisma.vehicle.update({
            where: { id: vehicleId },
            data: { runnedDistance: totalDistance },
        });

        // Повертаємо обчислену загальну дистанцію
        return totalDistance;
    }

    async getVehicleInfo(vehicleId: string): Promise<VehicleDto> {
        const result = await this.prisma.vehicle.findUnique({
            where: { id: vehicleId },
        });
        return result;
    }
```

```
    async calculateAverageSpeed(vehicleId: string): Promise<number>
{
        const rentals = await this.prisma.rentalVehicle.findMany({
            where: { vehicleId },
            include: { rental: true },
        });

        const totalSpeed = rentals.reduce((sum, rentalVehicle) =>
sum + rentalVehicle.rental.avgSpeed, 0);
        return rentals.length > 0 ? totalSpeed / rentals.length : 0;
    }

    async findMostEfficientVehicle(): Promise<{ vehicleId: string,
efficiency: number }> {
        const vehicles = await this.prisma.vehicle.findMany({
            include: { rentalVehicle: { include: { rental: true } }
},
        });

        const efficiencyData = vehicles.map(vehicle => {
            const totalDistance = vehicle.rentalVehicle.reduce((sum,
rv) => sum + rv.rental.distance, 0);
            const totalEnergy = vehicle.rentalVehicle.reduce((sum,
rv) => sum + rv.rental.energyConsumed, 0);
            const efficiency = totalDistance > 0 ? totalEnergy /
totalDistance : Infinity;
            return { vehicleId: vehicle.id, efficiency };
        });

        return efficiencyData.reduce((best, current) =>
(current.efficiency < best.efficiency ? current : best));
    }

    async findAllVehicleWithStatus(status: VehicleStatus):
Promise<VehicleDto[]> {
        const result = await this.prisma.vehicle.findMany({
            where: { status: status },
        });
        return result;
    }
```

Лістинг 2 – Функції з елементами математичної обробки з IoT, а саме робота з координатами та відстанями (фрагмент сервісу для таблиці Rental)

```
// Метод для оновлення місцезнаходження транспортного засобу
    async updateVehicleLocation(vehicleId: string, lat: number, lng:
number) {
        try {
            // Переконайтемось, що транспорт існує
            const vehicle = await this.prisma.vehicle.findUnique({
```

```
                where: { id: vehicleId },
            });

            if (!vehicle) {
                throw new Error('Vehicle not found');
            }

            // Оновлюємо місцезнаходження з використанням PostGIS
            await this.prisma.$executeRaw`
        UPDATE "Vehicle"
        SET currentLocation = ST_SetSRID(ST_MakePoint(${lng},
${lat}), 4326)::geography
        WHERE id = ${vehicleId}
      `;

            return { success: true, vehicleId, location: { lat, lng
} };
        } catch (error) {
            console.error('Failed to update vehicle location:',
error);
            throw error;
        }
    }

    // Метод для пошуку найближчих транспортних засобів
    async findNearbyVehicles(lat: number, lng: number,
radiusInMeters: number = 500) {
        return this.prisma.$queryRaw`
      SELECT
        v.id,
        v.status,
        ST_Distance(
          v.currentLocation::geography,
          ST_SetSRID(ST_MakePoint(${lng}, ${lat}), 4326)::geography
        ) as distance
      FROM "Vehicle" v
      WHERE
        v.status = 'FREE'
        AND ST_DWithin(
          v.currentLocation::geography,
          ST_SetSRID(ST_MakePoint(${lng}, ${lat}), 4326)::geography,
          ${radiusInMeters}
        )
      ORDER BY distance
      LIMIT 10
    `;
    }

    async calculateAverageRentalDuration(): Promise<number> {
        // Отримуємо всі завершені оренди (де dateReturned не null)
        const rentals = await this.prisma.rental.findMany({
            where: {
                NOT: {
```

```
                    dateReturned: "",
                }
            },
            select: {
                dateRented: true,
                dateReturned: true,
            },
        });

        if (rentals.length === 0) return 0;

        // Підрахунок загального часу у годинах
        const totalDuration = rentals.reduce((sum, rental) => {
            const start = new Date(rental.dateRented).getTime();
            const end = new Date(rental.dateReturned).getTime();
            const durationHours = (end - start) / 3600000; //
мілісекунди → години
            return sum + durationHours;
        }, 0);

        // Середній час
        return Number((totalDuration / rentals.length).toFixed(2));
    }

    async getHourlyActivityDistribution(): Promise<{ [hour: string]:
number }> {
        const rentals = await this.prisma.rental.findMany({
            select: { dateRented: true },
        });

        // Об'єкт: { "0": 5, "1": 2, ..., "23": 7 }
        const distribution: { [hour: string]: number } = {};

        for (let i = 0; i < 24; i++) {
            distribution[i.toString()] = 0;
        }

        for (const rental of rentals) {
            if (!rental.dateRented) continue;
            const hour = new Date(rental.dateRented).getHours();
            distribution[hour.toString()]++;
        }

        return distribution;
    }

    async getDailyActivityDistribution(): Promise<{ [date: string]:
number }> {
        const rentals = await this.prisma.rental.findMany({
            select: { dateRented: true },
        });

        // Об'єкт: { "2025-06-01": 3, "2025-06-02": 7, ... }
```

```
        const distribution: { [date: string]: number } = {};

        for (const rental of rentals) {
            if (!rental.dateRented) continue;
            const date = new
Date(rental.dateRented).toISOString().slice(0, 10); // YYYY-MM-DD
            if (!distribution[date]) distribution[date] = 0;
            distribution[date]++;
        }

        return distribution;
    }
    async getIdleTimesForVehicle(vehicleId: string): Promise<Array<{
idleStart: string, idleEnd: string, idleHours: number }>> {
        // 1. Отримати всі оренди для цього транспорту, відсортувати
за dateRented
        const rentalVehicles = await
this.prisma.rentalVehicle.findMany({
            where: { vehicleId },
            include: {
                rental: {
                    select: {
                        dateRented: true,
                        dateReturned: true,
                    }
                }
            },
            orderBy: {
                rental: {
                    dateRented: 'asc'
                }
            }
        });

        // 2. Відфільтрувати тільки завершені оренди (де є
dateReturned)
        const completedRentals = rentalVehicles
            .map(rv => rv.rental)
            .filter(r => r.dateReturned && r.dateRented);

        // 3. Розрахувати простої між орендами
        const idleTimes: Array<{ idleStart: string, idleEnd: string,
idleHours: number }> = [];

        for (let i = 0; i < completedRentals.length - 1; i++) {
            const prev = completedRentals[i];
            const next = completedRentals[i + 1];

            // Простій: від повернення попередньої до старту
наступної
            const idleStart = prev.dateReturned;
            const idleEnd = next.dateRented;
```

```
            const start = new Date(idleStart).getTime();
            const end = new Date(idleEnd).getTime();
            const idleHours = (end - start) / 3600000;

            // Враховуємо тільки позитивний простій (можливо, дати
некоректні)
            if (idleHours >= 0) {
                idleTimes.push({
                    idleStart,
                    idleEnd,
                    idleHours: Number(idleHours.toFixed(2))
                });
            }
        }

        return idleTimes;
    }
    async getAllVehiclesIdleTimes(): Promise<Record<string, Array<{
idleStart: string, idleEnd: string, idleHours: number }>>> {
        const vehicles = await this.prisma.vehicle.findMany({
            select: { id: true }
        });

        const result: Record<string, Array<{ idleStart: string,
idleEnd: string, idleHours: number }>> = {};

        for (const vehicle of vehicles) {
            result[vehicle.id] = await
this.getIdleTimesForVehicle(vehicle.id);
        }

        return result;
    }

    /////============================

    // Функція для визначення ключа квадрата (grid cell)
    private getGridCellKey(lat: number, lng: number, cellSize =
0.002): string {
        const latIndex = Math.floor(lat / cellSize);
        const lngIndex = Math.floor(lng / cellSize);
        return `${latIndex}_${lngIndex}`;
    }

    // Найпопулярніші зони старту
    async getPopularStartZones(cellSize = 0.002, topN = 10) {
        console.log("zoneRevenue")
        const rentals = await this.prisma.rental.findMany({
            select: { startLocation: true }, // currentLocation =
стартова точка
        });
        console.log("rentals", rentals)
        const zoneCounts: Record<string, number> = {};
```

```
        for (const rental of rentals) {
            const loc = rental.startLocation as any;
            if (loc && loc.latitude !== undefined && loc.longitude
!== undefined) {
                const key = this.getGridCellKey(loc.latitude,
loc.longitude, cellSize);
                zoneCounts[key] = (zoneCounts[key] || 0) + 1;
            }
        }
        console.log("zoneCounts", zoneCounts)
        // Топ-N зон
        return Object.entries(zoneCounts)
            .sort((a, b) => b[1] - a[1])
            .slice(0, topN)
            .map(([cellKey, count]) => ({
                cellKey,
                count,
                // Центр квадрата для візуалізації
                centerLat: Number(cellKey.split('_')[0]) * cellSize
+ cellSize / 2,
                centerLng: Number(cellKey.split('_')[1]) * cellSize
+ cellSize / 2,
            }));
    }

    // Найпопулярніші зони фінішу (якщо є endLocation)
    async getPopularEndZones(cellSize = 0.002, topN = 10) {
        const rentals = await this.prisma.rental.findMany({
            select: { endLocation: true }, // Додай це поле у
Rental, якщо його ще немає!
        });

        const zoneCounts: Record<string, number> = {};

        for (const rental of rentals) {
            const loc = rental.endLocation as any;
            if (loc && loc.latitude !== undefined && loc.longitude
!== undefined) {
                const key = this.getGridCellKey(loc.latitude,
loc.longitude, cellSize);
                zoneCounts[key] = (zoneCounts[key] || 0) + 1;
            }
        }

        return Object.entries(zoneCounts)
            .sort((a, b) => b[1] - a[1])
            .slice(0, topN)
            .map(([cellKey, count]) => ({
                cellKey,
                count,
                centerLat: Number(cellKey.split('_')[0]) * cellSize
+ cellSize / 2,
```

```
                centerLng: Number(cellKey.split('_')[1]) * cellSize
+ cellSize / 2,
            }));
    }

