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

Лабораторна робота №5
з дисципліни: «Архітектура програмного забезпечення»
на тему: «РОЗРОБКА ПРОГРАМНОЇ АРХІТЕКТУРИ ТА СТВОРЕННЯ І ВІДЛАГОДЖЕННЯ ПРОГРАМНОГО КОДУ ПРИСТРОЮ ІНТЕРНЕТУ РЕЧЕЙ (IOT) АБО РОЗУМНОГО ПРИСТРОЯ (SMART DEVICE) 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 | 01.06.2025 | 0.1 | Створено усі розділи |

# 2 ЗАВДАННЯ

Метою роботи є є розробити програмне забезпечення для IoT або SmartDevice пристрою, реалізованого на базі будь-якої поширеної на сьогодні платформи, придатної для реалізації вбудованих систем (Embedded System). А також демонстрація працездатності повної 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.

Серверна частина та бізнес-логіка реалізована на базі фреймворку NestJS, що забезпечує модульність, масштабованість і підтримку сучасних стандартів розробки. Для автоматичної документації API інтегровано Swagger, що дозволяє генерувати інтерактивну документацію з можливістю тестування ендпоінтів безпосередньо через веб-інтерфейс. Серверна частина забезпечує аутентифікацію через JWT-токени, шифрування паролів, розмежування доступу відповідно до ролей користувачів (клієнт, адміністратор, технічний персонал), а також повний набір CRUD-операцій для всіх сутностей системи.

Система управління базою даних побудована на PostgreSQL з використанням ORM Prisma, що забезпечує типобезпечність, міграції схеми та зручну роботу з реляційними даними. Вся інфраструктура розгорнута у Docker-контейнерах, що спрощує розгортання, масштабування та підтримку середовища розробки. Для адміністрування бази даних використовується pgAdmin4, який також працює у контейнері та надає веб-інтерфейс для управління даними, створення резервних копій та моніторингу продуктивності системи.

IoT-компоненти системи реалізовані на базі мікроконтролера ESP-32 з інтеграцією GPS-модуля Neo6M та GSM-модуля SIM800L. Дана система забезпечує відстеження місцезнаходження самокатів у реальному часі, можливість дистанційного блокування/розблокування транспорту та передачу телеметричних даних на сервер через мобільний інтернет. GPS-модуль надає

точні координати з частотою оновлення, достатньою для відстеження руху, а GSM-модуль забезпечує надійний зв'язок із серверною частиною через HTTP API. Пристрій підтримує команди керування через SMS-повідомлення та автоматично передає дані про місцезнаходження, що дозволяє операторам системи здійснювати ефективний моніторинг та управління парком електросамокатів незалежно від наявності Wi-Fi покриття у зоні розташування транспорту.

Така архітектура забезпечує повний цикл управління парком електротранспорту — від фізичного контролю пристроїв через IoT-компоненти до аналітики та бізнес-логіки на серверній частині, що робить систему придатною для комерційного використання у сфері мікромобільності.

# 4 ВИСНОВКИ

У ході виконання роботи було розгорнуто комплексну систему для організації, аналітики та управління парком електросамокатів, яка включає сучасний сервер на базі NestJS із документацією Swagger, реляційну базу даних PostgreSQL під керуванням Prisma у Docker-контейнерах, веб-інтерфейс адміністратора на React із багаторівневою навігацією та інтерактивною аналітикою, мобільний застосунок на Flutter з інтеграцією Mapbox та платіжної системи PayPal, а також IoT-компонент для відстеження та керування самокатами в реальному часі. Така архітектура забезпечує надійний збір, обробку й візуалізацію даних, дозволяє гнучко масштабувати систему та легко підтримувати її у виробничих умовах, а також гарантує зручність і безпеку для всіх категорій користувачів.

# ДОДАТОК А
Відеозапис

Відеозапис презентації результатів лабораторної роботи:

https://youtu.be/_yOMgbINmCI

Хронологічний опис роботи:

00:00 - початок, Docker  та бд PostgreSql

00:50 – сервер та Swagger

02:40 – веб застосунок для адміністрування

03:18 – мобільний застосунок для користувачів

04:20 – сервер

# ДОДАТОК Б
Графічні матеріали

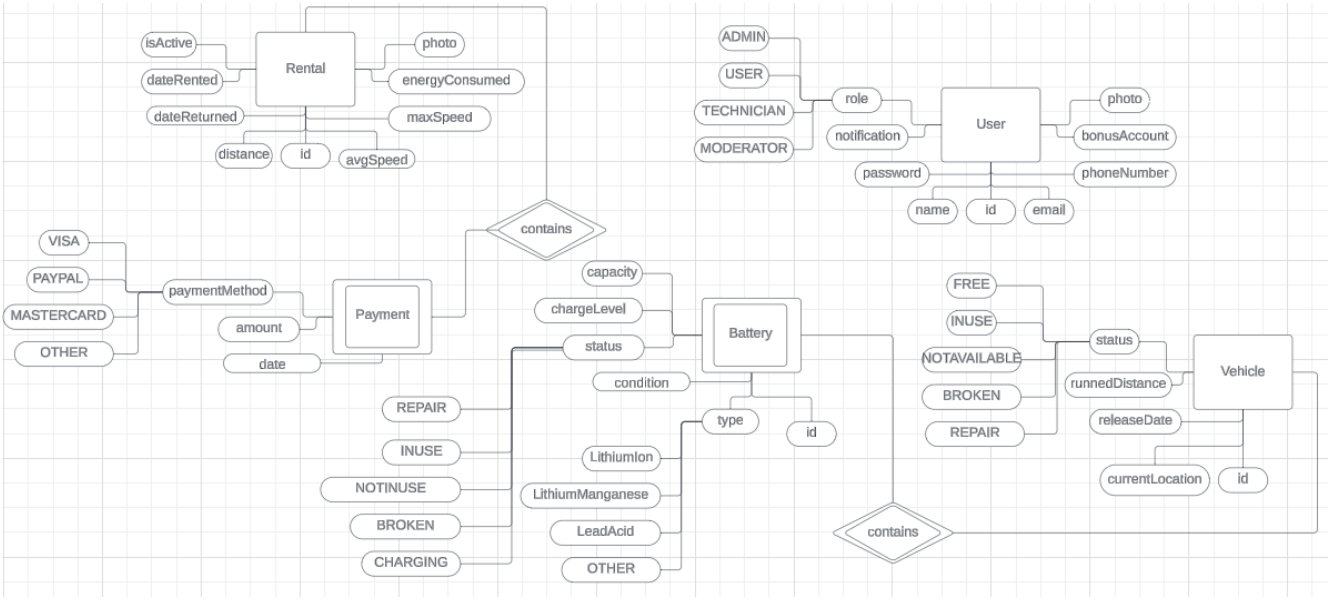


Рисунок Б.1 — ER-діаграма даних (нотація Чена)

