

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук та технологій

(повне найменування факультету)

Кафедра комп'ютерних систем та мереж

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавра

(ступінь вищої освіти (освітній ступінь))

на тему **РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ ПОШУКУ**
ЗАПРАВOK ЕЛЕКТРОКАРІВ

Виконав: студент 4 курсу, групи КНТ-511сп
спеціальності _____

123 «Комп'ютерна інженерія»

(код і найменування спеціальності)

Освітня програма (спеціалізація) _____

«Комп'ютерна інженерія»

ШКОДА Д.С.

(ПРИЗВИЩЕ та ініціали)

Керівник ГОЛУБ Т.М.

(ПРИЗВИЩЕ та ініціали)

Рецензент МАЛИЙ О.Ю.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування вищого навчального закладу)

Факультет Комп'ютерних наук і технологій
Кафедра «Комп'ютерні системи та мережі»
Ступінь вищої освіти бакалаврський
Спеціальність 123 Комп'ютерна інженерія
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерна інженерія
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ
Завідувач кафедри КУДЕРМЕТОВ Р.К.
“ ” 2024 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

ШКОДИ Данііла Сергійовича
(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проекту (роботи) Розробка мобільного застосунку для пошуку заправок для електрокарів

керівник проекту (роботи) к. т. н., доцент, ГОЛУБ Тетяна Василівна
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом вищого навчального закладу від “01” квітня 2024 року
№ 105

2. Строк подання студентом проекту (роботи) 26 травня 2024 року

3. Вихідні дані до проекту (роботи) дослідити процес розробки мобільних застосунків під різні мобільні системи. Дослідити існуючі програмні продукти та сформулювати вимоги до власного програмного продукту.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1) Аналіз предметної області;

2) Дослідження технологій створення мобільних застосунків;

3) Розробка програмної частини застосунку.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

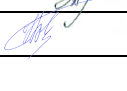
Структурна схема реалізації проекту;

Структура бази даних та серверу;

Інтерфейс мобільного застосунку;

Алгоритм роботи мобільного застосунку.

6. Консультанти розділів проекту (роботи)

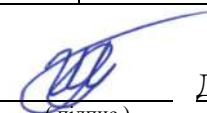
Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4	ГОЛУБ Т.В., к.т.н., доцент		
нормоконтроль	ЩЕРБАК Н.В., ст. викл.		

7. Дата видачі завдання 15.04.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Отримання завдання на дипломну роботу та побудова плану-графіку виконання робіт	15.04.2024 р.	
2	Опис та аналіз предметної області.	18.04.2024 р.	
3	Огляд існуючих аналогів мобільних застосунків	22.04.2024 р.	
4	Постановка завдання розробки та дослідження.	26.04.2024 р.	
5	Розробка програмної частини застосунку.	28.04.2024 р.	
6	Оформлення отриманих результатів у пояснювальній записці	18.05.2024 р.	
7	Оформлення графічного матеріалу	23.05.2024 р.	
8	Створення презентації, доповіді та підготовка до захисту дипломної роботи	26.05.2024 р.	

Студент (ка)


(підпис)

Даніїл ШКОДА
(Ім'я ПРІЗВИЩЕ)

Керівник проекту (роботи)


(підпис)

Тетяна ГОЛУБ
(Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

ПЗ: 89 с., 16 рис., 58 лістингів, 2 додатка, 18 джерел.

МОБІЛЬНИЙ ЗАСТОСУНОК, ЕЛЕКТРОКАР, ЗАПРАВКА, ПРОГРАМНІ ЗАСОБИ, МЕРЕЖА, БАЗА ДАНИХ, КЛІЄНТ-СЕРВЕР, АРХІТЕКТУРА, МОВА ПРОГРАМУВАННЯ

Об'єкт розробки – мобільний застосунок для пошуку заправок для електрокарів.

Метою роботи є розробка програмного засобу для ОС Android, тобто мобільного застосунку для пошуку заправок для електрокарів. Застосунок призначений для швидкого знаходження заправок для електромобілів за допомогою мапи електрозаправок.

Проект складається з трьох розділів.

Перший розділ присвячено проведенню аналізу предметної області. Проведено аналіз мобільних застосунків з подібним функціоналом, здійснено порівняльну характеристику, виявлено переваги та недоліки.

У другому розділі розкриті питання щодо поняття мобільних застосунків, розробка та їх створення. Сформована постановка завдання, вимоги та задачі до об'єкту розробки. Обрано середовище розробки, програмні засоби для реалізації проекту.

Третій розділ присвячено розробці програмної частини мобільного застосунку, а саме розробка архітектури застосунку та клієнт-сервера.

В результаті роботи був реалізований мобільний застосунок для пошуку заправок електрокарів. Реалізований застосунок можна використовувати для вирішення особистих задач при подорожуванні, так і в комерційних цілях.

ЗМІСТ

Скорочення та умовні позначки	6
Вступ.....	7
1 Аналіз предметної області.....	9
1.1 Опис предметної області	9
1.2 Огляд існуючих аналогів.....	10
2 Теоретична частина.....	15
2.1 Мобільний застосунок	15
2.2 Розробка та створення мобільних застосунків.....	16
2.3 Постановка завдання розробки та дослідження.....	23
3 Розробка програмної частини застосунку	23
3.1 Розробка архітектури застосунку	23
3.2 Розробка архітектури системи (клієнт-сервер).....	56
Висновки	72
Перелік джерел посилання	73
Додаток А Код головного класу FragmentMap	75
Додаток Б Фрагмент коду класу CommentsAdapter.....	86

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

СКБД – Система Керування Базою Даних

API – Application Programming Interface

IDE – Integrated Development Environment

XML – Extensible Markup Language

Android – операційна система

iOS – операційна система

Java – мова програмування

C# – мова програмування

ОСРР – європейський протокол

HTML – мова розмітки

Android Studio – інтегроване середовище розробки

QR – код

SQLite – реляційна база даних

IntelliJ IDEA – інтегроване середовище розробки

CRUD – функції управління даними

Google Maps – картографічний вебсервіс

ORM – Object-Relational Mapping

ВСТУП

Електрокар – це електричний транспортний засіб, який пересувається за допомогою електродвигуна, живлення якого забезпечує акумуляторна батарея.

У багатьох країнах світу електричними транспортними засобами користується понад 30 % користувачів. Великі автовиробники починають відмовлятися від виробництва автомобілів з бензиновими та дизельними двигунами, а переходять на випуск електричних транспортних засобів [1] .

У 2023 році в Україні більше 10 % користувачів автомобілем обрали саме електромобіль.

Зі зростанням кількості електромобілів одночасно збільшується і кількість зарядних станцій. Станом на серпень 2023 було відкрито 4786 зарядних станцій. Співвідношення станцій до електромобілів становить на 1 станцію приходить 15 атомобілей. Таким чином зарядна інфраструктура не справляється з навантаженням і водії вимушені витратити більше часу на заряджання своїх електромобілей.[2].

Найбільш поширеною проблемою власників електромобілів є тривалість подорожі на одній заправці. Адже є хвилювання, що автомобіль розрядиться посеред поїздки там, де не буде доступу до зарядної станції.

Для вирішення зазначених проблем зручно користуватися мобільними застосунками.

Мобільні застосунки становлять невід'ємну складову сучасного цифрового життя кожної людини хто користується гаджетами. З кожним днем вони стають все більш необхідними для розваг, навчання, роботи, спілкування та отримання послуг.

Вони відіграють критичну роль у полегшенні нашого щоденного життя, забезпечують можливість отримання необхідної інформації або виконання завдань у будь-який час і в будь-якому місці, де є доступ до мережі Інтернет або навіть у режимі офлайн. Вони дозволяють користувачам налаштовувати свій

досвід відповідно до їхніх потреб і уподобань, надаючи персоналізовані рекомендації та функції. Багато мобільних додатків допомагають у збереженні часу та зусиль, шляхом автоматизації процесів, нагадування про важливі події або навіть виконання рутинних завдань.

Ці переваги роблять мобільні застосунки незамінною частиною нашого цифрового життя, допомагаючи нам бути більш продуктивними, організованими та підключеними в будь-який момент і в будь-якому місці.