    // Найприбутковіші зони старту
    async getMostProfitableStartZones(cellSize = 0.002, topN = 10) {
        console.log("zoneRevenue")
        // Витягуємо всі поїздки з платежами
        const rentals = await this.prisma.rental.findMany({
            select: {
                id: true,
                payment: { select: { amount: true } }, // payment
має бути зв'язком rentalId → Payment
                startLocation: true,
            },
        });

        const zoneRevenue: Record<string, number> = {};

        for (const rental of rentals) {
            const loc = rental.startLocation as any;
            const amount = rental.payment?.amount ?
Number(rental.payment.amount) : 0;
            if (loc && loc.latitude !== undefined && loc.longitude
!== undefined) {
                const key = this.getGridCellKey(loc.latitude,
loc.longitude, cellSize);
                zoneRevenue[key] = (zoneRevenue[key] || 0) + amount;
            }
        }
        console.log("zoneRevenue", zoneRevenue)

        return Object.entries(zoneRevenue)
            .sort((a, b) => b[1] - a[1])
            .slice(0, topN)
            .map(([cellKey, totalRevenue]) => ({
                cellKey,
                totalRevenue,
                centerLat: Number(cellKey.split('_')[0]) * cellSize
+ cellSize / 2,
                centerLng: Number(cellKey.split('_')[1]) * cellSize
+ cellSize / 2,
            }));
    }
```

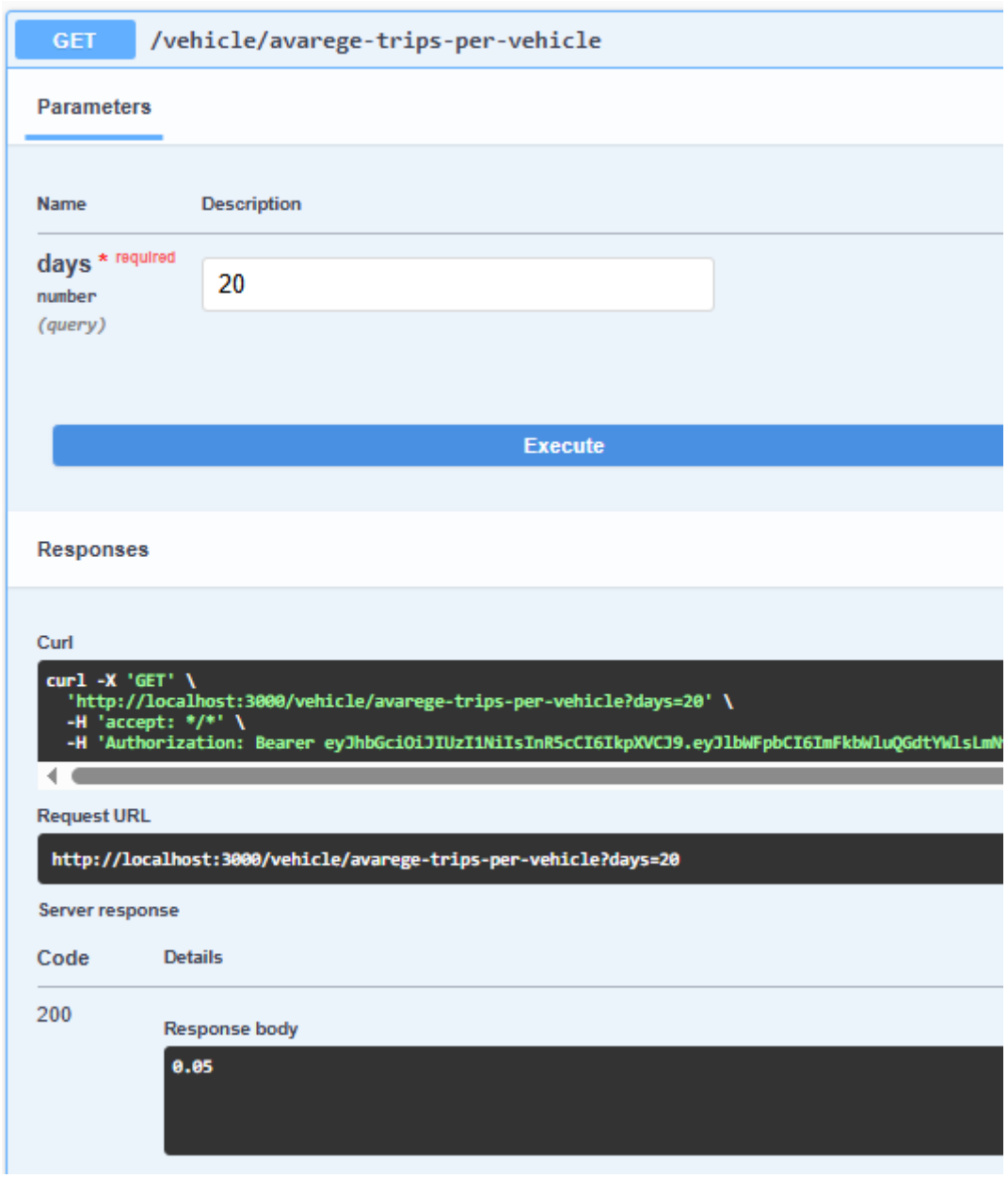

Рисунок В.1 – Приклад 1 (для числових даних)

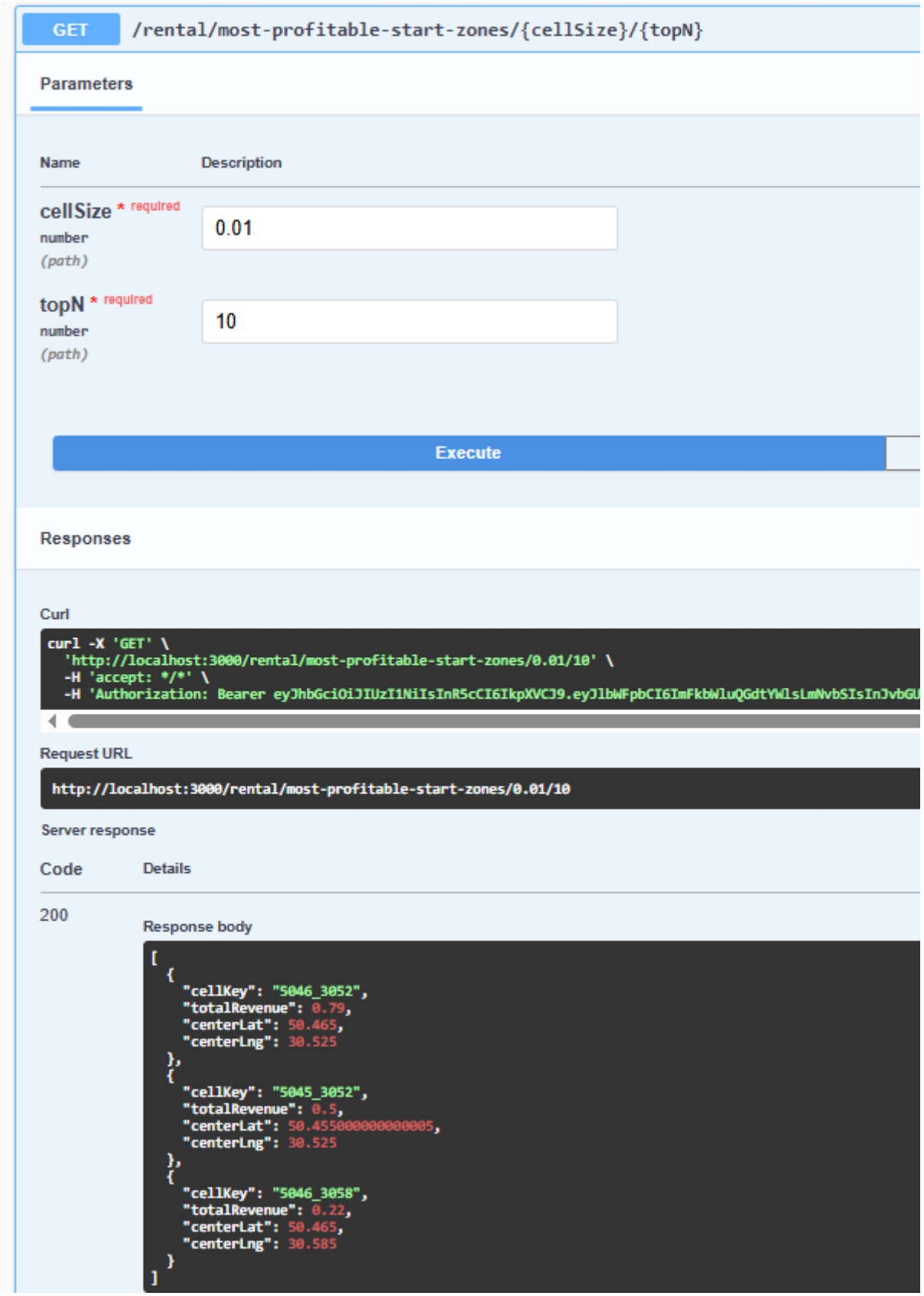

Рисунок В.2 – Приклад 2 (для координат)

B.2 Адміністрування бізнес-логіки системи

GitHub репозиторій: https://github.com/NureVeriasovVladyslav/apz-pzpi-22-2-veriasov-vladyslav/blob/main/Lab2/apz-pzpi-22-2-veriasov-vladyslav-lab2/apz-pzpi-22-2-veriasov-vladyslav-lab2-code/src/rental/rental.controller.ts

Лістинг 3 - Контроллер сторінки для управління орендами

```
import { Body, Controller, Delete, Get, HttpCode, HttpException,
HttpStatus, InternalServerErrorException, NotFoundException, Param,
Post, Put, Query, UseGuards } from '@nestjs/common';
import { ApiBearerAuth, ApiResponse, ApiTags } from
'@nestjs/swagger';
import { RentalService } from './rental.service';
import { RentalDto } from './dtos/rental.dto';
import { UpdateRentalDto } from './dtos/update-rental.dto';
import { PaymentMethodTypes } from '@prisma/client';
import { CreateRentalDto } from './dtos/create-rental.dto';

import { RentalVehicleDto } from 'src/rental-vehicle/dtos/rental-
vehicle.dto';
import { PaymentDto } from 'src/user/dtos/userPlus.dto';
import { RentalNotFoundException } from 'src/exceptions/user-
exceptions';
import { Roles } from 'src/auth/roles.decorator';
import { Role } from '@prisma/client';
import { RoleGuard } from 'src/auth/roles.guard';
import { JwtAuthGuard } from 'src/auth/jwt-auth.guard';
import { OwnershipGuard } from 'src/auth/ownership.guard';
import { RentalFullDto } from './dtos/rental-full.dto';
import { CreateRentalFullDto } from './dtos/create-rental-full.dto';
import { PaymentRentalVehicleDto } from
'./dtos/paymentRentalVehicle.dto';

@ApiTags('rental')
@Controller('rental')
export class RentalController {
    constructor(private readonly rentalService: RentalService) { }

