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

Лабораторна робота №4
з дисципліни: «Архітектура програмного забезпечення»
на тему: «РОЗРОБКА ПРОГРАМНОЇ АРХІТЕКТУРИ, СТВОРЕННЯ ТА ВІДЛАГОДЖЕННЯ ПРОГРАМНОГО КОДУ МОБІЛЬНОГО КЛІЄНТА 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

1 ІСТОРІЯ ЗМІН

№	Дата	Версія звіту	Опис змін та виправлень
1	28.05.2025	0.1	Створено усі розділи

2 ЗАВДАННЯ

Метою роботи є розробити клієнтську / front-end частину програмної системи.

3 ОПИС ВИКОНАНОЇ РОБОТИ

Назва: «Система для підтримки та регуляції прокату електротранспорту».

Назва англійською мовою: «Electric Transport Rental Management System».

Власна назва: «E-Transport».

У процесі виконання проєкту було розроблено комплексну систему для організації, аналітики та управління парком електросамокатів. Система складається з двох основних частин: мобільного застосунку для користувачів і адміністративної панелі для операторів. Уся інфраструктура взаємодіє із серверною частиною, що забезпечує зберігання, обробку даних та бізнес-логіку.

Мобільний застосунок було реалізовано за допомогою Flutter, що забезпечило кросплатформенність і високу швидкість розробки. Інтеграція з картографічною системою Mapbox дала змогу відображати в реальному часі місцезнаходження самокатів, зони прокату, геозони (дозволені, заборонені для паркування, бонусні) та маршрути руху. Було реалізовано функціонал реєстрації, авторизації та верифікації користувачів. Для оренди самокатів інтегровано платіжну систему PayPal. Також користувачі мають змогу переглядати історію

поїздок, стан рахунку та накопичені бонуси.

Адміністративну панель розроблено з використанням ReactJS та фреймворку Bootstrap, що забезпечило зручний, адаптивний інтерфейс і динамічну взаємодію з сервером. Панель дозволяє моніторити парк самокатів у реальному часі, переглядати аналітику за поїздками та зонами, керувати геозонами (додавати, редагувати зони заборони паркування або руху), а також працювати зі звітами про несправності та історією обслуговування техніки. Додатково реалізовано управління верифікацією користувачів.

4 ВИСНОВКИ

У результаті виконаної роботи було розроблено повнофункціональну систему для управління парком електросамокатів, що включає кросплатформений мобільний застосунок на Flutter для користувачів, адміністративну панель для операторів React, серверну частину на базі NestJS та Prisma, а також інтегровану платіжну систему PayPal і Mapbox для навігації. Уся система успішно розгорнута в Docker-середовищі, що забезпечує її стабільну роботу, масштабованість та зручність в обслуговуванні.