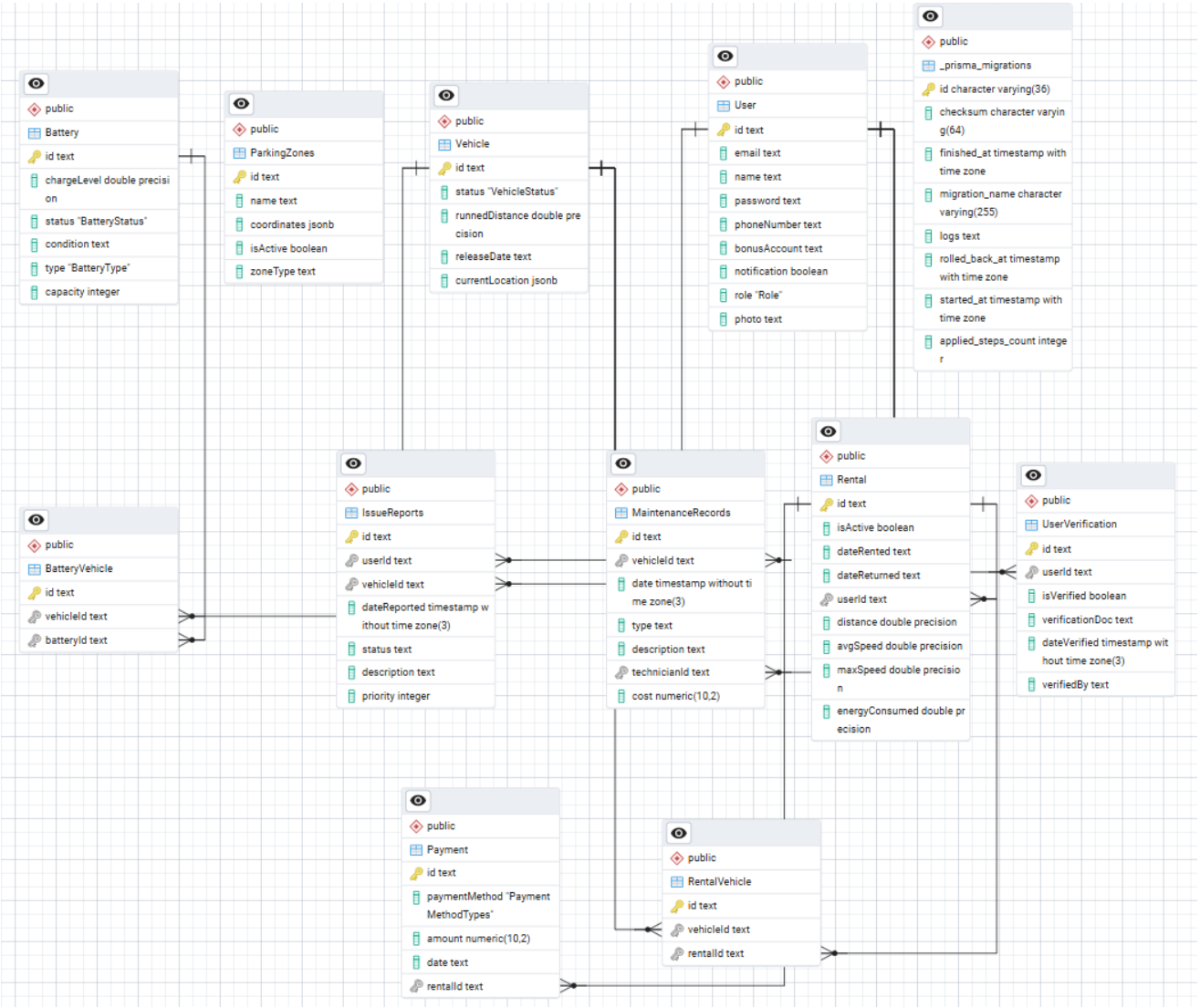


Рисунок Б.2 — Структура бази даних

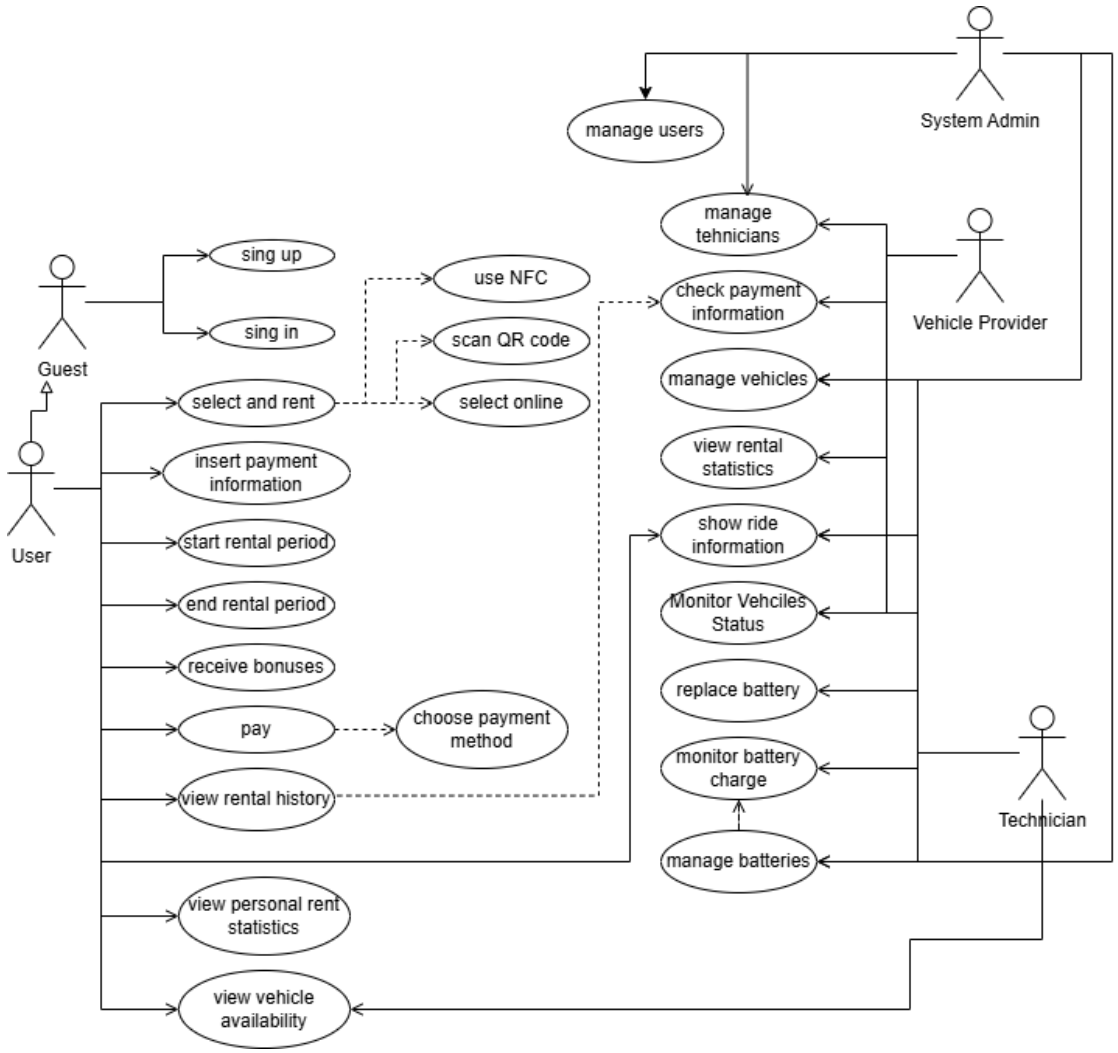


Рисунок Б.3 — UML-діаграма прецедентів

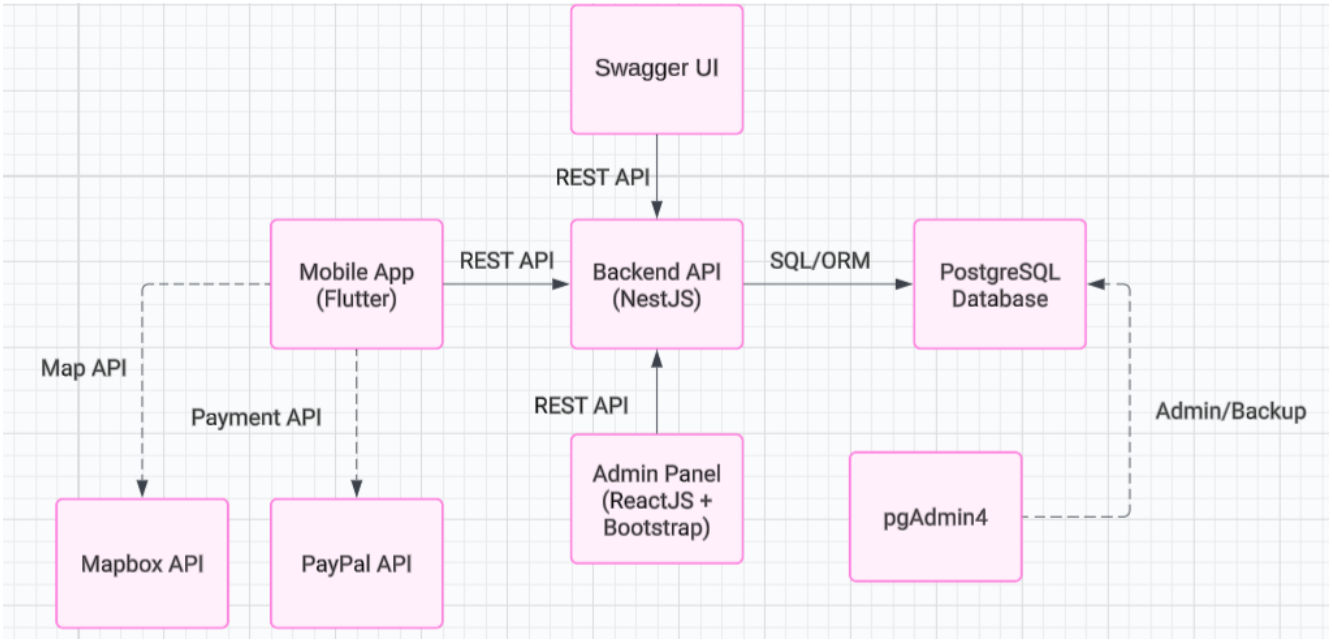


Рисунок Б.4 — UML-діаграма компонентів


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

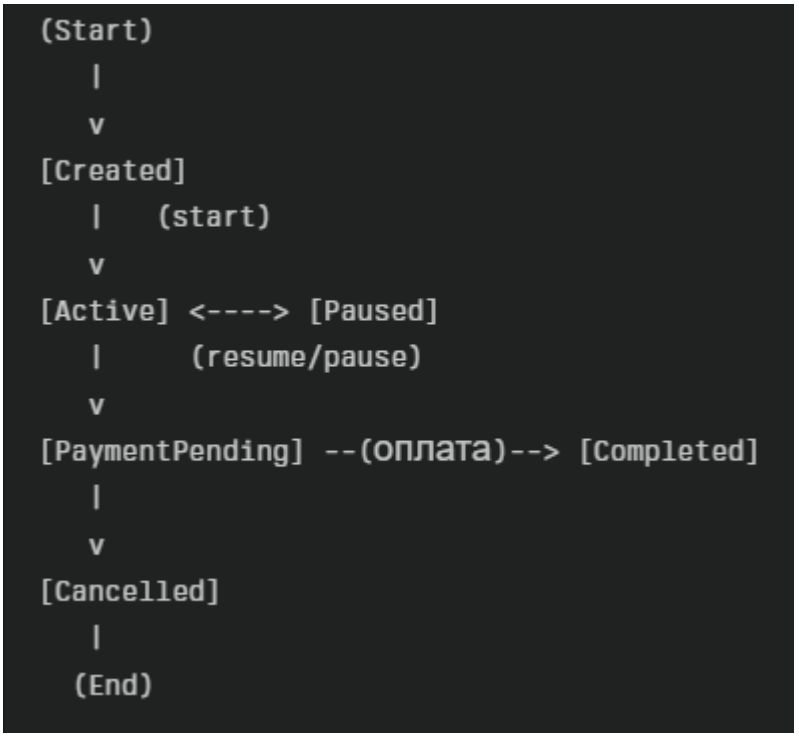


Рисунок Б.6 – UML діаграма станів (оренда самоката)

# ДОДАТОК В

Фрагменти коду

## В.1 Математична обробка прикладних даних

GitHub репозиторій: https://github.com/NureVeriasovVladyslav/apz-pzpi-22-2-veriasov-vladyslav/blob/main/Lab5/apz-pzpi-22-2-veriasov-vladyslav-lab5/apz-pzpi-22-2-veriasov-vladyslav-lab5-code/IoT/main.cpp

Лістинг 1 – Функції IoT

```
#include <TinyGsmClient.h>
#include <HTTPClient.h>
#include <SoftwareSerial.h>
#include <ArduinoJson.h>

#define LOCATION_RX 16
#define LOCATION_TX 17
#define MODEM_RX 18