    @Get()
    @UseGuards(JwtAuthGuard, RoleGuard)
    @Roles(Role.MODERATOR, Role.ADMIN)
    @ApiBearerAuth()
    @HttpCode(200)
    @ApiResponse({ status: 200, description: 'List of all rentals
returned successfully.' })
    @ApiResponse({ status: 500, description: 'Internal server
error.' })
    public async findAllRental() {
        const result = await this.rentalService.findAllRental();
```

```
        return result;
    }

    @Post(':vehicleId')
    @UseGuards(JwtAuthGuard, RoleGuard)
    @Roles(Role.ADMIN, Role.USER)
    @ApiBearerAuth()
    @HttpCode(201)
    @ApiResponse({ status: 201, description: 'Rental created
successfully.' })
    @ApiResponse({ status: 400, description: 'Invalid input data.'
})
    @ApiResponse({ status: 500, description: 'Internal server
error.' })
    public async createRental(@Body() rental: RentalDto) {
        const result = await
this.rentalService.createRental(rental);
        return result;
    }

    @Put(':id')
    @UseGuards(JwtAuthGuard, RoleGuard)
    @Roles(Role.ADMIN, Role.USER)
    @ApiBearerAuth()
    @HttpCode(200)
    @ApiResponse({ status: 200, description: 'Rental updated
successfully.' })
    @ApiResponse({ status: 404, description: 'Rental not found.' })
    @ApiResponse({ status: 500, description: 'Internal server
error.' })
    public async updateRental(@Body() rental: UpdateRentalDto,
@Param('id') id: string) {
        const result = await this.rentalService.updateRental(rental,
id);
        return result;
    }

    @Delete(':id')
    @UseGuards(JwtAuthGuard, RoleGuard)
    @Roles(Role.ADMIN, Role.USER)
    @ApiBearerAuth()
    @HttpCode(200)
    @ApiResponse({ status: 200, description: 'Rental deleted
successfully.' })
    @ApiResponse({ status: 404, description: 'Rental not found.' })
    @ApiResponse({ status: 500, description: 'Internal server
error.' })
    public async deleteRental(@Param('id') id: string) {
        const result = await this.rentalService.deleteRental(id);
        return result;
    }

    @Get('total')
```

```
    @UseGuards(JwtAuthGuard, RoleGuard)
    @Roles(Role.ADMIN)
    @ApiBearerAuth()
    @HttpCode(200)
    async getTotalProfit(
        @Query('startDate') startDate: string,
        @Query('endDate') endDate: string
    ): Promise<{ totalProfit: number }> {
        try {
            const totalProfit = await
this.rentalService.calculateTotalProfit(startDate, endDate);
            return { totalProfit };
        } catch (error) {
            throw new Error(`Error while calculating total profit:
${error.message}`);
        }
    }


    @Get(':userId/rentals-with-vehicles')
    @UseGuards(JwtAuthGuard, RoleGuard)
    @Roles(Role.ADMIN, Role.MODERATOR)
    @ApiBearerAuth()
    @ApiResponse({ status: 200, description: 'List of rentals with
vehicles.' })
    @ApiResponse({ status: 404, description: 'User not found.' })
    public async getUserRentalsWithVehicles(@Param('userId') userId:
string) {
        const result = await
this.rentalService.getUserRentalsWithVehicles(userId);
        return result;
    }

    @Get(':id')
    @UseGuards(JwtAuthGuard, RoleGuard)
    @Roles(Role.ADMIN, Role.MODERATOR)
    @ApiBearerAuth()
    @HttpCode(200)
    async getRentalById(@Param('id') id: string) {
        try {
            const rental = await
this.rentalService.getRentalById(id);
            return rental;
        } catch (error) {
            if (error instanceof RentalNotFoundException) {
                throw new NotFoundException(error.message);  // HTTP
404
            }
            throw new InternalServerErrorException('Internal Server
Error');
        }
    }
```

```
    @Post('full/start/:data')
    @UseGuards(JwtAuthGuard, RoleGuard)
    @Roles(Role.ADMIN, Role.USER)
    @ApiBearerAuth()
    @HttpCode(201)
    @ApiResponse({ status: 201, description: 'Rental created
successfully.' })
    @ApiResponse({ status: 400, description: 'Invalid input data.'
})
    @ApiResponse({ status: 500, description: 'Internal server
error.' })
    public async createRentalFull(@Body() rental: RentalFullDto) {
        const result = await
this.rentalService.createRentalFull(rental);
        return result;
    }

    @Post('full/end/:payment')
    @UseGuards(JwtAuthGuard, RoleGuard)
    @Roles(Role.ADMIN, Role.USER)
    @ApiBearerAuth()
    @HttpCode(201)
    @ApiResponse({ status: 201, description: 'Rental created
successfully.' })
    @ApiResponse({ status: 400, description: 'Invalid input data.'
})
    @ApiResponse({ status: 500, description: 'Internal server
error.' })
    public async endRentalFull(@Body() paymentRentalVehicle:
PaymentRentalVehicleDto) {
        const result = await
this.rentalService.endRentalFull(paymentRentalVehicle);
        return result;
    }

    @Get('all/rental/average-duration')
    @UseGuards(JwtAuthGuard, RoleGuard)
    @Roles(Role.MODERATOR, Role.ADMIN, Role.TECHNICIAN)
    @ApiBearerAuth()
    @HttpCode(200)
    @ApiResponse({ status: 200, description: 'Average rental
duration returned successfully.' })
    @ApiResponse({ status: 500, description: 'Internal server
error.' })
    public async calculateAverageRentalDuration(): Promise<number> {
        return await
this.rentalService.calculateAverageRentalDuration();
    }

    @Get('activity/hourly')
    @UseGuards(JwtAuthGuard, RoleGuard)
    @Roles(Role.MODERATOR, Role.ADMIN, Role.TECHNICIAN)
    @ApiBearerAuth()
```

```
    @ApiResponse({ status: 200, description: 'Hourly activity
distribution returned successfully.' })
    @ApiResponse({ status: 500, description: 'Internal server
error.' })
    public async getHourlyActivityDistribution() {
        return await
this.rentalService.getHourlyActivityDistribution();
    }

    @Get('activity/daily')
    @UseGuards(JwtAuthGuard, RoleGuard)
    @Roles(Role.MODERATOR, Role.ADMIN, Role.TECHNICIAN)
    @ApiBearerAuth()
    @ApiResponse({ status: 200, description: 'Daily activity
distribution returned successfully.' })
    @ApiResponse({ status: 500, description: 'Internal server
error.' })
    public async getDailyActivityDistribution() {
        return await
this.rentalService.getDailyActivityDistribution();
    }

    @Get('idle-times/:vehicleId')
    @UseGuards(JwtAuthGuard, RoleGuard)
    @Roles(Role.MODERATOR, Role.ADMIN, Role.TECHNICIAN)
    @ApiBearerAuth()
    @ApiResponse({ status: 200, description: 'Idle times for vehicle
returned successfully.' })
    @ApiResponse({ status: 500, description: 'Internal server
error.' })
    public async getIdleTimesForVehicle(@Param('vehicleId')
vehicleId: string) {
        return await
this.rentalService.getIdleTimesForVehicle(vehicleId);
    }

    @Get('idle-times/all')
    @UseGuards(JwtAuthGuard, RoleGuard)
    @Roles(Role.MODERATOR, Role.ADMIN, Role.TECHNICIAN)
    @ApiBearerAuth()
    @ApiResponse({ status: 200, description: 'Idle times for vehicle
returned successfully.' })
    @ApiResponse({ status: 500, description: 'Internal server
error.' })
    public async getAllVehiclesIdleTimes() {
        return await this.rentalService.getAllVehiclesIdleTimes();
    }

    @Get('popular-start-zones/:cellSize/:topN')
    @UseGuards(JwtAuthGuard, RoleGuard)
    @Roles(Role.ADMIN, Role.MODERATOR, Role.TECHNICIAN)
    @ApiBearerAuth()
    @ApiResponse({ status: 200, description: 'Popular start zones
```

```
returned.' })
    @ApiResponse({ status: 500, description: 'Internal server
error.' })
    public async getPopularStartZones(@Param('cellSize') cellSize:
number, @Param('topN') topN: number) {
        // @Query('cellSize') cellSize?: number,
        // @Query('topN') topN?: number
        console.log("zoneRevenue")
        return this.rentalService.getPopularStartZones(cellSize,
topN);
    }

    @Get('popular-end-zones/:cellSize/:topN')
    @UseGuards(JwtAuthGuard, RoleGuard)
    @Roles(Role.ADMIN, Role.MODERATOR, Role.TECHNICIAN)
    @ApiBearerAuth()
    @ApiResponse({ status: 200, description: 'Popular end zones
returned.' })
    @ApiResponse({ status: 500, description: 'Internal server
error.' })
    async getPopularEndZones(@Param('cellSize') cellSize: number,
@Param('topN') topN: number
        // @Query('cellSize') cellSize?: number,
        // @Query('topN') topN?: number
    ) {
        return this.rentalService.getPopularEndZones(cellSize,
topN);
    }

    @Get('most-profitable-start-zones/:cellSize/:topN')
    @UseGuards(JwtAuthGuard, RoleGuard)
    @Roles(Role.ADMIN, Role.MODERATOR, Role.TECHNICIAN)
    @ApiBearerAuth()
    @ApiResponse({ status: 200, description: 'Most profitable start
zones returned.' })
    @ApiResponse({ status: 500, description: 'Internal server
error.' })
    async getMostProfitableStartZones(
        @Param('cellSize') cellSize: number, @Param('topN') topN:
number
    ) {
        return
this.rentalService.getMostProfitableStartZones(cellSize, topN);
    }
}
```

В.3 Резервне копіювання користувацьких даних

GitHub репозиторій: https://github.com/NureVeriasovVladyslav/apz-pzpi-22-2-veriasov-vladyslav/blob/main/Lab2/apz-pzpi-22-2-veriasov-vladyslav-lab2/apz-pzpi-22-2-veriasov-vladyslav-lab2-code/backup/backup.sh

Лістинг 4 - Запуск сервісу автоматичного збереження бекапів бази даних