Сучасний мобільний застосунок повинен бути не лише функціональним, але й зручним, безпечним та ефективним інструментом для досягнення поставлених цілей користувача [3].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Із щоденним збільшенням інтересу до альтернативної енергетики, у цій сфері відбуваються активні розвиток, інновації та виробництво. На сучасних дорогах електромобілі вже не викликають такого захвату, як раніше, стаючи стандартом як для великих міст, так і для маленьких провінційних населених пунктів. Запит на екологічно чистий транспорт викликає відповідь у вигляді будівництва заправних станцій та техобслуговування таких авто. Екологічні автомобілі допомагають зменшити негативний вплив на довкілля, переважною мірою електромобілі працюють на електричній енергії, а гібридні транспортні засоби поєднують в собі двигуни внутрішнього згоряння та електродвигуни.

Постійне зростання попиту на електромобілі в різних містах України –це сигнал до того, що зовсім скоро заправки для такого транспорту стануть також поширені, як і газові та бензинові заправки [4].

Для водіїв електрокарів одна з основних функцій, необхідна в подібних мобільних додатках – це швидко та зручно знаходити зарядні заправки для електромобілів за допомогою мапи електрозаправок, адже будучи за кермом електромобіля набагато простіше вибудовувати маршрут до зарядної заправки, бронювати її для зарядної сесії в мобільному телефоні. Та й оплатити зарядну сесію в Особистому кабінеті мобільного додатка не складає труднощів.

На картах мобільних додатків також розміщені торгівельні центри, ресторани, готелі та стоянки. Таким чином, паралельно з резервацією зарядки власник електромобіля може забронювати готельний номер або столик в кафе.

Мобільні застосунки для зарядки електромобілів допомагають не тільки спростити процес зарядки електромобілів, але і вирішити бізнес–завдання для власників електрозарядних станцій і їх мережевого рішення.

1.2 Огляд існуючих аналогів

На ринку мобільних додатків на даний момент вже існують рішення, які виконують відповідні функції. Розглянемо деякі з них.

Мобільний застосунок «GO TO-U».

Мобільний застосунок GO TO-U дозволяє швидко та зручно знаходити зарядні заправки для електромобілів (за допомогою мапи чи пошукового вікна в застосунку) і резервувати їх на потрібний час у декілька кліків.

На карті застосунку GO TO-U є ресторани, кафе, кінотеатри та навіть торговельні центри, стоянки яких оснащені чарджерами. GO TO-U працює в режимі 24/7 та постійно оновлюється.

Застосунок налічує кілька сотень точок заряджання по всій Україні – на Заході, Сході, Півночі, Півдні. GO TO-U (рис. 1.1) представлений у 47 країнах, серед них: Велика Британія, Німеччина, Франція, Фінляндія, Іспанія, Швейцарія, Польща, Латвія, Таїланд та багато інших [5] .

Створений для Android та iOS та містить такі функції:

- переглядати актуальну інформацію про зарядні станції, їх розташування та відгуки від інших користувачів;
- знаходити найвигідніші платні або безкоштовні зарядні станції;
- резервувати час для зарядки та сервіси біля зарядної станції;
- планувати маршрут та вибирати локації зарядних станцій за різними критеріями;
- отримувати бонуси та подарунки за використання;
- оплачувати зарядження на платних станціях безпосередньо у додатку (доступно на партнерських станціях);
- отримувати сповіщення щодо процесу зарядки, обсягу електроенергії та іншої важливої інформації.

Поряд з перевагами мобільний застосунок має і недоліки, а саме:

- мапи в додатку іноді підвисають;

- фільтри працюють не коректно;
- постійно треба звертатись в службу технічної підтримки.



Рисунок 1.1 – Інтерфейс мобільного застосунку «GO TO-U»

Мобільний застосунок «UGV Chargers».

Мобільний застосунок «UGV_chargers» (рис. 1.2) сервіс для контролю за процесом зарядки електрокарів та здійснення оплати за надані послуги. Дозволяє в лічені секунди знайти зарядну станцію та зарядити батарею електрокару.

Створений для Android та iOS та містить такі функції:

- планування маршруту і навігація до зарядних пунктів;
- докладна інформація про місце, дату та кількість заряджених кВт;
- узагальнена інформація про спожиту електроенергію, час зарядження та загальну вартість;
- можливість безпосередньо оплатити зарядку за допомогою карти Visa чи MasterCard через додаток;
- можливість поповнити віртуальний рахунок з кредитної карти і продовжувати заряджати авто без використання банківських карток;

- отримання сповіщень про стан зарядження;
- отримання допомоги від диспетчерської служби в режимі 24/7 [6].

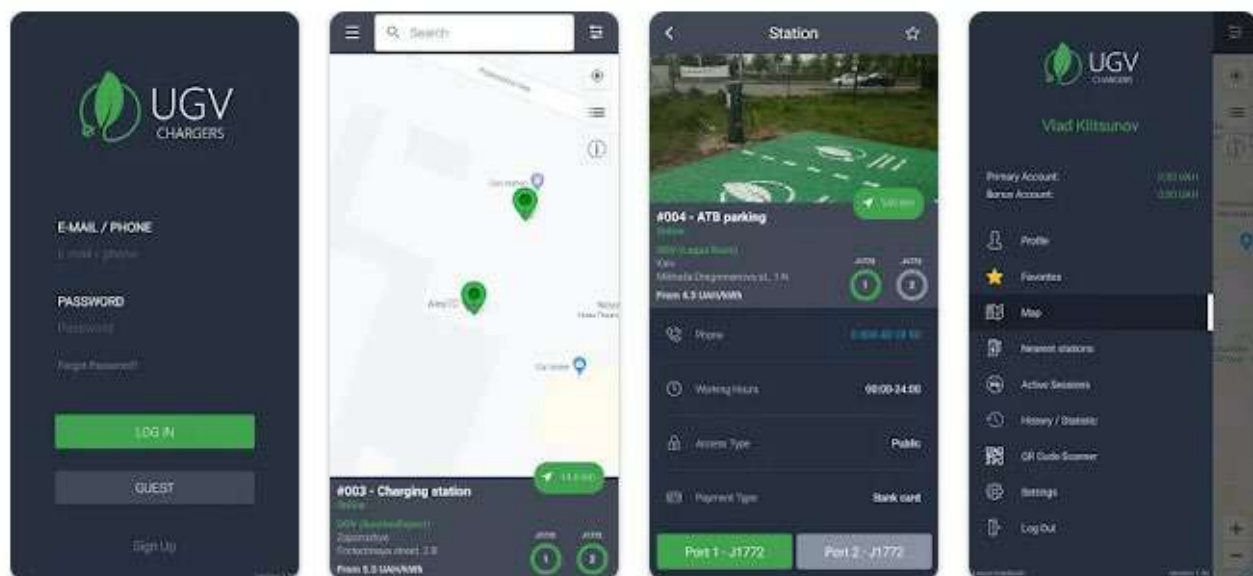


Рисунок 1.2 – Інтерфейс мобільного застосунку «UGV Chargers»

Мобільний застосунок «OnCharge».

Мобільний додаток "OnCharge" (рис. 1.3) працює через протокол OCPP (Open Charge Point Protocol), що використовується для зв'язку між зарядними станціями для електротранспорту та центральною управляючою системою [7]. Доступний для платформ Android та iOS.

"OnCharge" є хмарним сервісом, який діє як посередник між користувачем та оператором. З його допомогою власник електрокару може знаходити найближчі вільні зарядні станції, планувати оптимальний маршрут до них, отримувати інформацію про тарифи та програми лояльності. Додаток також надає можливість переглядати історію транзакцій та статистику заряджання. Легкий вхід до додатку без логіну та паролю, лише за номером телефону, сприяє зручності користувачів. Додаток дозволяє заряджати за допомогою QR-коду та розраховувати в реальному часі, а не після завершення транзакції. Користувач

завжди має можливість переглядати витрати та обсяг електроенергії в режимі реального часу.

Користувач завжди в режимі реального часу бачить, скільки з його рахунку списується коштів, і яка кількість електроенергії він споживає.