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

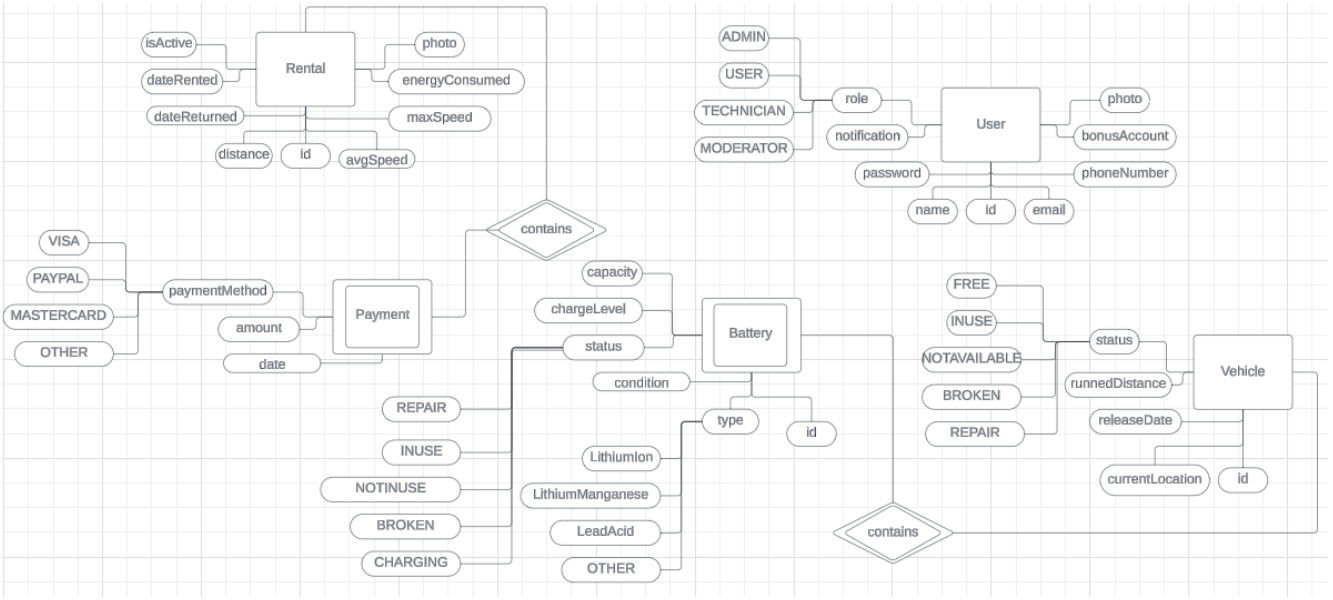


Рисунок Б.1 — ER-діаграма даних (нотація Чена)