```
#!/bin/bash

# Змінні
TIMESTAMP=$(date +"%Y-%m-%d_%H-%M")
BACKUP_FILE="/backups/e-transport-$TIMESTAMP.backup"

# Команда pg_dump
pg_dump -h postgres -U postgres -d e-transport -F c -f
"$BACKUP_FILE"

# Вивід у лог
echo "Backup created at $BACKUP_FILE"
```

Лістинг 2 - Докерфайл

```
FROM postgres:15

RUN apt-get update && apt-get install -y cron

# Копіюємо скрипт та план
COPY backup.sh /usr/local/bin/backup.sh
COPY crontab /etc/cron.d/backup-cron

# Дозволи
RUN chmod +x /usr/local/bin/backup.sh \
    && chmod 0644 /etc/cron.d/backup-cron

# Додаємо cron job
RUN crontab /etc/cron.d/backup-cron

# Створюємо директорію для бекапів
VOLUME /backups
RUN mkdir -p /backups

# Запускаємо cron
CMD ["cron", "-f"]
```

ДОДАТОК Г

Логіка взаємодії серверної частини з БД (приклад розробленого сервісу для таблиці Rental)

```
import { BadRequestException, Injectable, NotFoundException } from
'@nestjs/common';
import { PrismaService } from 'src/prisma/prisma.service';
import { RentalDto } from './dtos/rental.dto';
import { CreateRentalDto } from './dtos/create-rental.dto';
import { UpdateRentalDto } from './dtos/update-rental.dto';
import { PaymentDto } from 'src/user/dtos/userPlus.dto';
import { RentalVehicleDto } from 'src/rental-vehicle/dtos/rental-
vehicle.dto';
import { VehicleDto } from 'src/vehicle/dtos/vehicle.dto';
import { CreateRentalVehicleDto } from 'src/rental-
vehicle/dtos/create-rental-vehicle.dto';
import { CreatePaymentDto } from 'src/payment/dtos/create-
payment.dto';
import { RentalNotFoundException } from 'src/exceptions/user-
exceptions';
import { Rental } from '@prisma/client';
import { strict } from 'assert';
import { RentalFullDto } from './dtos/rental-full.dto';
import { CreateRentalFullDto } from './dtos/create-rental-full.dto';
import { PaymentRentalVehicleDto } from
'./dtos/paymentRentalVehicle.dto';

import booleanPointInPolygon from '@turf/boolean-point-in-polygon';
import { point, polygon } from '@turf/helpers';

@Injectable()
export class RentalService {
    constructor(private prisma: PrismaService) { }

    async findAllRental(): Promise<RentalDto[]> {
        const result = await this.prisma.rental.findMany();
        return result;
    }

    async createRental(rental: RentalDto): Promise<CreateRentalDto>
{
        const result = await this.prisma.rental.create({ data: {
...rental } });
        return result;
    }

    async updateRental(rental: UpdateRentalDto, id: string):
Promise<CreateRentalDto> {
        const result = await this.prisma.rental.update({ data: {
...rental }, where: { id: id } });
        return result;
    }
```

```
    async deleteRental(id: string): Promise<CreateRentalDto> {
        const result = await this.prisma.rental.delete({ where: {
id: id } });
        return result;
    }

    async calculateTotalProfit(startDate: string, endDate: string):
Promise<number> {
        const payments = await this.prisma.payment.findMany({
            where: {
                date: {
                    gte: startDate,
                    lte: endDate,
                },
            },
        });

        // return payments.reduce((total, payment) => total +
parseFloat(payment.amount), 0);
        return payments.reduce((total, payment) => total +
payment.amount.toNumber(), 0);
    }

    async getUserRentalsWithVehicles(userId: string) {
        return await this.prisma.rental.findMany({
            where: { userId },
            include: {
                rentalVehicle: {
                    include: {
                        vehicle: true,
                    },
                },
            },
        });
    }

    // Приклад методу, який може генерувати виключення
    async getRentalById(rentalId: string) {
        const rental = await this.prisma.rental.findUnique({
            where: { id: rentalId },
        });

        if (!rental) {
            throw new RentalNotFoundException(`Rental with ID
${rentalId} not found.`);
        }

        return rental;
    }
    async createRentalFull(rental: RentalFullDto):
Promise<CreateRentalFullDto> {
        try {
```

```
            return await this.prisma.$transaction(async (tx) => {
                const vehicle = await
this.prisma.vehicle.findUnique({
                    where: { id: rental.vehicleId },
                });

                if (!vehicle) {
                    throw new NotFoundException('Vehicle not
found');
                }

                const resultVehicle = await
this.prisma.vehicle.update({
                    data: { ...vehicle, status: 'INUSE' }, where: {
id: rental.vehicleId }
                });

                const resultRental = await
this.prisma.rental.create({
                    data: {
                        userId: rental.userId,
                        dateRented: new Date().toISOString(),
                        dateReturned: "not returned",
                        distance: rental.distance,
                        startLocation: rental.startLocation,
                        avgSpeed: rental.avgSpeed,
                        maxSpeed: rental.maxSpeed,
                        energyConsumed: rental.energyConsumed,
                        isActive: true,
                    },
                });

                const resultRentalVehicle = await
this.prisma.rentalVehicle.create({
                    data: {
                        rentalId: resultRental.id,
                        vehicleId: rental.vehicleId,
                    },
                });

                return {
                    id: resultRental.id,
                    userId: resultRental.userId,
                    dateRented: resultRental.dateRented,
                    dateReturned: resultRental.dateReturned,
                    startLocation: resultRental.startLocation,
                    distance: resultRental.distance,
                    avgSpeed: resultRental.avgSpeed,
                    maxSpeed: resultRental.maxSpeed,
                    energyConsumed: resultRental.energyConsumed,
                    isActive: resultRental.isActive,
                    vehicleId: resultRentalVehicle.vehicleId,
                    // rentalId: resultRentalVehicle.rentalId,
```

```
                };
            })
        } catch (error) {
            console.error('Error creating rental:', error);
            throw new BadRequestException('Failed to create
rental');
        }

    }

    async isInZone(lat: number, lng: number, geoJsonPolygon: any):
Promise<boolean> {
        const pt = point([lng, lat]);
        const poly = polygon(geoJsonPolygon.coordinates);
        return booleanPointInPolygon(pt, poly);
    }

    async endRentalFull(payment: PaymentRentalVehicleDto):
Promise<PaymentRentalVehicleDto> {
        console.log("parkingZones");
        // Находим активную аренду
        console.log("payment", payment)
        const rentalVehicle = await
this.prisma.rentalVehicle.findUnique({
            where: { id: payment.rentalVehicleId },
        });

        console.log("rentalVehicle", rentalVehicle)

        const rental = await this.prisma.rental.findUnique({
            where: { id: rentalVehicle.rentalId },
            include: { rentalVehicle: { include: { vehicle: true } }
},
        });

        console.log("rental", rental)

        if (!rental || !rental.isActive) {
            throw new BadRequestException('Rental is not active or
not found');
        }

        const vehicle = rental.rentalVehicle[0]?.vehicle;

        if (!vehicle) {
            throw new NotFoundException('Associated vehicle not
found');
        }

        // Рассчитываем изменения

        const traveledTime = (new Date().getTime() - new
Date(rental.dateRented).getTime()) / 3600000;
```

```
        const traveledDistance = (Math.random() * (25 - 15) + 10) *
traveledTime; // Дистанція у кілометрах

        // Приклад розрахунку середньої швидкості
        const avgSpeed = traveledDistance / traveledTime;
        const maxSpeed = 25; // Максимальна швидкість у км/год
        const energyPerKm = 20; // Середнє енергоспоживання на 1 км
у Wh

        // Фактор швидкості: якщо середня швидкість ближче до
максимальної, енергоспоживання зростає.
        const speedFactor = avgSpeed / maxSpeed > 1 ? 1 : avgSpeed /
maxSpeed;

        // Розрахунок витраченої енергії
        const energyConsumed = traveledDistance * energyPerKm *
speedFactor;
        const amount = energyConsumed * 0.1; // Пример расчета
оплаты: 0.1 доллара за 1 Wh

        const resultPayment = await this.prisma.payment.create({
            data: {
                rentalId: rental.id,
                paymentMethod: payment.paymentMethod,
                amount: amount.toFixed(2),
                date: new Date().toISOString(),
            },
        });

        // Обновляем статус транспортного средства
        await this.prisma.vehicle.update({
            where: { id: vehicle.id },
            data: {
                status: 'FREE',
                runnedDistance: vehicle.runnedDistance +
traveledDistance,
            },
        });

        // Обновляем заряд батареи
        const batteryVehicle = await
this.prisma.batteryVehicle.findFirst({
            where: { vehicleId: vehicle.id },
            include: { battery: true },
        });

        if (batteryVehicle?.battery) {
            await this.prisma.battery.update({
                where: { id: batteryVehicle.battery.id },
                data: {
                    chargeLevel: Math.max(
                        batteryVehicle.battery.chargeLevel -
energyConsumed,
```

```
                        0,
                    ),
                },
            });
        }

        // Завершаем аренду
        const updatedRental = await this.prisma.rental.update({
            where: { id: rental.id },
            data: {
                isActive: false,
                dateReturned: new Date().toISOString(),
                distance: traveledDistance,
                endLocation: payment.endLocation,
                avgSpeed: avgSpeed,
                maxSpeed: maxSpeed, // повинно бути знято з
контролера
                energyConsumed,
            },
        });

        return {
            ...resultPayment,
            rentalVehicleId: payment.rentalVehicleId,
            endLocation: payment.endLocation,
        };
    }

    // Метод для оновлення місцезнаходження транспортного засобу
    async updateVehicleLocation(vehicleId: string, lat: number, lng:
number) {
        try {
            // Переконайтемось, що транспорт існує
            const vehicle = await this.prisma.vehicle.findUnique({
                where: { id: vehicleId },
            });

            if (!vehicle) {
                throw new Error('Vehicle not found');
            }

            // Оновлюємо місцезнаходження з використанням PostGIS
            await this.prisma.$executeRaw`
        UPDATE "Vehicle"
        SET currentLocation = ST_SetSRID(ST_MakePoint(${lng},
${lat}), 4326)::geography
        WHERE id = ${vehicleId}
      `;

            return { success: true, vehicleId, location: { lat, lng
} };
        } catch (error) {
            console.error('Failed to update vehicle location:',
```

```
error);
            throw error;
        }
    }

    // Метод для пошуку найближчих транспортних засобів
    async findNearbyVehicles(lat: number, lng: number,
radiusInMeters: number = 500) {
        return this.prisma.$queryRaw`
      SELECT
        v.id,
        v.status,
        ST_Distance(
          v.currentLocation::geography,
          ST_SetSRID(ST_MakePoint(${lng}, ${lat}), 4326)::geography
        ) as distance
      FROM "Vehicle" v
      WHERE
        v.status = 'FREE'
        AND ST_DWithin(
          v.currentLocation::geography,
          ST_SetSRID(ST_MakePoint(${lng}, ${lat}), 4326)::geography,
          ${radiusInMeters}
        )
      ORDER BY distance
      LIMIT 10
    `;
    }

    async calculateAverageRentalDuration(): Promise<number> {
        // Отримуємо всі завершені оренди (де dateReturned не null)
        const rentals = await this.prisma.rental.findMany({
            where: {
                NOT: {
                    dateReturned: "",
                }
            },
            select: {
                dateRented: true,
                dateReturned: true,
            },
        });

        if (rentals.length === 0) return 0;