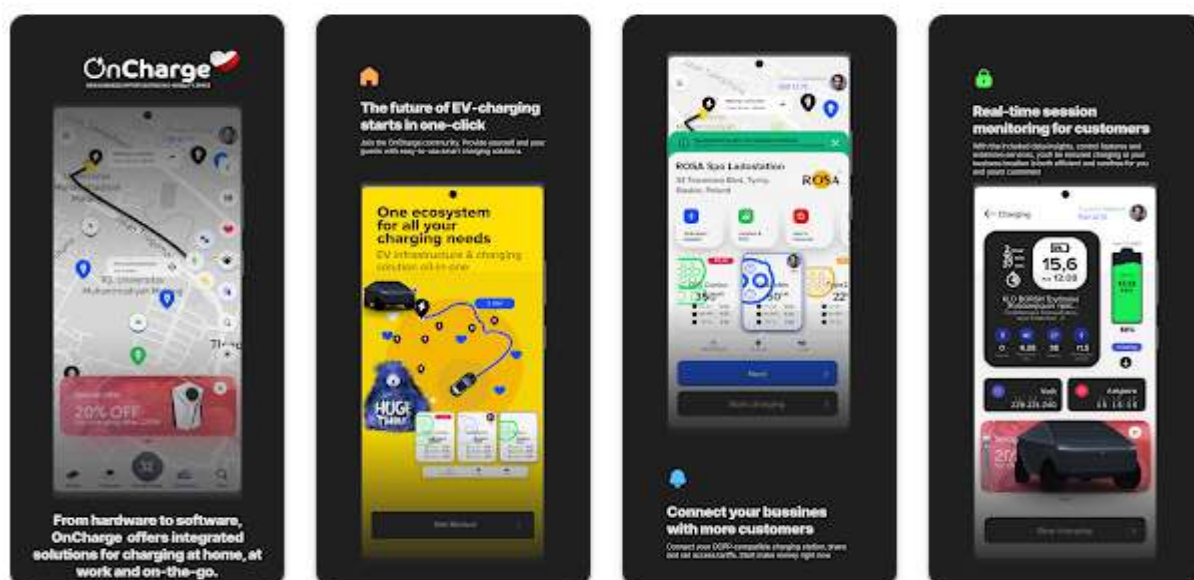


Рисунок 1.3 – Інтерфейс мобільного застосунку «OnCharge»

Мобільний застосунок «ТОКА Network».

Мобільний застосунок ТОКА Network (рис. 1.4) дозволяє знаходити найближчі зарядні станції або планувати маршрут для вашого електрокару. Доступний для платформ Android та iOS.

Основні переваги використання додатку:

- зручний пошук найближчих зарядних станцій мережі ТОКА;
- онлайн статус доступності зарядних станцій;
- можливість використовувати безконтактне включення та відключення зарядних станцій;
- завжди доступна статистика щодо циклів зарядки.

За допомогою застосунку можливо:

- знаходити найближчі зарядні станції або планувати маршрути для електромобіля;
- вибирати конектори за допомогою фільтрів, які підходять вашому авто;
- спостерігати за станом зарядних станцій у реальному часі;
- оплачувати зарядку через додаток;
- контролювати процес заряджання;
- авторизуватися на зарядній станції;
- аналізувати витрати на зарядку [8].

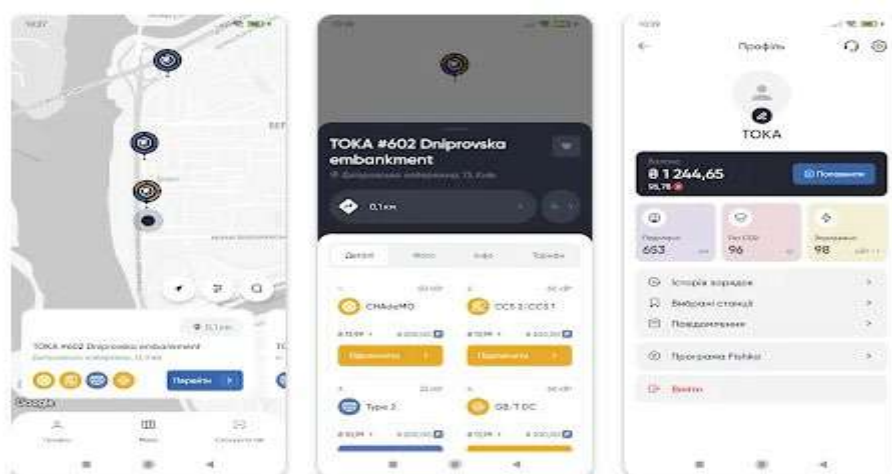


Рисунок 1.4 – Інтерфейс мобільного застосунку «ТОКА Network»

2 ТЕОРЕТИЧНА ЧАСТИНА

2.1 Мобільний застосунок

2.1.1 Поняття мобільного застосунку

Мобільний застосунок представляє собою тип програмного забезпечення, спеціально розроблений для використання на портативних пристроях, таких як смартфони або смарт-годинники.

Ці програми можуть:

- використовувати унікальні функції пристрою (наприклад, отримувати дані від вбудованого сенсора в гаджеті);
- здійснювати фотозйомку або відеозаписи;
- працювати у режимі онлайн або офлайн відповідно до своєї мети.

Для завантаження та встановлення мобільних застосунків використовуються цільові маркетплейси, наприклад, магазини програм App Store і Google Play (колишня назва – Android Market).

Також існують центри поширення мобільного програмного забезпечення у великих виробників гаджетів, наприклад: Huawei AppGallery, BlackBerry World, Samsung Galaxy Store [9].

2.1.2 Можливості мобільних застосунків

Мобільні застосунки надають зручний спосіб користувачам отримати доступ до різноманітних функцій, послуг, онлайн-інформації, веб-ресурсів, соціальних мереж та медіа-контенту. Для бізнесу мобільні додатки полегшують взаємодію з клієнтами, допомагаючи збільшити продажі товарів або послуг.

Крім прямого спілкування, є можливість проводити точний аналіз, враховуючи смаки користувачів, їх географічне розташування, соціальний статус, запити, що вводяться, збирати інформацію про актуальні тренди та оцінювати популярність певних послуг.

2.1.3 Принцип роботи мобільного застосунка

Мобільні застосунки використовують функціональні можливості пристрою, такі як:

- Робота з камерою;
- Підключення до мереж стільникового зв'язку;
- Можливість підключення сумісних пристроїв через Bluetooth;
- Використання Wi-Fi для доступу в інтернет;
- Використання GPS-навігації.

Процес встановлення мобільних додатків залежить від платформи. Для Android достатньо просто запустити файл APK, а для iOS необхідно попередньо зареєструвати пристрій у особистому акаунті Apple.

2.1.4 Переваги та недоліки мобільних застосунків

Перед тим як приступити до розробки мобільного застосунку, обов'язково ретельно необхідно врахувати всі переваги та недоліки. Незважаючи на очевидні плюси, мобільні застосунки мають свої обмеження.

По-перше, проблема безпеки. Оскільки програмне забезпечення зберігає конфіденційну інформацію користувачів, воно може стати об'єктом атак злоумисників.

Іншою складністю є необхідність підтримки та регулярного оновлення мобільного додатка, що може призвести до додаткових витрат. Крім того, рекламування програми – це ще одна серйозна стаття витрат.

Мобільний застосунок може працювати в фоновому режимі.

2.2 Розробка та створення мобільних застосунків

Для розробки мобільного застосунку для пошуку заправок електрокарів необхідний набір інструментів та єдине середовище розробки (Integrated Development Environment – IDE) (рис. 2.1).



Рисунок 2.1 – Єдине середовище розробки (Integrated Development Environment – IDE)

Середовище розробки – це програма, що забезпечує всі необхідні інструменти для написання коду та розробки інтерфейсу для програмного забезпечення. Воно також має вбудований емулятор для тестування працездатності програм.

Розробка програм для Android здійснюється в Android Studio, а для iOS – у xCode.

Для створення додатків для різних мобільних операційних систем використовуються різні мови програмування. Основна частина коду для iOS пишеться на Objective C і Swift, тоді як на платформі Android переважно використовуються Java і Kotlin.

2.2.1 Середовище розробки мобільних застосунків

Для розробки мобільного застосунку для пошуку заправок електрокарів планується використовувати середовище Android Studio.

Android Studio – це інтегроване середовище розробки, створене Google, яке надає розробникам інструменти для створення додатків на платформі Android OS.

Android Studio (рис. 2.2) можна встановити на Windows, Mac та Linux. Android Studio створене з урахуванням IntelliJ IDEA [9].