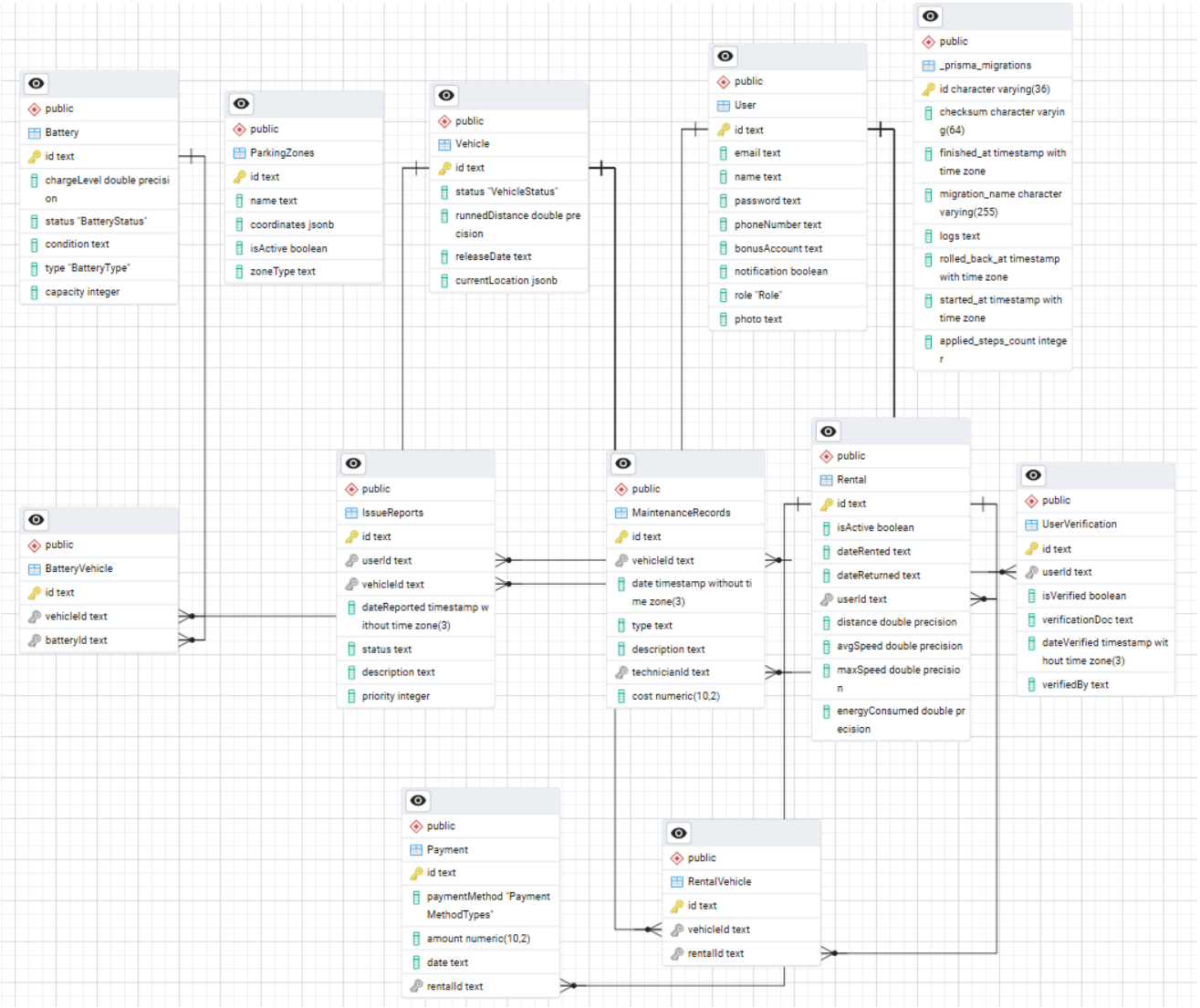


Рисунок Б.2 — Структура бази даних

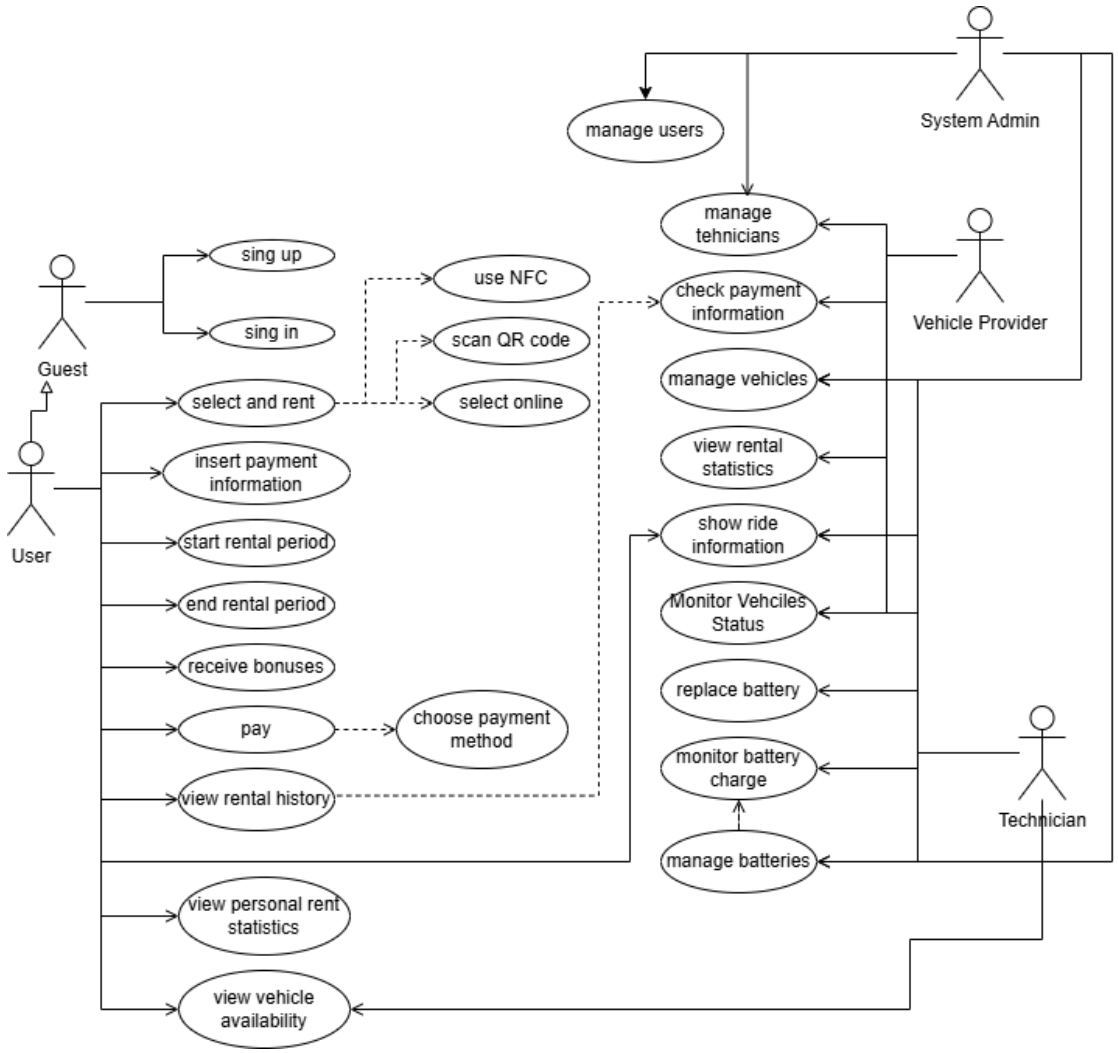


Рисунок Б.3 — UML-діаграма прецедентів