        // Підрахунок загального часу у годинах
        const totalDuration = rentals.reduce((sum, rental) => {
            const start = new Date(rental.dateRented).getTime();
            const end = new Date(rental.dateReturned).getTime();
            const durationHours = (end - start) / 3600000; //
мілісекунди → години
            return sum + durationHours;
        }, 0);
```

```
        // Середній час
        return Number((totalDuration / rentals.length).toFixed(2));
    }

    async getHourlyActivityDistribution(): Promise<{ [hour: string]:
number }> {
        const rentals = await this.prisma.rental.findMany({
            select: { dateRented: true },
        });

        // Об'єкт: { "0": 5, "1": 2, ..., "23": 7 }
        const distribution: { [hour: string]: number } = {};

        for (let i = 0; i < 24; i++) {
            distribution[i.toString()] = 0;
        }

        for (const rental of rentals) {
            if (!rental.dateRented) continue;
            const hour = new Date(rental.dateRented).getHours();
            distribution[hour.toString()]++;
        }

        return distribution;
    }

    async getDailyActivityDistribution(): Promise<{ [date: string]:
number }> {
        const rentals = await this.prisma.rental.findMany({
            select: { dateRented: true },
        });

        // Об'єкт: { "2025-06-01": 3, "2025-06-02": 7, ... }
        const distribution: { [date: string]: number } = {};

        for (const rental of rentals) {
            if (!rental.dateRented) continue;
            const date = new
Date(rental.dateRented).toISOString().slice(0, 10); // YYYY-MM-DD
            if (!distribution[date]) distribution[date] = 0;
            distribution[date]++;
        }

        return distribution;
    }
    async getIdleTimesForVehicle(vehicleId: string): Promise<Array<{
idleStart: string, idleEnd: string, idleHours: number }>> {
        // 1. Отримати всі оренди для цього транспорту, відсортувати
за dateRented
        const rentalVehicles = await
this.prisma.rentalVehicle.findMany({
            where: { vehicleId },
```

```
            include: {
                rental: {
                    select: {
                        dateRented: true,
                        dateReturned: true,
                    }
                }
            },
            orderBy: {
                rental: {
                    dateRented: 'asc'
                }
            }
        });

        // 2. Відфільтрувати тільки завершені оренди (де є
dateReturned)
        const completedRentals = rentalVehicles
            .map(rv => rv.rental)
            .filter(r => r.dateReturned && r.dateRented);

        // 3. Розрахувати простої між орендами
        const idleTimes: Array<{ idleStart: string, idleEnd: string,
idleHours: number }> = [];

        for (let i = 0; i < completedRentals.length - 1; i++) {
            const prev = completedRentals[i];
            const next = completedRentals[i + 1];

            // Простій: від повернення попередньої до старту
наступної
            const idleStart = prev.dateReturned;
            const idleEnd = next.dateRented;

            const start = new Date(idleStart).getTime();
            const end = new Date(idleEnd).getTime();
            const idleHours = (end - start) / 3600000;

            // Враховуємо тільки позитивний простій (можливо, дати
некоректні)
            if (idleHours >= 0) {
                idleTimes.push({
                    idleStart,
                    idleEnd,
                    idleHours: Number(idleHours.toFixed(2))
                });
            }
        }

        return idleTimes;
    }
    async getAllVehiclesIdleTimes(): Promise<Record<string, Array<{
idleStart: string, idleEnd: string, idleHours: number }>>> {
```

```
        const vehicles = await this.prisma.vehicle.findMany({
            select: { id: true }
        });

        const result: Record<string, Array<{ idleStart: string,
idleEnd: string, idleHours: number }>> = {};

        for (const vehicle of vehicles) {
            result[vehicle.id] = await
this.getIdleTimesForVehicle(vehicle.id);
        }

        return result;
    }

    /////============================

    // Функція для визначення ключа квадрата (grid cell)
    private getGridCellKey(lat: number, lng: number, cellSize =
0.002): string {
        const latIndex = Math.floor(lat / cellSize);
        const lngIndex = Math.floor(lng / cellSize);
        return `${latIndex}_${lngIndex}`;
    }

    // Найпопулярніші зони старту
    async getPopularStartZones(cellSize = 0.002, topN = 10) {
        console.log("zoneRevenue")
        const rentals = await this.prisma.rental.findMany({
            select: { startLocation: true }, // currentLocation =
стартова точка
        });
        console.log("rentals", rentals)
        const zoneCounts: Record<string, number> = {};

        for (const rental of rentals) {
            const loc = rental.startLocation as any;
            if (loc && loc.latitude !== undefined && loc.longitude
!== undefined) {
                const key = this.getGridCellKey(loc.latitude,
loc.longitude, cellSize);
                zoneCounts[key] = (zoneCounts[key] || 0) + 1;
            }
        }
        console.log("zoneCounts", zoneCounts)
        // Топ-N зон
        return Object.entries(zoneCounts)
            .sort((a, b) => b[1] - a[1])
            .slice(0, topN)
            .map(([cellKey, count]) => ({
                cellKey,
                count,
                // Центр квадрата для візуалізації
```

```
                centerLat: Number(cellKey.split('_')[0]) * cellSize
+ cellSize / 2,
                centerLng: Number(cellKey.split('_')[1]) * cellSize
+ cellSize / 2,
            }));
    }

    // Найпопулярніші зони фінішу (якщо є endLocation)
    async getPopularEndZones(cellSize = 0.002, topN = 10) {
        const rentals = await this.prisma.rental.findMany({
            select: { endLocation: true }, // Додай це поле у
Rental, якщо його ще немає!
        });

        const zoneCounts: Record<string, number> = {};

        for (const rental of rentals) {
            const loc = rental.endLocation as any;
            if (loc && loc.latitude !== undefined && loc.longitude
!== undefined) {
                const key = this.getGridCellKey(loc.latitude,
loc.longitude, cellSize);
                zoneCounts[key] = (zoneCounts[key] || 0) + 1;
            }
        }

        return Object.entries(zoneCounts)
            .sort((a, b) => b[1] - a[1])
            .slice(0, topN)
            .map(([cellKey, count]) => ({
                cellKey,
                count,
                centerLat: Number(cellKey.split('_')[0]) * cellSize
+ cellSize / 2,
                centerLng: Number(cellKey.split('_')[1]) * cellSize
+ cellSize / 2,
            }));
    }

    // Найприбутковіші зони старту
    async getMostProfitableStartZones(cellSize = 0.002, topN = 10) {
        console.log("zoneRevenue")
        // Витягуємо всі поїздки з платежами
        const rentals = await this.prisma.rental.findMany({
            select: {
                id: true,
                payment: { select: { amount: true } }, // payment
має бути зв'язком rentalId → Payment
                startLocation: true,
            },
        });

        const zoneRevenue: Record<string, number> = {};
```

```
        for (const rental of rentals) {
            const loc = rental.startLocation as any;
            const amount = rental.payment?.amount ?
Number(rental.payment.amount) : 0;
            if (loc && loc.latitude !== undefined && loc.longitude
!== undefined) {
                const key = this.getGridCellKey(loc.latitude,
loc.longitude, cellSize);
                zoneRevenue[key] = (zoneRevenue[key] || 0) + amount;
            }
        }
        console.log("zoneRevenue", zoneRevenue)

        return Object.entries(zoneRevenue)
            .sort((a, b) => b[1] - a[1])
            .slice(0, topN)
            .map(([cellKey, totalRevenue]) => ({
                cellKey,
                totalRevenue,
                centerLat: Number(cellKey.split('_')[0]) * cellSize
+ cellSize / 2,
                centerLng: Number(cellKey.split('_')[1]) * cellSize
+ cellSize / 2,
            }));
    }
}
```

ДОДАТОК Е

Логіка взаємодії серверної частини оренди (приклад розробленого контролера для таблиці Rental)

```
import { Body, Controller, Delete, Get, HttpCode, HttpException,
HttpStatus, InternalServerErrorException, NotFoundException, Param,
Post, Put, Query, UseGuards } from '@nestjs/common';
import { ApiBearerAuth, ApiResponse, ApiTags } from
'@nestjs/swagger';
import { RentalService } from './rental.service';
import { RentalDto } from './dtos/rental.dto';
import { UpdateRentalDto } from './dtos/update-rental.dto';
import { PaymentMethodTypes } from '@prisma/client';
import { CreateRentalDto } from './dtos/create-rental.dto';

import { RentalVehicleDto } from 'src/rental-vehicle/dtos/rental-
vehicle.dto';
import { PaymentDto } from 'src/user/dtos/userPlus.dto';
import { RentalNotFoundException } from 'src/exceptions/user-
exceptions';
import { Roles } from 'src/auth/roles.decorator';
import { Role } from '@prisma/client';
```

```
import { RoleGuard } from 'src/auth/roles.guard';
import { JwtAuthGuard } from 'src/auth/jwt-auth.guard';
import { OwnershipGuard } from 'src/auth/ownership.guard';
import { RentalFullDto } from './dtos/rental-full.dto';
import { CreateRentalFullDto } from './dtos/create-rental-full.dto';
import { PaymentRentalVehicleDto } from
'./dtos/paymentRentalVehicle.dto';

@ApiTags('rental')
@Controller('rental')
export class RentalController {
    constructor(private readonly rentalService: RentalService) { }

    @Get()
    @UseGuards(JwtAuthGuard, RoleGuard)
    @Roles(Role.MODERATOR, Role.ADMIN)
    @ApiBearerAuth()
    @HttpCode(200)
    @ApiResponse({ status: 200, description: 'List of all rentals
returned successfully.' })
    @ApiResponse({ status: 500, description: 'Internal server
error.' })
    public async findAllRental() {
        const result = await this.rentalService.findAllRental();
        return result;
    }

    @Post(':vehicleId')
    @UseGuards(JwtAuthGuard, RoleGuard)
    @Roles(Role.ADMIN, Role.USER)
    @ApiBearerAuth()
    @HttpCode(201)
    @ApiResponse({ status: 201, description: 'Rental created
successfully.' })
    @ApiResponse({ status: 400, description: 'Invalid input data.'
})
    @ApiResponse({ status: 500, description: 'Internal server
error.' })
    public async createRental(@Body() rental: RentalDto) {
        const result = await
this.rentalService.createRental(rental);
        return result;
    }