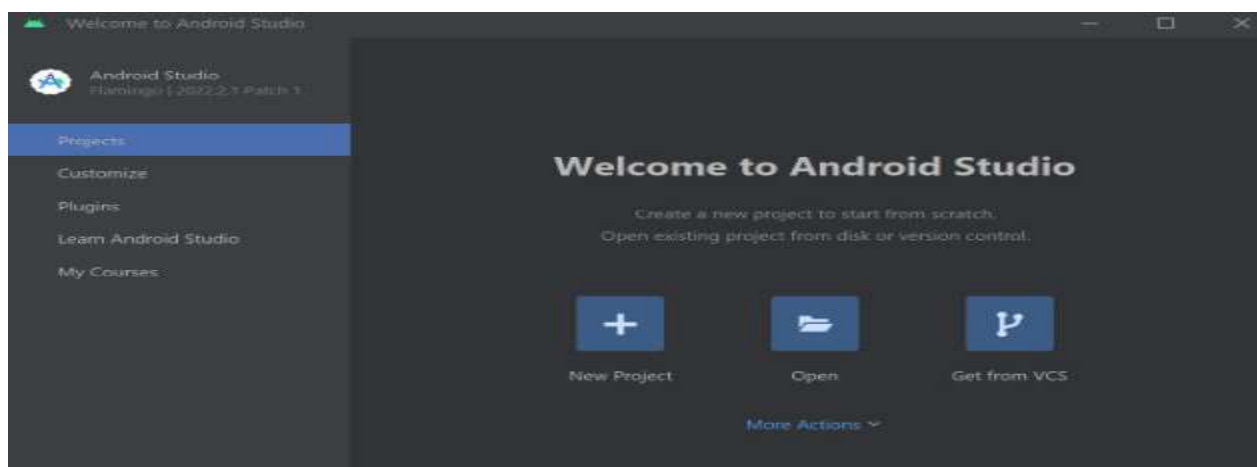


Рисунок 2.2 – Інтерфейс середовища Android Studio

Призначене, як для невеликих команд розробників мобільних додатків, так і для великих міжнародних організацій, що використовують GIT або інші системи управління версіями. Розробка рішень для Android здійснюється в Android Studio з використанням мов Java або Kotlin.

Сумісне з платформою Google App Engine, що дозволяє швидко інтегрувати нові API та функції у хмару. Під час розробки можна використовувати різні API, такі як Google Play, Android Pay і Health. Підтримуються всі версії Android, починаючи з Android 1.6.

Пропонує наступні функції:

- розширений редактор макетів: WYSIWYG, можливість працювати з UI-компонентами за допомогою Drag-and-Drop, функція попереднього перегляду макета на різних конфігураціях екрана;
- складання додатків на базі Gradle;
- різні типи збірок і генерація кількох .apk файлів;
- рефакторинг коду;

- статичний аналізатор коду (Lint), що дозволяє виявляти проблеми з продуктивністю, несумісністю версій тощо;
- вбудований ProGuard і утиліта для підписування додатків;
- шаблони основних макетів і компонентів Android;
- підтримка розробки додатків для Android Wear і Android TV;
- вбудована підтримка Google Cloud Platform, що включає інтеграцію з сервісами Google Cloud Messaging та App Engine.

2.2.2 Мова програмування мобільних застосунків

Для розробки мобільного застосунку для пошуку заправок електрокарів планується використовувати мову програмування Java.

Java – одна з найпопулярніших мов програмування (рис. 2.3), яка завдяки своїм широким можливостям, бібліотекам і портативності дозволяє створювати програмне забезпечення для різних сфер, таких як ігри, мобільні додатки, корпоративні рішення тощо.



Рисунок 2.3 – мова програмування Java

Основні характеристики Java включають:

- обов'язкове оголошення змінних із зазначенням їх типу;
- автоматичне збирання сміття для звільнення пам'яті, зайнятої непотрібними об'єктами;

– використання в таких компаніях, як Netflix, Spotify, Google, LinkedIn та інших.

Переваги Java:

- простий синтаксис і семантика, а також чіткі правила роблять Java відмінним вибором для новачків.
- код, написаний на Java, можна запускати на будь-якій операційній системі, яка підтримує JVM (віртуальну машину Java).
- величезна стандартна бібліотека містить безліч класів і методів для вирішення різноманітних завдань.
- завдяки механізму перевірки типів, відсутності покажчиків на пам'ять і суворій системі дозволів, Java забезпечує високий рівень безпеки.
- вбудована підтримка багатопоточності.
- висока продуктивність завдяки віртуальній машині та оптимізації коду.
- можливість писати код для мобільних додатків, вебсервісів, ігор, фінансових і наукових систем, а також інтернету речей.
- підтримка рефлексії.

Недоліки Java:

- виконання деяких завдань може бути повільнішим;
- потребує більше пам'яті для виконання програм;
- обмежує можливості деяких типів додатків.

Java – це одна з основних мов програмування на платформі Android. За допомогою Android SDK і Java-фреймворків розробники можуть писати різноманітні застосунки, наприклад, ігри або софт для бізнесу [10].

2.2.3 Допоміжні програмні засоби

SQLite – це компактна вбудована реляційна база даних з відкритим кодом. Вона є однією з найпопулярніших у світі, отримала нагороду Google–O'Reilly Open Source Awards і широко використовується в додатках та системах для організації зберігання даних (рис. 2.4).



Рисунок 2.4 – SQLite база даних з відкритим кодом

SQLite не є окремим сервером баз даних, як MySQL або PostgreSQL; натомість це бібліотека, яку можна вбудувати безпосередньо в застосунок.

Особливості SQLite:

- має низькі системні вимоги і споживає мінімальні ресурси, що робить її ідеальною для мобільних застосунків та інших середовищ з обмеженими ресурсами;
- підтримує більшість стандарту SQL, дозволяючи виконувати запити та керувати даними;
- оскільки SQLite не має окремого сервера, доступ до бази даних здійснюється через бібліотеки API різними мовами програмування, що спрощує інтеграцію з базою даних у застосунку;
- підтримує транзакції, що забезпечує цілісність даних та безпеку при виконанні багатьох операцій одночасно;
- широко використовується в різних типах програм, включаючи мобільні та десктопні додатки, браузері тощо.

Вся база даних SQLite зберігається в одному файлі на диску, який має розширення .sqlite або .db і містить всі таблиці, індекси, схему та дані. Його можна копіювати, переміщати та резервувати, як звичайний файл.

Вибір SQLite залежить від конкретних вимог та обмежень проекту, і важливо знати її переваги та недоліки.

Переваги SQLite:

- дуже легка база даних, яка не потребує окремого сервера. Вона працює швидко та ефективно, особливо на системах з обмеженими ресурсами;
- відносно проста у використанні і не вимагає складних налаштувань;
- взаємодія з базою даних можлива за допомогою стандартних мов програмування і SQL;
- можна вбудовувати безпосередньо в програмне забезпечення, що полегшує розповсюдження та розгортання додатків;
- всі дані зберігаються в одному файлі бази даних, що робить резервне копіювання і перенесення простим завданням;
- підтримує транзакції, що забезпечує цілісність і безпеку даних;
- має багато різних оболонок і бібліотек, які роблять SQLite доступною для багатьох мов програмування, таких як Python, Java, C# тощо.

Недоліки SQLite:

- не підтримує велику кількість одночасних записів, що може призводити до блокувань і конфліктів;
- не підходить для великих обсягів даних або високонавантажених систем через обмежену підтримку;
- не підтримує віддалений доступ через мережу;
- відсутня безкоштовна технічна підтримка, тому проблеми доведеться вирішувати через спільноту;
- призначена в основному для операцій CRUD (створення, читання, оновлення, видалення) і не підходить для складних аналітичних задач.

Хоча SQLite має багато переваг, важливо також враховувати обмеження СУБД, особливо щодо одночасного доступу та роботи з великими обсягами даних. Для високонавантажених проектів краще використовувати інші систем управління базами даних: наприклад, MySQL, PostgreSQL або Microsoft SQL Server [11].

2.3 Постановка завдання розробки та дослідження

Метою даного проекту є розробка мобільного застосунку для пошуку заправок електрокарів.

З метою підготовки матеріальної бази, для написання дипломної роботи бакалавра, необхідно:

- провести аналіз предметної області;
- здійснити аналіз існуючих аналогів мобільних застосунків;
- сформулювати постановку завдання, вимоги і задачі до об'єкту розробки;
- сформувати середовище розробки, обрані програмні засоби для реалізації проекту.

3 РОЗРОБКА ПРОГРАМНОЇ ЧАСТИНИ ЗАСТОСУНКУ

3.1 Розробка архітектури застосунку

Основним етапом в реалізації мобільного додатка є розробка архітектури. Архітектура, побудована належним чином, здатна полегшити майбутнє розширення і розвиток програми.