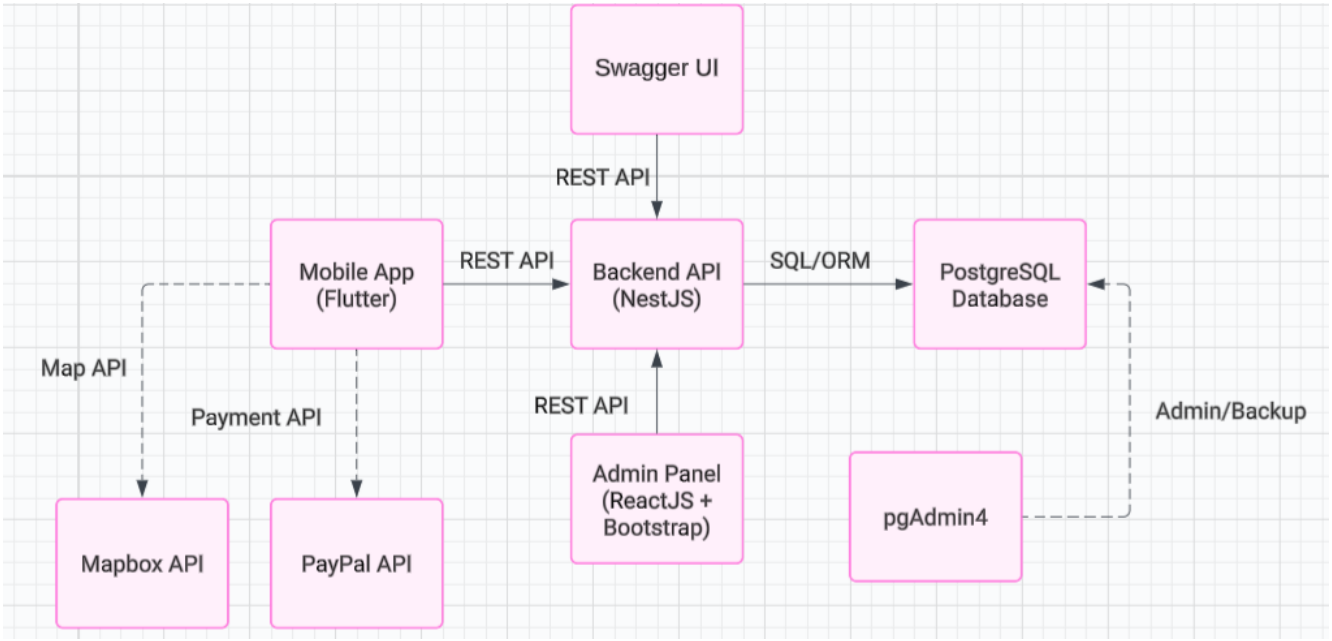


Рисунок Б.4 — UML-діаграма компонентів

Рисунок Б.5 – UML діаграма діяльності (оренда самоката)