    @Put(':id')
    @UseGuards(JwtAuthGuard, RoleGuard)
    @Roles(Role.ADMIN, Role.USER)
    @ApiBearerAuth()
    @HttpCode(200)
    @ApiResponse({ status: 200, description: 'Rental updated
successfully.' })
    @ApiResponse({ status: 404, description: 'Rental not found.' })
    @ApiResponse({ status: 500, description: 'Internal server
```

```
error.' })
    public async updateRental(@Body() rental: UpdateRentalDto,
@Param('id') id: string) {
        const result = await this.rentalService.updateRental(rental,
id);
        return result;
    }

    @Delete(':id')
    @UseGuards(JwtAuthGuard, RoleGuard)
    @Roles(Role.ADMIN, Role.USER)
    @ApiBearerAuth()
    @HttpCode(200)
    @ApiResponse({ status: 200, description: 'Rental deleted
successfully.' })
    @ApiResponse({ status: 404, description: 'Rental not found.' })
    @ApiResponse({ status: 500, description: 'Internal server
error.' })
    public async deleteRental(@Param('id') id: string) {
        const result = await this.rentalService.deleteRental(id);
        return result;
    }

    @Get('total')
    @UseGuards(JwtAuthGuard, RoleGuard)
    @Roles(Role.ADMIN)
    @ApiBearerAuth()
    @HttpCode(200)
    async getTotalProfit(
        @Query('startDate') startDate: string,
        @Query('endDate') endDate: string
    ): Promise<{ totalProfit: number }> {
        try {
            const totalProfit = await
this.rentalService.calculateTotalProfit(startDate, endDate);
            return { totalProfit };
        } catch (error) {
            throw new Error(`Error while calculating total profit:
${error.message}`);
        }
    }

    @Get(':userId/rentals-with-vehicles')
    @UseGuards(JwtAuthGuard, RoleGuard)
    @Roles(Role.ADMIN, Role.MODERATOR)
    @ApiBearerAuth()
    @ApiResponse({ status: 200, description: 'List of rentals with
vehicles.' })
    @ApiResponse({ status: 404, description: 'User not found.' })
    public async getUserRentalsWithVehicles(@Param('userId') userId:
string) {
        const result = await
this.rentalService.getUserRentalsWithVehicles(userId);
```

```
        return result;
    }

    @Get(':id')
    @UseGuards(JwtAuthGuard, RoleGuard)
    @Roles(Role.ADMIN, Role.MODERATOR)
    @ApiBearerAuth()
    @HttpCode(200)
    async getRentalById(@Param('id') id: string) {
        try {
            const rental = await
this.rentalService.getRentalById(id);
            return rental;
        } catch (error) {
            if (error instanceof RentalNotFoundException) {
                throw new NotFoundException(error.message);  // HTTP
404
            }
            throw new InternalServerErrorException('Internal Server
Error');
        }
    }

    @Post('full/start/:data')
    @UseGuards(JwtAuthGuard, RoleGuard)
    @Roles(Role.ADMIN, Role.USER)
    @ApiBearerAuth()
    @HttpCode(201)
    @ApiResponse({ status: 201, description: 'Rental created
successfully.' })
    @ApiResponse({ status: 400, description: 'Invalid input data.'
})
    @ApiResponse({ status: 500, description: 'Internal server
error.' })
    public async createRentalFull(@Body() rental: RentalFullDto) {
        const result = await
this.rentalService.createRentalFull(rental);
        return result;
    }

    @Post('full/end/:payment')
    @UseGuards(JwtAuthGuard, RoleGuard)
    @Roles(Role.ADMIN, Role.USER)
    @ApiBearerAuth()
    @HttpCode(201)
    @ApiResponse({ status: 201, description: 'Rental created
successfully.' })
    @ApiResponse({ status: 400, description: 'Invalid input data.'
})
    @ApiResponse({ status: 500, description: 'Internal server
error.' })
    public async endRentalFull(@Body() paymentRentalVehicle:
PaymentRentalVehicleDto) {
```

```
        const result = await
this.rentalService.endRentalFull(paymentRentalVehicle);
        return result;
    }

    @Get('all/rental/average-duration')
    @UseGuards(JwtAuthGuard, RoleGuard)
    @Roles(Role.MODERATOR, Role.ADMIN, Role.TECHNICIAN)
    @ApiBearerAuth()
    @HttpCode(200)
    @ApiResponse({ status: 200, description: 'Average rental
duration returned successfully.' })
    @ApiResponse({ status: 500, description: 'Internal server
error.' })
    public async calculateAverageRentalDuration(): Promise<number> {
        return await
this.rentalService.calculateAverageRentalDuration();
    }

    @Get('activity/hourly')
    @UseGuards(JwtAuthGuard, RoleGuard)
    @Roles(Role.MODERATOR, Role.ADMIN, Role.TECHNICIAN)
    @ApiBearerAuth()
    @ApiResponse({ status: 200, description: 'Hourly activity
distribution returned successfully.' })
    @ApiResponse({ status: 500, description: 'Internal server
error.' })
    public async getHourlyActivityDistribution() {
        return await
this.rentalService.getHourlyActivityDistribution();
    }

    @Get('activity/daily')
    @UseGuards(JwtAuthGuard, RoleGuard)
    @Roles(Role.MODERATOR, Role.ADMIN, Role.TECHNICIAN)
    @ApiBearerAuth()
    @ApiResponse({ status: 200, description: 'Daily activity
distribution returned successfully.' })
    @ApiResponse({ status: 500, description: 'Internal server
error.' })
    public async getDailyActivityDistribution() {
        return await
this.rentalService.getDailyActivityDistribution();
    }

    @Get('idle-times/:vehicleId')
    @UseGuards(JwtAuthGuard, RoleGuard)
    @Roles(Role.MODERATOR, Role.ADMIN, Role.TECHNICIAN)
    @ApiBearerAuth()
    @ApiResponse({ status: 200, description: 'Idle times for vehicle
returned successfully.' })
    @ApiResponse({ status: 500, description: 'Internal server
error.' })
```

```
    public async getIdleTimesForVehicle(@Param('vehicleId')
vehicleId: string) {
        return await
this.rentalService.getIdleTimesForVehicle(vehicleId);
    }

    @Get('idle-times/all')
    @UseGuards(JwtAuthGuard, RoleGuard)
    @Roles(Role.MODERATOR, Role.ADMIN, Role.TECHNICIAN)
    @ApiBearerAuth()
    @ApiResponse({ status: 200, description: 'Idle times for vehicle
returned successfully.' })
    @ApiResponse({ status: 500, description: 'Internal server
error.' })
    public async getAllVehiclesIdleTimes() {
        return await this.rentalService.getAllVehiclesIdleTimes();
    }

    @Get('popular-start-zones/:cellSize/:topN')
    @UseGuards(JwtAuthGuard, RoleGuard)
    @Roles(Role.ADMIN, Role.MODERATOR, Role.TECHNICIAN)
    @ApiBearerAuth()
    @ApiResponse({ status: 200, description: 'Popular start zones
returned.' })
    @ApiResponse({ status: 500, description: 'Internal server
error.' })
    public async getPopularStartZones(@Param('cellSize') cellSize:
number, @Param('topN') topN: number) {
        // @Query('cellSize') cellSize?: number,
        // @Query('topN') topN?: number
        console.log("zoneRevenue")
        return this.rentalService.getPopularStartZones(cellSize,
topN);
    }

    @Get('popular-end-zones/:cellSize/:topN')
    @UseGuards(JwtAuthGuard, RoleGuard)
    @Roles(Role.ADMIN, Role.MODERATOR, Role.TECHNICIAN)
    @ApiBearerAuth()
    @ApiResponse({ status: 200, description: 'Popular end zones
returned.' })
    @ApiResponse({ status: 500, description: 'Internal server
error.' })
    async getPopularEndZones(@Param('cellSize') cellSize: number,
@Param('topN') topN: number
        // @Query('cellSize') cellSize?: number,
        // @Query('topN') topN?: number
    ) {
        return this.rentalService.getPopularEndZones(cellSize,
topN);
    }

    @Get('most-profitable-start-zones/:cellSize/:topN')
```

```
    @UseGuards(JwtAuthGuard, RoleGuard)
    @Roles(Role.ADMIN, Role.MODERATOR, Role.TECHNICIAN)
    @ApiBearerAuth()
    @ApiResponse({ status: 200, description: 'Most profitable start
zones returned.' })
    @ApiResponse({ status: 500, description: 'Internal server
error.' })
    async getMostProfitableStartZones(
        @Param('cellSize') cellSize: number, @Param('topN') topN:
number
    ) {
        return
this.rentalService.getMostProfitableStartZones(cellSize, topN);
    }
}
```

ДОДАТОК Ж
Специфікація API


```
{
    "openapi": "3.0.0",
    "paths": {
        "/": {
            "get": {
                "operationId": "AppController_getHello",
                "parameters": [],
                "responses": {
                    "200": {
                        "description": ""
                    }
                },
                "tags": [
                    "App"
                ]
            }
        },
        "/user": {
            "get": {
                "operationId": "UserController_findAllUser",
                "parameters": [],
                "responses": {
                    "200": {
                        "description": "List of all users returned
successfully."
                    },
                    "500": {
                        "description": "Internal server error."
                    }
                },
                "tags": [
                    "user"
                ]
            },
            "post": {
                "operationId": "UserController_createUser",
                "parameters": [],
                "requestBody": {
                    "required": true,
                    "content": {
                        "application/json": {
                            "schema": {
                                "$ref":
"#/components/schemas/UserDto"
                            }
                        }
                    }
                },
                "responses": {
```

```
                    "201": {
                        "description": "User created successfully."
                    },
                    "400": {
                        "description": "Invalid input data."
                    },
                    "500": {
                        "description": "Internal server error."
                    }
                },
                "tags": [
                    "user"
                ]
            }
        },
        "/user/{id}": {
            "put": {
                "operationId": "UserController_updateUser",
                "parameters": [
                    {
                        "name": "id",
                        "required": true,
                        "in": "path",
                        "schema": {
                            "type": "string"
                        }
                    }
                ],
                "requestBody": {
                    "required": true,
                    "content": {
                        "application/json": {
                            "schema": {
                                "$ref":
"#/components/schemas/UpdateUserDto"
                            }
                        }
                    }
                },
                "responses": {
                    "200": {
                        "description": "User updated successfully."
                    },
                    "404": {
                        "description": "User not found."
                    },
                    "500": {
                        "description": "Internal server error."
                    }
                },
                "tags": [
                    "user"
                ]
```