Для розробки мобільного застосунку для пошуку заправок електрокарів буде використовуватися середовище Android Studio та мова програмування Java.

Розглянемо опис роботи мобільного застосунку крок за кроком (креслення А1):

- користувач взаємодіє з мобільним застосунком, вводячи відгуки про заправок електрокарів;
- мобільний застосунок перетворює дані, введені користувачем, у формат JSON. JSON - це легкий формат даних, який використовується для обміну інформацією;

- мобільний застосунок надсилає запит на сервер через REST API. REST API - це набір принципів, що визначають, як веб-застосунки взаємодіють з серверами. Запит містить дані JSON, перетворені на кроці 2, а також інші необхідні параметри;
- обробка запиту на сервері: Сервер отримує запит від мобільного застосунку та обробляє його. Далі сервер отримує доступ до бази даних, щоб отримати або оновити інформацію, пов'язану з запитом. Наприклад, якщо користувач ввів відгук, сервер може отримати доступ до бази даних, щоб отримати інформацію про всі відгуки запропонованої заправки електрокарів;
- після обробки запиту сервер надсилає відповідь мобільному застосунку. Відповідь містить дані JSON, які включають інформацію про відгук;
- мобільний застосунок отримує відповідь від сервера та візуалізує дані, а також використовує Google Maps API для відображення мапи заправок, а саме маркери - це точки на мапі, які представляють певну локацію заправки;
- користувач може взаємодіяти з даними, візуалізований на мапі. Наприклад, користувач може натиснути на маркер, щоб отримати додаткову інформацію про місце знаходження заправки для електрокарів та середню оцінку наданих послуг. На рисунку рис. 3.1 наведена архітектура застосунку, що розробляється.

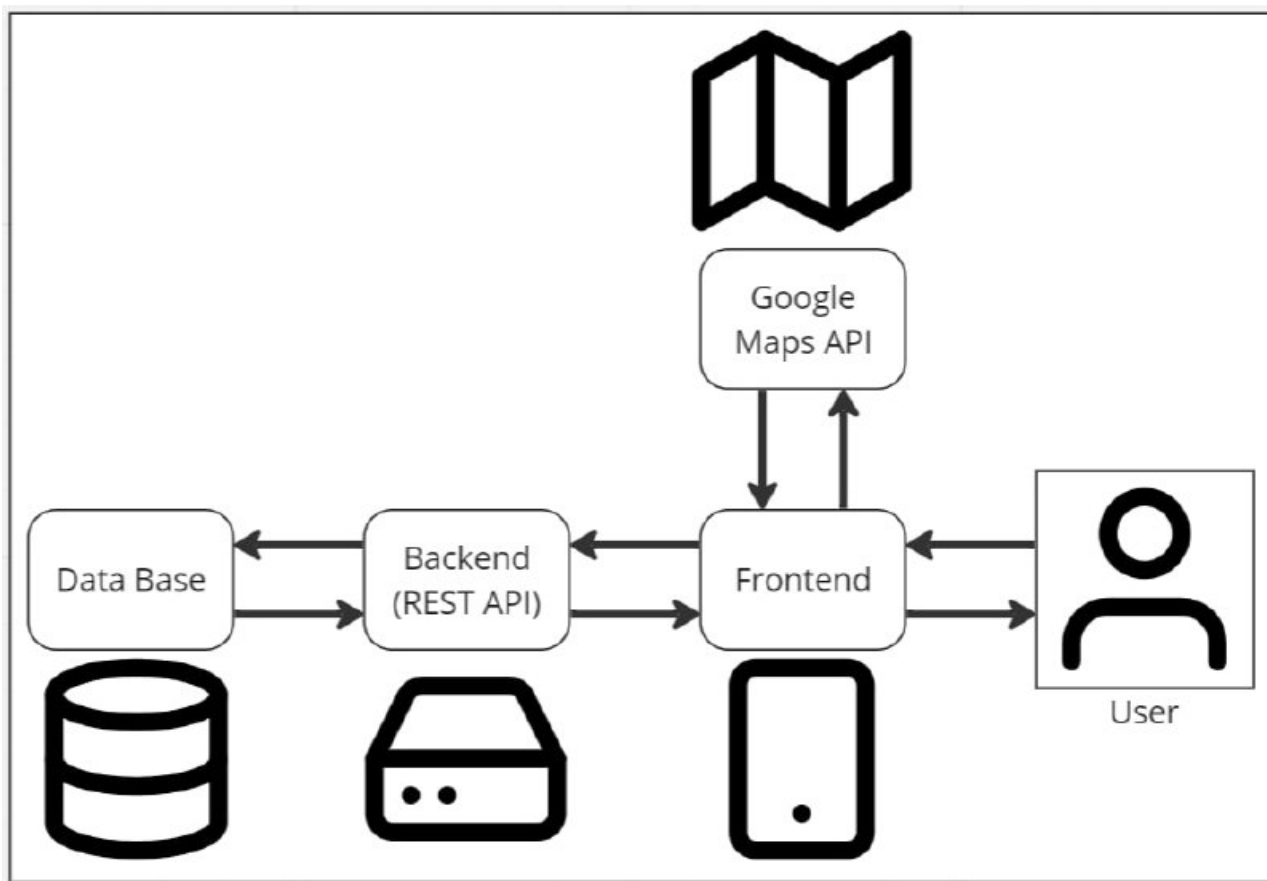


Рисунок 3.1 – Архітектура мобільного застосунку

3.1.1 Отримання доступу до Google Maps

Безпосередньо для роботи з мапою пошуку заправок електрокарів було встановлено в файлі маніфесту `AndroidManifest.xml` відповідний дозвіл:

Лістинг 3.1 - Метадані API-ключа Google Maps

```

<meta-data
    android:name="com.google.android.geo.API_KEY"
    android:value="AIzaSyAtdtBX8uoFBkHjKLg36blRFN4IECaxNao" />

```

Оскільки завантаження даних може зайняти деякий час, то для звернення до інтернету визначимо асинхронну операцію, а для організації візуального інтерфейсу було використано фрагменти.

З метою відображення мапи застосунку на екрані мобільного телефону у файлі `fragment_map.xml`, який представляє розмітку фрагмента.

Файл `fragment_map.xml` описує макет для фрагмента застосунку, який відображає мапу з використанням `MapView` від Google Maps. Макет визначений у відносному розташуванні (`RelativeLayout`), що дозволяє компонованню елементів відносно один до одного.

Лістинг 3.2 - Оголошення `RelativeLayout`

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
```

`RelativeLayout` є контейнером, який дозволяє розташовувати елементи відносно один одного або батьківського контейнера.

Визначає простір імен, який необхідний для використання атрибутів `Android` у цьому макеті.

Встановлює ширину та висоту контейнера `RelativeLayout` на весь екран батьківського контейнера.

Лістинг 3.3 - Оголошення елемента `MapView`

```
<com.google.android.gms.maps.MapView
    android:id="@+id/mapView"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

`MapView` є спеціалізованим видом, який відображає карту Google Maps у застосунку.

```
android:id="@+id/mapView"
```

Встановлює унікальний ідентифікатор для цього елемента, що дозволяє програмно взаємодіяти з ним у Java коді (наприклад, за допомогою `findViewById`).

```
android:layout_width="match_parent"  
android:layout_height="match_parent"
```

Встановлює ширину та висоту `MapView` на весь екран батьківського контейнера (`RelativeLayout`).

Як результат, макет визначає простий інтерфейс, що складається з одного елемента `MapView`, який займає весь екран фрагмента. `MapView` використовується для відображення мапи Google Maps у вашому застосунку. Відносне розташування (`RelativeLayout`) дозволяє потенційно додавати інші елементи навколо або поверх `MapView`, якщо це необхідно в майбутньому.

В подальшому, щоб користувач застосунку мав змогу в повному обсязі користуватись функціоналом мапи для пошуку заправок електрокарів з використанням Google Maps API у класі фрагмента `FragmentMap` доданий код, який наведено на лістингу A.1 додатку A.

Цей фрагмент коду визначає пакет `com.daniil.myapplication` та імпортує необхідні класи з Google Maps API, які знадобляться для роботи з мапами та в подальшому з маркерами, наведено на лістингу A.1 додатку A.