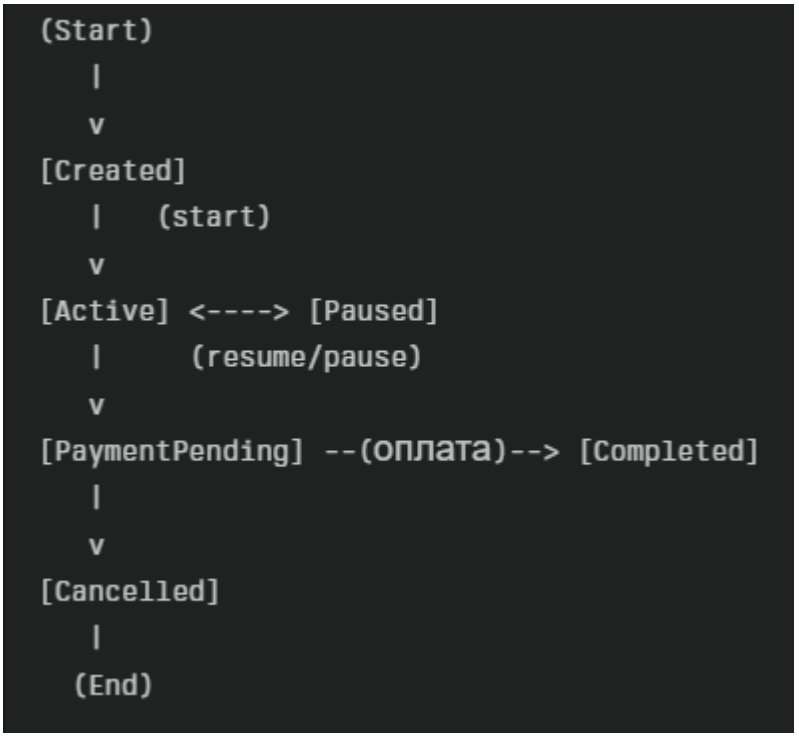


Рисунок Б.6 – UML діаграма станів (оренда самоката)

ДОДАТОК В

Фрагменти коду серверу

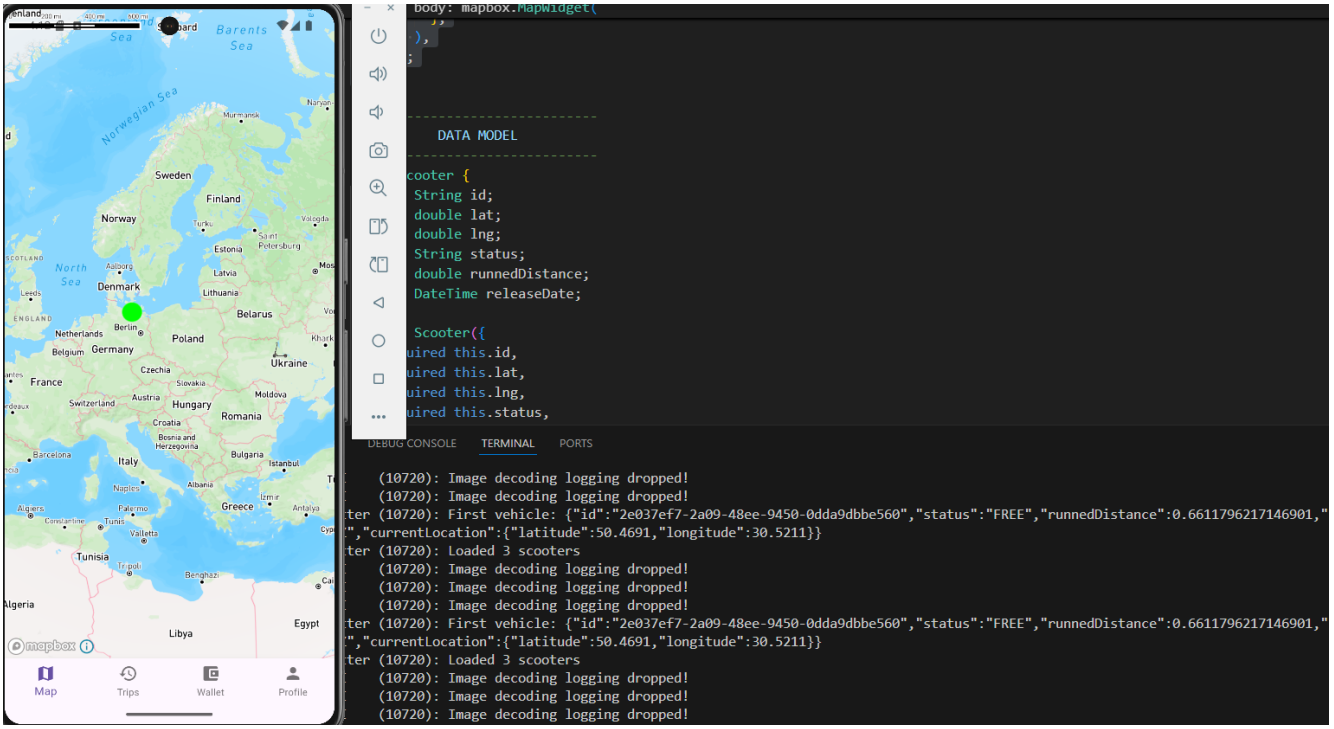
В.1 Математична обробка прикладних даних