```
            },
            "delete": {
                "operationId": "UserController_deleteUser",
                "parameters": [
                    {
                        "name": "id",
                        "required": true,
                        "in": "path",
                        "schema": {
                            "type": "string"
                        }
                    }
                ],
                "responses": {
                    "200": {
                        "description": "User deleted successfully."
                    },
                    "404": {
                        "description": "User not found."
                    },
                    "500": {
                        "description": "Internal server error."
                    }
                },
                "tags": [
                    "user"
                ]
            },
            "get": {
                "operationId": "UserController_getUserDetails",
                "parameters": [
                    {
                        "name": "id",
                        "required": true,
                        "in": "path",
                        "schema": {
                            "type": "string"
                        }
                    }
                ],
                "responses": {
                    "200": {
                        "description": "User details returned
successfully."
                    },
                    "404": {
                        "description": "User not found."
                    },
                    "500": {
                        "description": "Internal server error."
                    }
                },
                "tags": [
```

```
                    "user"
                ]
            }
        },
        "/rental": {
            "get": {
                "operationId": "RentalController_findAllRental",
                "parameters": [],
                "responses": {
                    "200": {
                        "description": "List of all rentals returned
successfully."
                    },
                    "500": {
                        "description": "Internal server error."
                    }
                },
                "tags": [
                    "rental"
                ]
            },
            "post": {
                "operationId": "RentalController_createRental",
                "parameters": [],
                "requestBody": {
                    "required": true,
                    "content": {
                        "application/json": {
                            "schema": {
                                "$ref":
"#/components/schemas/RentalDto"
                            }
                        }
                    }
                },
                "responses": {
                    "201": {
                        "description": "Rental created
successfully."
                    },
                    "400": {
                        "description": "Invalid input data."
                    },
                    "500": {
                        "description": "Internal server error."
                    }
                },
                "tags": [
                    "rental"
                ]
            }
        },
        "/rental/{id}": {
```

```
            "put": {
                "operationId": "RentalController_updateRental",
                "parameters": [
                    {
                        "name": "id",
                        "required": true,
                        "in": "path",
                        "schema": {
                            "type": "string"
                        }
                    }
                ],
                "requestBody": {
                    "required": true,
                    "content": {
                        "application/json": {
                            "schema": {
                                "$ref":
"#/components/schemas/UpdateRentalDto"
                            }
                        }
                    }
                },
                "responses": {
                    "200": {
                        "description": "Rental updated
successfully."
                    },
                    "404": {
                        "description": "Rental not found."
                    },
                    "500": {
                        "description": "Internal server error."
                    }
                },
                "tags": [
                    "rental"
                ]
            },
            "delete": {
                "operationId": "RentalController_deleteRental",
                "parameters": [
                    {
                        "name": "id",
                        "required": true,
                        "in": "path",
                        "schema": {
                            "type": "string"
                        }
                    }
                ],
                "responses": {
                    "200": {
```

```
                        "description": "Rental deleted
successfully."
                    },
                    "404": {
                        "description": "Rental not found."
                    },
                    "500": {
                        "description": "Internal server error."
                    }
                },
                "tags": [
                    "rental"
                ]
            }
        },
        "/payment": {
            "get": {
                "operationId": "PaymentController_findAllPayment",
                "parameters": [],
                "responses": {
                    "200": {
                        "description": "List of all payments
returned successfully."
                    },
                    "500": {
                        "description": "Internal server error."
                    }
                },
                "tags": [
                    "payment"
                ]
            },
            "post": {
                "operationId": "PaymentController_createPayment",
                "parameters": [],
                "requestBody": {
                    "required": true,
                    "content": {
                        "application/json": {
                            "schema": {
                                "$ref":
"#/components/schemas/PaymentDto"
                            }
                        }
                    }
                },
                "responses": {
                    "201": {
                        "description": "Payment created
successfully."
                    },
                    "400": {
                        "description": "Invalid input data."
```

```
                    },
                    "500": {
                        "description": "Internal server error."
                    }
                },
                "tags": [
                    "payment"
                ]
            }
        },
        "/payment/{id}": {
            "put": {
                "operationId": "PaymentController_updatePayment",
                "parameters": [
                    {
                        "name": "id",
                        "required": true,
                        "in": "path",
                        "schema": {
                            "type": "string"
                        }
                    }
                ],
                "requestBody": {
                    "required": true,
                    "content": {
                        "application/json": {
                            "schema": {
                                "$ref":
"#/components/schemas/UpdatePaymentDto"
                            }
                        }
                    }
                },
                "responses": {
                    "200": {
                        "description": "Payment updated
successfully."
                    },
                    "404": {
                        "description": "Payment not found."
                    },
                    "500": {
                        "description": "Internal server error."
                    }
                },
                "tags": [
                    "payment"
                ]
            },
            "delete": {
                "operationId": "PaymentController_deletePayment",
                "parameters": [
```

```
                    {
                        "name": "id",
                        "required": true,
                        "in": "path",
                        "schema": {
                            "type": "string"
                        }
                    }
                ],
                "responses": {
                    "200": {
                        "description": "Payment deleted
successfully."
                    },
                    "404": {
                        "description": "Payment not found."
                    },
                    "500": {
                        "description": "Internal server error."
                    }
                },
                "tags": [
                    "payment"
                ]
            }
        },
        "/vehicle": {
            "get": {
                "operationId": "VehicleController_findAllVehicle",
                "parameters": [],
                "responses": {
                    "200": {
                        "description": "List of all vehicles
returned successfully."
                    },
                    "500": {
                        "description": "Internal server error."
                    }
                },
                "tags": [
                    "vehicle"
                ]
            },
            "post": {
                "operationId": "VehicleController_createVehicle",
                "parameters": [],
                "requestBody": {
                    "required": true,
                    "content": {
                        "application/json": {
                            "schema": {
                                "$ref":
"#/components/schemas/VehicleDto"
```

```
                            }
                        }
                    }
                },
                "responses": {
                    "201": {
                        "description": "Vehicle created
successfully."
                    },
                    "400": {
                        "description": "Invalid input data."
                    },
                    "500": {
                        "description": "Internal server error."
                    }
                },
                "tags": [
                    "vehicle"
                ]
            }
        },
        "/vehicle/{id}": {
            "put": {
                "operationId": "VehicleController_updateVehicle",
                "parameters": [
                    {
                        "name": "id",
                        "required": true,
                        "in": "path",
                        "schema": {
                            "type": "string"
                        }
                    }
                ],
                "requestBody": {
                    "required": true,
                    "content": {
                        "application/json": {
                            "schema": {
                                "$ref":
"#/components/schemas/UpdateVehicleDto"
                            }
                        }
                    }
                },
                "responses": {
                    "200": {
                        "description": "Vehicle updated
successfully."
                    },
                    "404": {
                        "description": "Vehicle not found."
                    },
```

```
                    "500": {
                        "description": "Internal server error."
                    }
                },
                "tags": [
                    "vehicle"
                ]
            },
            "delete": {
                "operationId": "VehicleController_deleteVehicle",
                "parameters": [
                    {
                        "name": "id",
                        "required": true,
                        "in": "path",
                        "schema": {
                            "type": "string"
                        }
                    }
                ],
                "responses": {
                    "200": {
                        "description": "Vehicle deleted
successfully."
                    },
                    "404": {
                        "description": "Vehicle not found."
                    },
                    "500": {
                        "description": "Internal server error."
                    }
                },
                "tags": [
                    "vehicle"
                ]
            }
        },
        "/battery": {
            "get": {
                "operationId": "BatteryController_findAllBattery",
                "parameters": [],
                "responses": {
                    "200": {
                        "description": "List of all batteries
returned successfully."
                    },
                    "500": {
                        "description": "Internal server error."
                    }
                },
                "tags": [
                    "battery"
                ]
```

```
            },
            "post": {
                "operationId": "BatteryController_createBattery",
                "parameters": [],
                "requestBody": {
                    "required": true,
                    "content": {
                        "application/json": {
                            "schema": {
                                "$ref":
"#/components/schemas/BatteryDto"
                            }
                        }
                    }
                },
                "responses": {
                    "201": {
                        "description": "Battery created
successfully."
                    },
                    "400": {
                        "description": "Invalid input data."
                    },
                    "500": {
                        "description": "Internal server error."
                    }
                },
                "tags": [
                    "battery"
                ]
            }
        },
        "/battery/{id}": {
            "put": {
                "operationId": "BatteryController_updateBattery",
                "parameters": [
                    {
                        "name": "id",
                        "required": true,
                        "in": "path",
                        "schema": {
                            "type": "string"
                        }
                    }
                ],
                "requestBody": {
                    "required": true,
                    "content": {
                        "application/json": {
                            "schema": {
                                "$ref":
"#/components/schemas/UpdateBatteryDto"
                            }
```

```
                        }
                    }
                },
                "responses": {
                    "200": {
                        "description": "Battery updated
successfully."
                    },
                    "404": {
                        "description": "Battery not found."
                    },
                    "500": {
                        "description": "Internal server error."
                    }
                },
                "tags": [
                    "battery"
                ]
            },
            "delete": {
                "operationId": "BatteryController_deleteBattery",
                "parameters": [
                    {
                        "name": "id",
                        "required": true,
                        "in": "path",
                        "schema": {
                            "type": "string"
                        }
                    }
                ],
                "responses": {
                    "200": {
                        "description": "Battery deleted
successfully."
                    },
                    "404": {
                        "description": "Battery not found."
                    },
                    "500": {
                        "description": "Internal server error."
                    }
                },
                "tags": [
                    "battery"
                ]
            }
        },
        "/rental-vehicle": {
            "get": {
                "operationId":
"RentalVehicleController_findAllRentalVehicle",
                "parameters": [],
```

```
                "responses": {
                    "200": {
                        "description": "List of all rental vehicles
returned successfully."
                    },
                    "500": {
                        "description": "Internal server error."
                    }
                },
                "tags": [
                    "rental-vehicle"
                ]
            },
            "post": {
                "operationId":
"RentalVehicleController_createRentalVehicle",
                "parameters": [],
                "requestBody": {
                    "required": true,
                    "content": {
                        "application/json": {
                            "schema": {
                                "$ref":
"#/components/schemas/RentalVehicleDto"
                            }
                        }
                    }
                },
                "responses": {
                    "201": {
                        "description": "Rental vehicle created
successfully."
                    },
                    "400": {
                        "description": "Invalid input data."
                    },
                    "500": {
                        "description": "Internal server error."
                    }
                },
                "tags": [
                    "rental-vehicle"
                ]
            }
        },
        "/rental-vehicle/{id}": {
            "put": {
                "operationId":
"RentalVehicleController_updateRentalVehicle",
                "parameters": [
                    {
                        "name": "id",
                        "required": true,
```