Клас `FragmentMap` наслідується від `Fragment` і реалізує інтерфейс `OnMapReadyCallback`. Це означає, що клас буде містити методи для роботи з мапою, які відображаються у фрагменті, наведено на лістингу A.1 додатку A.

Вказаний метод відповідає за створення та налаштування представлення фрагмента. Він ініціалізує `MapView`, зв'язує його з макетом (використовуючи `R.layout.fragment_map`), і викликає метод `getMapAsync (this)`, який асинхронно завантажує мапу та повідомляє про це через `onMapReady`, наведено на лістингу A.1 додатку A.

Метод `onMapReady` викликається, коли мапа готова до використання. Тут зберігаємо посилання на об'єкт `Google Map`, потім налаштовуємо початковий вигляд мапи, задаючи координати (широта та довгота) і рівень масштабу. У даному випадку карта центрована на місті Запоріжжя з рівнем масштабу 10.0, наведено на лістингу А.1 додатку А.

Зазначені методи забезпечують коректну роботу `MapView` відповідно до життєвого циклу фрагмента. Вони викликаються у відповідні моменти життєвого циклу та забезпечують правильне управління ресурсами мапи, такими як пауза, відновлення та знищення мапи.

Після завершення створення фрагменту ініціалізується `MapView`, яка асинхронно завантажує мапу та налаштовує її початковий вигляд, центруючи на заданих координатах. Життєвий цикл `MapView` коректно управляється методами `onResume`, `onPause`, `onDestroy` та `onLowMemory`, що забезпечує правильне використання ресурсів і уникнення витоків пам'яті.

3.1.2 Налаштування мапи розміщення заправок електрокарів

Клас `ChargingStation`, представляє заправку для електрокарів. Клас містить кілька полів для зберігання інформації про заправку, а також відповідні методи для доступу та зміни цих даних.

Лістинг 3.4 - Імпорт пакетів для роботи з Google Maps

```
package com.daniil.myapplication;

import com.google.android.gms.maps.model.LatLng;

import java.util.List;
package com.daniil.myapplication
```

Імпортується клас `LatLng` з `Google Maps API`, який використовується для зберігання координат (широти та довготи) заправки.

Лістинг 3.5 - оголошення класу ChargingStation

```
public class ChargingStation {
    private String stationId;
    private String title;
    private String snippet;
    private LatLng location;
    private float markerColor;
}
```

Клас ChargingStation містить наступні приватні поля:

stationId - унікальний ідентифікатор заправки;

title - назва заправки;

snippet - короткий опис заправки;

location - об'єкт LatLng, що зберігає координати заправки;

markerColor - колір маркера на карті для цієї заправки.

Лістинг 3.6 - Конструктор класу ChargingStation

```
public ChargingStation(String stationId, String title, String
snippet, LatLng location, float markerColor) {
    this.stationId = stationId;
    this.title = title;
    this.snippet = snippet;
    this.location = location;
    this.markerColor = markerColor;
}
```

Конструктор класу приймає п'ять параметрів і ініціалізує відповідні поля об'єкта ChargingStation.

Лістинг 3.7 - Гетери та сетери класу ChargingStation

```
public String getStationId() {
    return stationId;
}
```

```
public void setStationId(String stationId) {
    this.stationId = stationId;
}

public String getTitle() {
    return title;
}

public void setTitle(String title) {
    this.title = title;
}

public String getSnippet() {
    return snippet;
}

public void setSnippet(String snippet) {
    this.snippet = snippet;
}

public LatLng getLocation() {
    return location;
}

public void setLocation(LatLng location) {
    this.location = location;
}

public float getMarkerColor() {
    return markerColor;
}

public void setMarkerColor(float markerColor) {
    this.markerColor = markerColor;
}
```

Зазначені методи забезпечують доступ до приватних полів класу:

`getStationId` та `setStationId` - для отримання та встановлення значення `stationId`;

`getTitle` та `setTitle` - для отримання та встановлення значення `title`;

`getSnippet` та `setSnippet` - для отримання та встановлення значення `snippet`;

`getLocation` та `setLocation` - для отримання та встановлення значення `location`;

`getMarkerColor` та `setMarkerColor` - для отримання та встановлення значення `markerColor`.

У підсумку клас `ChargingStation` представляє заправку для електрокарів з кількома полями для зберігання інформації про заправку, включаючи її ідентифікатор, назву, опис, місцезнаходження та колір маркера на карті. Конструктор класу дозволяє ініціалізувати ці поля під час створення об'єкта, а геттери та сеттери забезпечують доступ до цих полів для читання та зміни їх значень.

Наступним кроком до фрагменту класу `FragmentManager` додані заправки електрокарів, а також два методи : `addChargingStation`- інформація про заправки та `addMarkers` – додавання маркерів на мапу, наведено на лістингу А.1 додатку А.

Кожен виклик методу додає маркер заправки на вказані координати з відповідними параметрами:

`stationId` - унікальний ідентифікатор заправки;

`latitude` та `longitude` - географічні координати заправки;

`title` - заголовок маркера;

`snippet` - короткий опис або адреса заправки;

`markerColor` - колір маркера, визначений через `BitmapDescriptorFactory`, наведено на лістингу А.1 додатку А.

Викликається конструктор `ChargingStation` для створення нового об'єкта заправки, який додається до списку `chargingStations` (Додаток А).

Метод `addMarkers` відповідає за додавання маркерів на мапу для кожної заправки електрокарів, яка зберігається в списку `chargingStations`.

Для кожної заправки у списку створюється об'єкт `MarkerOptions`, що налаштовує параметри маркера:

`position(station.getLocation())` - встановлює координати маркера;

`title(station.getTitle())` - встановлює заголовок маркера;

`snippet(station.getSnippet())` - встановлює короткий опис маркера;

`icon(BitmapDescriptorFactory.defaultMarker(station.getMarkerColor()))` - встановлює колір маркера.

Після налаштування параметрів маркер додається на мапу за допомогою методу `googleMap.addMarker(markerOptions)`.

Метод `setTag` прив'язує об'єкт заправки до маркера, що дозволяє зберігати додаткову інформацію про заправки електрокарів в маркері.

3.1.3 Створення бази даних SQLite

З метою отримання та накопичення даних щодо відгуків користувачів про заправки електрокарів та керування ними потрібно створити базу даних.

Спочатку створимо структуру бази даних (рис. 3.2)

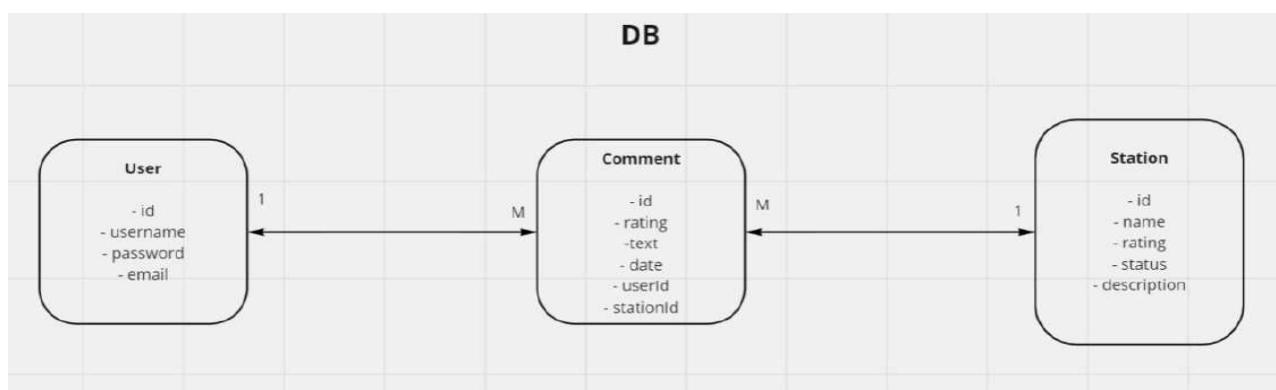


Рисунок 3.2 – Структура бази даних

За допомогою класу `DatabaseHelper` створено відповідну базу даних. Він містить різноманітні методи для створення, оновлення та видалення записів у базі даних.

Назва бази даних та версія:

DATABASE_NAME - назва файлу бази даних.

DATABASE_VERSION - версія бази даних.

Лістинг 3.8 - Константи для назви та версії бази даних

```
private      static      final      String      DATABASE_NAME      =
"charging_station_reviews.db";
private static final int DATABASE_VERSION = 1;
```

Назви таблиці та стовпців:

TABLE_REVIEWS - назва таблиці відгуків.