GitHub репозиторій: https://github.com/NureVeriasovVladyslav/apz-pzpi-22-2-veriasov-vladyslav/blob/main/Lab4/apz-pzpi-22-2-veriasov-vladyslav-lab4/etr_app/lib/main.dart

Лістинг 1 – Функції що отримують з серверу елементами математичної обробки інформації

```
    /// -------- маркери самокатів --------
      Future<void> _drawScooterMarkers(List<Scooter> list) async {
        _scooterManager ??=
            await
_map!.annotations.createPointAnnotationManager();

        await _scooterManager!.deleteAll();

        final icon = await
_loadBytes('assets/images/scooter.png');
        for (final s in list) {
          await _scooterManager!.create(
            mapbox.PointAnnotationOptions(
              geometry: mapbox.Point(
                  coordinates: mapbox.Position(s.lng, s.lat)),
              image: icon,
              iconSize: 0.12,
              // textField: s.battery != null ? '${s.battery}%' :
null,
              textSize: 12,
              textOffset: [0.0, 1.5],
            ),
          );
        }
```

```
  }

  Future<Uint8List> _loadBytes(String path) async {
    final bytes = await rootBundle.load(path);
    return bytes.buffer.asUint8List();
  }

  /// -------- UI --------
  @override
  Widget build(BuildContext context) => Scaffold(
        body: mapbox.MapWidget(
          key: const ValueKey('map'),
          cameraOptions: mapbox.CameraOptions(
            center: mapbox.Point(
                coordinates: mapbox.Position(0.0, 0.0)), //
стартова точка
            zoom: 1,
          ),
          onMapCreated: (controller) {
            _map = controller;
            _initLocationAndCamera();
            _fetchAndShowScooters();
          },
        ),
      );
}
```