#define MODEM_TX 19
#define GPS_BAUD 9600

HardwareSerial gpsSerial(2); //Ports 16, 17
TinyGPSPlus gps;

SoftwareSerial SerialAT(MODEM_RX, MODEM_TX);
TinyGsm modem(SerialAT);
TinyGsmClient client(modem);
HTTPClient http;

void getLocation(double &lat, double &lon) {
  if(gpsSerial.available()) {
    char gpsData = gpsSerial.read();
    gps.encode(gpsData);
    if (gps.location.isValid()) {
      lat = gps.location.lat();
      lon = gps.location.lng();
    }
  }
}

void sendLocationToServer() {
  if (!modem.gprsConnect("apn", "user", "pass")) {
    Serial.println("GPRS fail");
    return;
  }

  http.begin(client, "http://server.com/api/location");
  http.addHeader("Content-Type", "application/json");
```

```
  double latitude 0.0;
  double longitude = 0.0;
  getLocation(latitude, longitude);

  StaticJsonDocument<200> doc;
  doc["latitude"] = coords.first;
  doc["longitude"] = coords.second;

  String json;
  serializeJson(doc, json);
    int httpResponseCode = http.POST(json);

  Serial.println(httpResponseCode);
  http.end();
}

void lockScooter() {
  //Here must be the calling of specific scooter commands.
}

void unlockScooter() {
  //Here must be the calling of specific scooter commands.
}

void setup() {
  Serial.begin(115200);
  SerialAT.begin(9600);
  modem.restart();
  modem.sendAT("+CMGF=1");
  modem.sendAT("+CNMI=1,2,0,0,0");

  gpsSerial.begin(GPS_BAUD, SERIAL_8N1, LOCATION_RX, LOCATION_TX);
}

void loop() {
  if (SerialAT.available()) {
    String sms = SerialAT.readString();
    if (sms.indexOf("GET_LOCATION") != -1) {
      sendLocationToServer();
    } else if (sms.indexOf("LOCK_SCOOTER") != -1) {
      lockScooter();
    } else if (sms.indexOf("UNLOCK_SCOOTER") != -1) {
      unlockScooter();
    }
  }
}
```