COLUMN_ID, COLUMN_STATION_ID, COLUMN_RATING,
COLUMN_REVIEW_TEXT, COLUMN_DATE_ADDED - назви стовпців у
таблиці.

Лістинг 3.9 - Константи для таблиці відгуків у базі даних

```
public static final String TABLE_REVIEWS = "reviews";
public static final String COLUMN_ID = "id";
public static final String COLUMN_STATION_ID = "station_id";
public static final String COLUMN_RATING = "rating";
public static final String COLUMN_REVIEW_TEXT = "review_text";
public static final String COLUMN_DATE_ADDED = "date_added";
```

Конструктор, який викликає суперклас SQLiteOpenHelper, для ініціалізації
бази даних:

Лістинг 3.10 - Клас-допоміжник бази даних

```
public DatabaseHelper(Context context) {
    super(context, DATABASE_NAME, null, DATABASE_VERSION);
}
```

Метод onCreate створює таблицю відгуків, якщо вона не існує:

Лістинг 3.11 – Метод onCreate

```
@Override
public void onCreate(SQLiteDatabase db) {
    String CREATE_REVIEWS_TABLE = "CREATE TABLE " +
TABLE_REVIEWS + "("
        + COLUMN_ID + " INTEGER PRIMARY KEY,"
        + COLUMN_STATION_ID + " TEXT,"
        + COLUMN_RATING + " REAL,"
        + COLUMN_REVIEW_TEXT + " TEXT,"
        + COLUMN_DATE_ADDED + " TEXT"
        + ")";
    db.execSQL(CREATE_REVIEWS_TABLE);
}
```

Оновлення бази даних здійснюється за допомогою методу onUpgrade , який видаляє стару таблицю, якщо вона існує, і створює нову.

Лістинг 3.12 - Оновлення бази даних

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion) {
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_REVIEWS);
    onCreate(db);
}
```

Додавання нових відгуків про заправку електорокарів до таблиці здійснюється за допомогою методу addReview.

Лістинг 3.13 – Метод addReview

```
public void addReview(String stationId, double rating, String
reviewText, String dateAdded) {
```

```

        SQLiteDatabase db = this.getWritableDatabase();
        ContentValues values = new ContentValues();
        values.put(COLUMN_STATION_ID, stationId);
        values.put(COLUMN_RATING, rating);
        values.put(COLUMN_REVIEW_TEXT, reviewText);
        values.put(COLUMN_DATE_ADDED, dateAdded);
        db.insert(TABLE_REVIEWS, null, values);
        db.close();
    }

```

Отримання всіх відгуків для певної заправки електрокарів здійснюється за допомоги методу `getAllReviewsForStation`, який повертає всі відгуки для конкретної заправки електрокарів.

Лістинг 3.14 – Метод `getAllReviewsForStation`

```

public Cursor getAllReviewsForStation(String stationId) {
    SQLiteDatabase db = this.getReadableDatabase();
    String[] columns = {COLUMN_ID, COLUMN_RATING,
        COLUMN_REVIEW_TEXT, COLUMN_DATE_ADDED};
    String selection = COLUMN_STATION_ID + "=?";
    String[] selectionArgs = {stationId};
    Cursor cursor = db.query(TABLE_REVIEWS, columns,
        selection, selectionArgs, null, null, null);

    if (cursor != null && cursor.getCount() > 0) {
        cursor.moveToFirst();
    }

    return cursor;
}

```

За допомогою методу `updateReview` оновлюється відгук за його ID.

Лістинг 3.15 – Метод updateReview

```

public int updateReview(int id, double rating, String
reviewText, String dateAdded) {
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues values = new ContentValues();
    values.put(COLUMN_RATING, rating);
    values.put(COLUMN_REVIEW_TEXT, reviewText);
    values.put(COLUMN_DATE_ADDED, dateAdded);
    return db.update(TABLE_REVIEWS, values, COLUMN_ID + "=?",
new String[]{String.valueOf(id)});
}

```

Метод deleteReview видаляє відгук за його ID.

Лістинг 3.16 – Метод deleteReview

```

public void deleteReview(int id) {
    SQLiteDatabase db = this.getWritableDatabase();
    db.delete(TABLE_REVIEWS, COLUMN_ID + "=?", new
String[]{String.valueOf(id)});
    db.close();
}

```

Метод addReviewAndRating додає відгук та рейтинг до бази даних, автоматично встановлюючи поточну дату та час.

Лістинг 3.17 - Метод addReviewAndRating

```

public boolean addReviewAndRating(String stationId, String
reviewText, float rating) {
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues values = new ContentValues();
    values.put("station_id", stationId);
    values.put("review_text", reviewText);
    values.put("rating", rating);
    values.put("date_added", System.currentTimeMillis());
}

```

```

        long result = db.insert("reviews", null, values);
        db.close();

        return result != -1;
    }

```

3.1.4 Доопрацювання користувацького інтерфейсу застосунку

Layout_review_dialog.xml файл описує макет користувацького інтерфейсу для залишення відгуків про заправки електрокарів в додатку Android. Макет організований у вертикальний LinearLayout і містить кілька елементів, таких як текстове поле, рейтингова шкала, поле введення тексту, кнопки для надсилання та перегляду відгуків, а також RecyclerView для відображення коментарів.

Основний контейнер. LinearLayout з вертикальною орієнтацією, який займає всю ширину екрану та має динамічну висоту, залежно від вмісту. Встановлено внутрішні відступи 16dp з усіх сторін.

Лістинг 3.18 – Оголошення LinearLayout

```

<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:padding="16dp">

```

Текстове поле з інструкцією. TextView, який відображає інструкцію для користувача. Текст розміром 18sp та чорного кольору. Нижній відступ 16dp для відокремлення від наступного елемента.

Лістинг 3.19 – Оголошення TextView

```

<TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"

```

```

        android:text="Залиште відгук та поставте оцінку для цієї
зарядної заправки:"
        android:textSize="18sp"
        android:textColor="@android:color/black"
    android:layout_marginBottom="16dp"/>

```

Рейтингова шкала. `RatingBar` для встановлення оцінки, з максимальним значенням у 5 зірок. Початковий рейтинг 0 та крок встановлення 1. Нижній відступ 16dp.

Лістинг 3.20 – Оголошення `RatingBar`

```

<RatingBar
    android:id="@+id/ratingBar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:numStars="5"
    android:rating="0"
    android:stepSize="1"
    android:layout_marginBottom="16dp"/>

```

Поле введення тексту для відгуку. `EditText` для введення тексту відгуку. Ширина займає всю доступну ширину контейнера. Встановлено підказку "Введіть ваш відгук". Тип введення – багаторядковий текст з мінімум 3 рядками та максимум 5 рядками.

Лістинг 3.21 – Оголошення `EditText`

```

<EditText
    android:id="@+id/editTextReview"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Введіть ваш відгук"
    android:inputType="textMultiLine"
    android:minLines="3"
    android:maxLines="5"/>

```

Кнопка для надсилання відгуку. Button для надсилання відгуку. Верхній відступ 16dp.

Лістинг 3.22 – Оголошення Button для надсилання відгуку

```
<Button
    android:id="@+id/buttonSubmitReview"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Відправити відгук"
    android:layout_marginTop="16dp"/>
```

Кнопка для виходу з вікна відгуків. Button для виходу з інтерфейсу відгуків. Верхній відступ 16dp.

Лістинг 3.23 – Оголошення Button для виходу з вікна відгуків

```
<Button
    android:id="@+id/buttonExitReview"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Вийти"
    android:layout_marginTop="16dp"/>
```

Кнопка для перегляду попередніх коментарів. Button для перегляду попередніх відгуків. Верхній відступ 16dp.

Лістинг 3.24 – Оголошення Button для перегляду попередніх коментарів

```
<Button
    android:id="@+id/buttonViewPreviousComments"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Переглянути попередні коментарі"
    android:layout_marginTop="16dp"/>
```

RecyclerView для відображення коментарів. RecyclerView для відображення списку коментарів. Ширина займає всю доступну ширину контейнера. Початково прихований (`visibility="gone"`). Верхній відступ 16dp.

Лістинг 3.25 – Оголошення RecyclerView