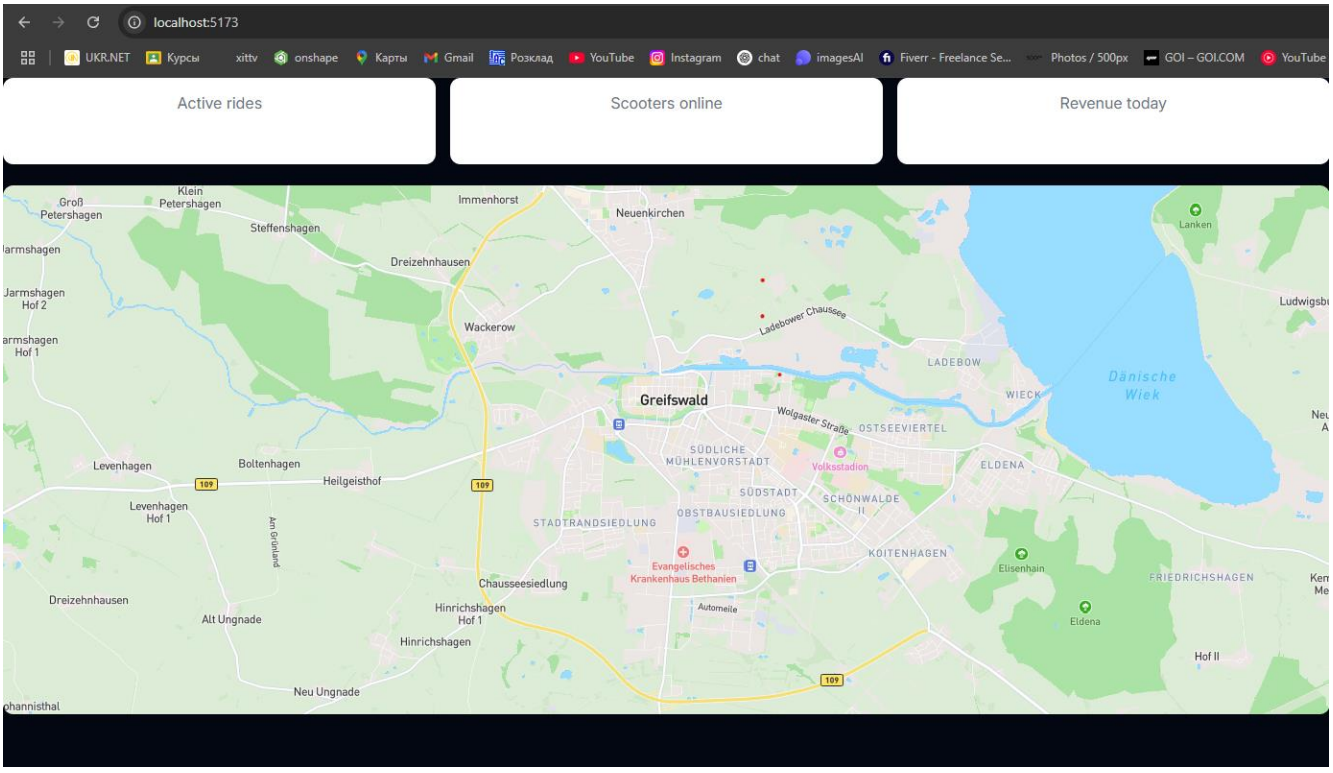


Рисунок В.2 – Отримання координат та місцезнаходження

В.2 Адміністрування бізнес-логіки системи

GitHub репозиторій: https://github.com/NureVeriasovVladyslav/apz-pzpi-22-2-veriasov-vladyslav/blob/main/Lab4/apz-pzpi-22-2-veriasov-vladyslav-lab4/etr_app/lib/main.dart