```
                        "in": "path",
                        "schema": {
                            "type": "string"
                        }
                    }
                ],
                "requestBody": {
                    "required": true,
                    "content": {
                        "application/json": {
                            "schema": {
                                "$ref":
"#/components/schemas/UpdateRentalVehicleDto"
                            }
                        }
                    }
                },
                "responses": {
                    "200": {
                        "description": "Rental vehicle updated
successfully."
                    },
                    "404": {
                        "description": "Rental vehicle not found."
                    },
                    "500": {
                        "description": "Internal server error."
                    }
                },
                "tags": [
                    "rental-vehicle"
                ]
            },
            "delete": {
                "operationId":
"RentalVehicleController_deleteRentalVehicle",
                "parameters": [
                    {
                        "name": "id",
                        "required": true,
                        "in": "path",
                        "schema": {
                            "type": "string"
                        }
                    }
                ],
                "responses": {
                    "200": {
                        "description": "Rental vehicle deleted
successfully."
                    },
                    "404": {
                        "description": "Rental vehicle not found."
```

```
                    },
                    "500": {
                        "description": "Internal server error."
                    }
                },
                "tags": [
                    "rental-vehicle"
                ]
            }
        },
        "/battery-vehicle": {
            "get": {
                "operationId":
"BatteryVehicleController_findAllBatteryVehicle",
                "parameters": [],
                "responses": {
                    "200": {
                        "description": "List of all battery vehicles
returned successfully."
                    },
                    "500": {
                        "description": "Internal server error."
                    }
                },
                "tags": [
                    "battery-vehicle"
                ]
            },
            "post": {
                "operationId":
"BatteryVehicleController_createBatteryVehicle",
                "parameters": [],
                "requestBody": {
                    "required": true,
                    "content": {
                        "application/json": {
                            "schema": {
                                "$ref":
"#/components/schemas/BatteryVehicleDto"
                            }
                        }
                    }
                },
                "responses": {
                    "201": {
                        "description": "Battery vehicle created
successfully."
                    },
                    "400": {
                        "description": "Invalid input data."
                    },
                    "500": {
                        "description": "Internal server error."
```

```
                    }
                },
                "tags": [
                    "battery-vehicle"
                ]
            }
        },
        "/battery-vehicle/{id}": {
            "put": {
                "operationId":
"BatteryVehicleController_updateBatteryVehicle",
                "parameters": [
                    {
                        "name": "id",
                        "required": true,
                        "in": "path",
                        "schema": {
                            "type": "string"
                        }
                    }
                ],
                "requestBody": {
                    "required": true,
                    "content": {
                        "application/json": {
                            "schema": {
                                "$ref":
"#/components/schemas/UpdateBatteryVehicleDto"
                            }
                        }
                    }
                },
                "responses": {
                    "200": {
                        "description": "Battery vehicle updated
successfully."
                    },
                    "404": {
                        "description": "Battery vehicle not found."
                    },
                    "500": {
                        "description": "Internal server error."
                    }
                },
                "tags": [
                    "battery-vehicle"
                ]
            },
            "delete": {
                "operationId":
"BatteryVehicleController_deleteBatteryVehicle",
                "parameters": [
                    {
```

```
                        "name": "id",
                        "required": true,
                        "in": "path",
                        "schema": {
                            "type": "string"
                        }
                    }
                ],
                "responses": {
                    "200": {
                        "description": "Battery vehicle deleted
successfully."
                    },
                    "404": {
                        "description": "Battery vehicle not found."
                    },
                    "500": {
                        "description": "Internal server error."
                    }
                },
                "tags": [
                    "battery-vehicle"
                ]
            }
        }
    },
    "info": {
        "title": "e-transport",
        "description": "The e-transport API description",
        "version": "1.0",
        "contact": {}
    },
    "tags": [],
    "servers": [],
    "components": {
        "schemas": {
            "UserDto": {
                "type": "object",
                "properties": {
                    "email": {
                        "type": "string"
                    },
                    "name": {
                        "type": "string"
                    },
                    "password": {
                        "type": "string"
                    },
                    "phoneNumber": {
                        "type": "string"
                    },
                    "bonusAccount": {
                        "type": "string"
```

```
            },
            "notification": {
                "type": "boolean"
            },
            "photo": {
                "type": "string"
            },
            "role": {
                "type": "string",
                "enum": [
                    "ADMIN",
                    "USER",
                    "MODERATOR",
                    "TECHNICIAN"
                ]
            }
        },
        "required": [
            "email",
            "name",
            "password",
            "phoneNumber",
            "bonusAccount",
            "notification",
            "photo",
            "role"
        ]
    },
    "UpdateUserDto": {
        "type": "object",
        "properties": {
            "email": {
                "type": "string"
            },
            "name": {
                "type": "string"
            },
            "password": {
                "type": "string"
            },
            "phoneNumber": {
                "type": "string"
            },
            "bonusAccount": {
                "type": "string"
            },
            "photo": {
                "type": "string"
            },
            "notification": {
                "type": "boolean"
            },
            "role": {
```

```
                "type": "string",
                "enum": [
                    "ADMIN",
                    "USER",
                    "MODERATOR",
                    "TECHNICIAN"
                ]
            }
        },
        "required": [
            "role"
        ]
    },
    "RentalDto": {
        "type": "object",
        "properties": {
            "isActive": {
                "type": "boolean"
            },
            "dateRented": {
                "type": "string"
            },
            "dateReturned": {
                "type": "string"
            },
            "userId": {
                "type": "string"
            },
            "distance": {
                "type": "number"
            },
            "avgSpeed": {
                "type": "number"
            },
            "maxSpeed": {
                "type": "number"
            },
            "energyConsumed": {
                "type": "number"
            }
        },
        "required": [
            "isActive",
            "dateRented",
            "dateReturned",
            "userId",
            "distance",
            "avgSpeed",
            "maxSpeed",
            "energyConsumed"
        ]
    },
    "UpdateRentalDto": {
```

```
        "type": "object",
        "properties": {
            "isActive": {
                "type": "boolean"
            },
            "dateRented": {
                "type": "string"
            },
            "dateReturned": {
                "type": "string"
            },
            "userId": {
                "type": "string"
            },
            "distance": {
                "type": "number"
            },
            "avgSpeed": {
                "type": "number"
            },
            "maxSpeed": {
                "type": "number"
            },
            "energyConsumed": {
                "type": "number"
            }
        }
    },
    "PaymentDto": {
        "type": "object",
        "properties": {
            "paymentMethod": {
                "type": "string",
                "enum": [
                    "VISA",
                    "PAYPAL",
                    "MASTERCARD",
                    "OTHER"
                ]
            },
            "amount": {
                "type": "string"
            },
            "date": {
                "type": "string"
            },
            "rentalId": {
                "type": "string"
            }
        },
        "required": [
            "paymentMethod",
            "amount",
```

```
            "date",
            "rentalId"
        ]
    },
    "UpdatePaymentDto": {
        "type": "object",
        "properties": {
            "paymentMethod": {
                "type": "string",
                "enum": [
                    "VISA",
                    "PAYPAL",
                    "MASTERCARD",
                    "OTHER"
                ]
            },
            "name": {
                "type": "string"
            },
            "amount": {
                "type": "string"
            },
            "date": {
                "type": "string"
            },
            "rentalId": {
                "type": "string"
            }
        },
        "required": [
            "paymentMethod"
        ]
    },
    "VehicleDto": {
        "type": "object",
        "properties": {
            "status": {
                "type": "string",
                "enum": [
                    "FREE",
                    "INUSE",
                    "NOTAVAILABLE",
                    "BROKEN",
                    "REPAIR"
                ]
            },
            "runnedDistance": {
                "type": "number"
            },
            "releaseDate": {
                "type": "string"
            },
            "currentLocation": {
```

```
                "type": "string"
            }
        },
        "required": [
            "status",
            "runnedDistance",
            "releaseDate",
            "currentLocation"
        ]
    },
    "UpdateVehicleDto": {
        "type": "object",
        "properties": {
            "status": {
                "type": "string",
                "enum": [
                    "FREE",
                    "INUSE",
                    "NOTAVAILABLE",
                    "BROKEN",
                    "REPAIR"
                ]
            },
            "runnedDistance": {
                "type": "number"
            },
            "releaseDate": {
                "type": "string"
            },
            "currentLocation": {
                "type": "string"
            }
        },
        "required": [
            "status"
        ]
    },
    "BatteryDto": {
        "type": "object",
        "properties": {
            "status": {
                "type": "string",
                "enum": [
                    "INUSE",
                    "NOTINUSE",
                    "BROKEN",
                    "CHARGING",
                    "REPAIR"
                ]
            },
            "chargeLevel": {
                "type": "number"
            },
```

```
            "condition": {
                "type": "string"
            },
            "type": {
                "type": "string",
                "enum": [
                    "LithiumIon",
                    "LithiumManganese",
                    "LeadAcid"
                ]
            },
            "capacity": {
                "type": "number"
            }
        },
        "required": [
            "status",
            "chargeLevel",
            "condition",
            "type",
            "capacity"
        ]
    },
    "UpdateBatteryDto": {
        "type": "object",
        "properties": {
            "chargeLevel": {
                "type": "number"
            },
            "status": {
                "type": "string",
                "enum": [
                    "INUSE",
                    "NOTINUSE",
                    "BROKEN",
                    "CHARGING",
                    "REPAIR"
                ]
            },
            "condition": {
                "type": "string"
            },
            "type": {
                "type": "string",
                "enum": [
                    "LithiumIon",
                    "LithiumManganese",
                    "LeadAcid"
                ]
            },
            "capacity": {
                "type": "number"
            }
```

```
        },
        "required": [
            "status",
            "type"
        ]
    },
    "RentalVehicleDto": {
        "type": "object",
        "properties": {
            "vehicleId": {
                "type": "string"
            },
            "rentalId": {
                "type": "string"
            }
        },
        "required": [
            "vehicleId",
            "rentalId"
        ]
    },
    "UpdateRentalVehicleDto": {
        "type": "object",
        "properties": {
            "vehicleId": {
                "type": "string"
            },
            "rentalId": {
                "type": "string"
            }
        }
    },
    "BatteryVehicleDto": {
        "type": "object",
        "properties": {
            "vehicleId": {
                "type": "string"
            },
            "batteryId": {
                "type": "string"
            }
        },
        "required": [
            "vehicleId",
            "batteryId"
        ]
    },
    "UpdateBatteryVehicleDto": {
        "type": "object",
        "properties": {
            "vehicleId": {
                "type": "string"
            },
```

```
                    "batteryId": {
                        "type": "string"
                    }
                }
            }
        }
    }
}
```