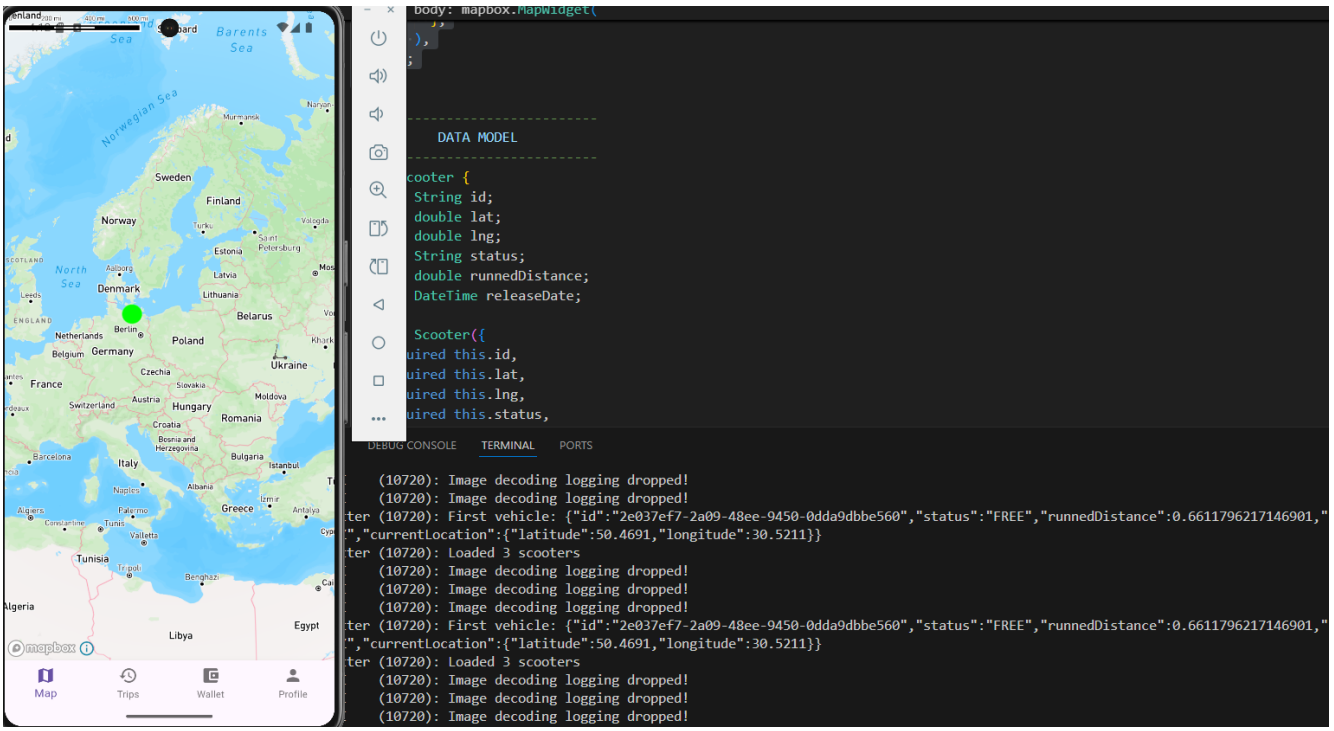


Рисунок В.2 – Отримання координат та місцезнаходження

## В.2 Адміністрування бізнес-логіки системи

GitHub репозиторій: https://github.com/NureVeriasovVladyslav/apz-pzpi-22-2-veriasov-vladyslav/blob/main/Lab5/apz-pzpi-22-2-veriasov-vladyslav-lab5/apz-pzpi-22-2-veriasov-vladyslav-lab5-code/mobile/etr_app/lib/main.dart

Лістинг 2 – Прорисовка карти та інших сторінок

```
/// -------- 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.eyJlbWFpbCI6ImFkbWluQGdt
```

```
YWlsLmNvbSIsInJvbGUiOiJBRE1JTiIsImlhdCI6MTc0OTU3MDc4NiwiZXhwIjoxNzQ5
NTc0Mzg2fQ.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>;

    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/Lab5/apz-pzpi-22-2-veriasov-vladyslav-lab5/apz-pzpi-22-2-veriasov-vladyslav-lab5-code/server/apz-pzpi-22-2-veriasov-vladyslav-lab5-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"]
```