Лістинг 2 – Прорисовка карти та інших сторінок

```
import 'dart:async';
import 'dart:convert';
import 'dart:typed_data';

import 'package:flutter/material.dart';
import 'package:flutter/services.dart' show rootBundle;
import 'package:http/http.dart' as http;
import 'package:geolocator/geolocator.dart' as geo;
import 'package:mapbox_maps_flutter/mapbox_maps_flutter.dart' as
mapbox;

void main() async {
  WidgetsFlutterBinding.ensureInitialized();

  // Mapbox access-token передається через --dart-define
  const ACCESS_TOKEN = String.fromEnvironment('ACCESS_TOKEN');
  mapbox.MapboxOptions.setAccessToken(ACCESS_TOKEN);

  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) => const MaterialApp(
        debugShowCheckedModeBanner: false,
        home: MainNavigation(),
      );
}

/// ---------------------------
///      BOTTOM NAVIGATION
/// ---------------------------
class MainNavigation extends StatefulWidget {
  const MainNavigation({super.key});
  @override
  State<MainNavigation> createState() => _MainNavigationState();
}

class _MainNavigationState extends State<MainNavigation> {
  int _current = 0;

  final pages = const [
    MapPage(),
    RidesPage(),
    WalletPage(),
    ProfilePage(),
  ];

  @override
```

```
  Widget build(BuildContext context) => Scaffold(
        body: pages[_current],
        bottomNavigationBar: BottomNavigationBar(
          type: BottomNavigationBarType.fixed,
          currentIndex: _current,
          onTap: (i) => setState(() => _current = i),
          items: const [
            BottomNavigationBarItem(icon: Icon(Icons.map), label:
'Map'),
            BottomNavigationBarItem(icon: Icon(Icons.history),
label: 'Trips'),
            BottomNavigationBarItem(
                icon: Icon(Icons.account_balance_wallet), label:
'Wallet'),
            BottomNavigationBarItem(icon: Icon(Icons.person), label:
'Profile'),
          ],
        ),
      );
}

/// ----------------------------
///            MAP
/// ----------------------------
class MapPage extends StatefulWidget {
  const MapPage({super.key});
  @override
  State<MapPage> createState() => _MapPageState();
}

class _MapPageState extends State<MapPage> {
  mapbox.MapboxMap? _map;

  mapbox.PointAnnotationManager? _scooterManager; // усі самокати
  mapbox.PointAnnotationManager? _userManager;    // тільки ваш
маркер

  Timer? _ticker;

  @override
  void initState() {
    super.initState();
    // опитуємо бекенд кожні 15 с
    _ticker = Timer.periodic(const Duration(seconds: 15), (_) {
      _fetchAndShowScooters();
    });
  }

  @override
  void dispose() {
    _ticker?.cancel();
    super.dispose();
  }
```

```
  /// -------- геолокація + маркер користувача --------
  Future<void> _initLocationAndCamera() async {
    // запит дозволу
    final perm = await geo.Geolocator.requestPermission();
    if (perm == geo.LocationPermission.denied ||
        perm == geo.LocationPermission.deniedForever) {
      return;
    }

    // координати
    final pos = await geo.Geolocator.getCurrentPosition();
    final point =
        mapbox.Point(coordinates: mapbox.Position(pos.longitude,
pos.latitude));

    // центруємо карту
    await _map?.flyTo(
      mapbox.CameraOptions(center: point, zoom: 2),
      mapbox.MapAnimationOptions(duration: 1500),
    );

    // показуємо маркер
    _userManager ??=
        await _map!.annotations.createPointAnnotationManager();

    await _userManager!.deleteAll(); // перезапис

    final icon = await _loadBytes('assets/images/pointer.png');
    await _userManager!.create(
      mapbox.PointAnnotationOptions(
        geometry: point,
        image: icon,
        iconSize: 0.15,
      ),
    );
  }
  Future<String?> getToken() async {
  try {
    const base = String.fromEnvironment('BACKEND_URL',
      defaultValue: 'http://10.0.2.2:3000');
    final response = await http.post(
      Uri.parse('$base/auth/login'),
      headers: {'Content-Type': 'application/json'},
      body: json.encode({
        'email': 'admin@gmail.com',
        'password': 'admin@gmail.com'
      }),
    );

    if (response.statusCode == 200) {
      final data = json.decode(response.body);
      return data['token'];
```

```
    }
  } catch (e) {
    debugPrint('Auth error: $e');
  }
  return null;
}

  /// -------- запит самокатів --------
  Future<void> _fetchAndShowScooters() async {
  if (_map == null) return;

  const base = String.fromEnvironment('BACKEND_URL',
      defaultValue: 'http://10.0.2.2:3000');
  final url = Uri.parse('$base/vehicle');

  // final token = await getToken(); // якщо це async

  const token =

'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6ImFkbWluQGdtYWlsLm
NvbSIsInJvbGUiOiJBRE1JTiIsImlhdCI6MTc0OTU3MDc4NiwiZXhwIjoxNzQ5NTc0Mz
g2fQ.j5rcyeWxgtxFARpurBc44-JcbCuUDnPJuvg00AWjmk0';

  try {
    final res = await http.get(
      url,
      headers: {'Authorization': 'Bearer $token'},
    );

    if (res.statusCode != 200) {
      debugPrint('Vehicle HTTP ${res.statusCode}: ${res.body}');
      return;
    }

    final List<dynamic> raw = jsonDecode(res.body) as List<dynamic>;

    // Додай логування для розуміння структури даних
    debugPrint('First vehicle: ${raw.isNotEmpty ?
json.encode(raw.first) : "none"}');

    final List<Scooter> scooters = raw
        .where((e) => e['currentLocation'] != null) // Фільтруємо
тільки з координатами
        .map((e) => Scooter.fromJson(e as Map<String, dynamic>))
        .toList(growable: false);

    debugPrint('Loaded ${scooters.length} scooters');

    await _drawScooterMarkers(scooters);
  } catch (e, stackTrace) {
    debugPrint('Vehicle request error: $e');
    debugPrint('Stack trace: $stackTrace');
  }
```

```
}
```

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"]
```