```
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/recyclerViewComments"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:visibility="gone"
    android:layout_marginTop="16dp"/>
```

Розроблений макет забезпечує простий та інтуїтивний інтерфейс для користувачів, що дозволяє їм залишати відгуки, переглядати попередні коментарі та виходити з вікна відгуків.

Код описує активність ReviewActivity в Android додатку, яка дозволяє користувачам залишати відгуки та оцінки для зарядних станцій, а також переглядати попередні коментарі.

Оголошення класу та імпортування бібліотек. Імпорт необхідних класів та бібліотек для роботи з базою даних, користувацьким інтерфейсом та управлінням активністю.

Лістинг 3.26 - Імпорт пакетів і класів

```
package com.daniil.myapplication;

import android.content.Intent;
import android.database.Cursor;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.RatingBar;
import android.widget.Toast;
```

```
import androidx.appcompat.app.AppCompatActivity;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;

import com.daniil.myapplication.database.DatabaseHelper;

import java.util.ArrayList;
import java.util.List;
```

Оголошення полів класу ReviewActivity. Оголошення змінних для елементів інтерфейсу, допоміжних класів та списку коментарів про заправки електрокарів.

Лістинг 3.27 - активність "Відгуки"

```
public class ReviewActivity extends AppCompatActivity {

    private EditText editTextReview;
    private RatingBar ratingBar;
    private Button buttonSubmitReview;
    private Button buttonExitReview;
    private Button buttonViewPreviousComments;
    private DatabaseHelper dbHelper;
    private String selectedStationId;

    // Оголошення RecyclerView та адаптера
    private RecyclerView recyclerViewComments;
    private CommentsAdapter commentsAdapter;
    private List<Comment> commentsList;
```

Метод onCreate для ініціалізації активності. Встановлення макета для активності та отримання ідентифікатора обраної заправки з Intent.

Лістинг 3.28 – Метод onCreate

```
@Override
```

```

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.layout_review_dialog);

    // Отримання ідентифікатора обраної заправки з Intent
    Intent intent = getIntent();
    selectedStationId =
intent.getStringExtra("selectedStationId");

```

Ініціалізація елементів інтерфейсу та бази даних. Зв'язування змінних з елементами інтерфейсу та створення об'єкту для роботи з базою даних.

Лістинг 3.29 - Ініціалізація елементів

```

    editTextReview = findViewById(R.id.editTextReview);
    ratingBar = findViewById(R.id.ratingBar);
    buttonSubmitReview =
findViewById(R.id.buttonSubmitReview);
    buttonExitReview = findViewById(R.id.buttonExitReview);
    buttonViewPreviousComments =
findViewById(R.id.buttonViewPreviousComments);

    dbHelper = new DatabaseHelper(this);

    // Ініціалізація списку коментарів
    commentsList = new ArrayList<>();

```

Налаштування RecyclerView та адаптера. Ініціалізація RecyclerView та встановлення адаптера для відображення коментарів.

Лістинг 3.30 - налаштування RecyclerView для коментарів

```

    recyclerViewComments =
findViewById(R.id.recyclerViewComments);
    commentsAdapter = new CommentsAdapter(this, commentsList);

```

```

        recyclerViewComments.setLayoutManager(new
LinearLayoutManager(this));
        recyclerViewComments.setAdapter(commentsAdapter);

```

**Обробники натискання кнопок. Перегляд попередніх коментарів.
Отримання коментарів з бази даних та їх відображення у RecyclerView.**

Лістинг 3.31 - Налаштування слухача та натискання кнопки

```

buttonViewPreviousComments.setOnClickListener(new
View.OnClickListener() {
    @Override
    public void onClick(View v) {
        commentsList = getPreviousCommentsFromDatabase();
        if (commentsList.isEmpty()) {
            Toast.makeText(ReviewActivity.this, "Немає
попередніх коментарів", Toast.LENGTH_SHORT).show();
        } else {
            commentsAdapter.setComments(commentsList);
            recyclerViewComments.setVisibility(View.VISIBLE);
            commentsAdapter.notifyDataSetChanged();
        }
    }
});

```

Вихід з активності. Закриття активності.

Лістинг 3.32 - Вихід з активності або закриття активності

```

buttonExitReview.setOnClickListener(new
View.OnClickListener() {
    @Override
    public void onClick(View v) {
        finish();
    }
});

```

Відправка відгуку. Перевірка введення тексту та рейтингу, додавання відгуку до бази даних та виведення повідомлень про успіх або помилку.

Лістинг 3.33 - Налаштування натискання кнопки для надсилання відгуку.

```
buttonSubmitReview.setOnClickListener(new
View.OnClickListener() {
    @Override
    public void onClick(View v) {
        String reviewText =
editTextReview.getText().toString();
        float rating = ratingBar.getRating();
        if (reviewText.trim().isEmpty()) {
            Toast.makeText(ReviewActivity.this, "Введіть
відгук", Toast.LENGTH_SHORT).show();
        } else if (rating == 0) {
            Toast.makeText(ReviewActivity.this, "Вкажіть
рейтинг", Toast.LENGTH_SHORT).show();
        } else {
            boolean isReviewAdded =
dbHelper.addReviewAndRating(selectedStationId, reviewText, rating);
            if (isReviewAdded) {
                Toast.makeText(ReviewActivity.this, "Відгук
успішно доданий", Toast.LENGTH_SHORT).show();
                finish();
            } else {
                Toast.makeText(ReviewActivity.this, "Помилка
при додаванні відгуку", Toast.LENGTH_SHORT).show();
            }
        }
    }
});
```

Метод для отримання попередніх коментарів з бази даних. Отримання всіх відгуків для заданої заправки електрокарів з бази даних та створення списку коментарів.

Лістинг 3.34 – Метод `getPreviousCommentsFromDatabase`

```
private List<Comment> getPreviousCommentsFromDatabase() {
    List<Comment> comments = new ArrayList<>();
    Cursor cursor =
dbHelper.getAllReviewsForStation(selectedStationId);
    if (cursor != null && cursor.moveToFirst()) {
        do {
            if
(cursor.getColumnIndex(DatabaseHelper.COLUMN_ID) != -1 &&
cursor.getColumnIndex(DatabaseHelper.COLUMN_RATING) != -1 &&
cursor.getColumnIndex(DatabaseHelper.COLUMN_REVIEW_TEXT) != -1 &&
cursor.getColumnIndex(DatabaseHelper.COLUMN_DATE_ADDED) != -1) {
                int id =
cursor.getInt(cursor.getColumnIndex(DatabaseHelper.COLUMN_ID));
                float rating =
cursor.getFloat(cursor.getColumnIndex(DatabaseHelper.COLUMN_RATING
));
                String reviewText =
cursor.getString(cursor.getColumnIndex(DatabaseHelper.COLUMN_REVIEW
TEXT));
                String dateAdded =
cursor.getString(cursor.getColumnIndex(DatabaseHelper.COLUMN_DATE_
ADDED));
                Comment comment = new Comment(id, rating,
reviewText, dateAdded);
                comments.add(comment);
            }
        } while (cursor.moveToNext());
    }
```

```

        cursor.close();
    }
    return comments;
}

```

Цей код описує клас `Comment`, який використовується для представлення відгуків про заправки електрокарів. Клас містить поля для збереження ідентифікатора відгуку, рейтингу, тексту відгуку та дати додавання відгуку.

Клас `Comment` оголошений у пакеті `com.daniil.myapplication`. Поля класу включають: `id` (ідентифікатор відгуку), `rating` (рейтинг), `reviewText` (текст відгуку) та `dateAdded` (дата додавання відгуку).

Лістинг 3.35 – Клас `Comment`

```

package com.daniil.myapplication;

public class Comment {
    private int id;
    private float rating;
    private String reviewText;
    private String dateAdded;
}

```

Порожній конструктор `Comment()`, який не приймає жодних аргументів. Конструктор `Comment(int id, float rating, String reviewText, String dateAdded)`, який приймає всі необхідні параметри для ініціалізації полів класу.

Лістинг 3.36 – Конструктори класу `Comment`

```

public Comment() {
    // Пустий конструктор
}

public Comment(int id, float rating, String reviewText,
String dateAdded) {
    this.id = id;
}

```

```

        this.rating = rating;
        this.reviewText = reviewText;
        this.dateAdded = dateAdded;
    }

```

Геттер-методи:

- метод getId() повертає значення поля id;
- метод getRating() повертає значення поля rating;
- метод getReviewText() повертає значення поля reviewText;
- метод getDateAdded() повертає значення поля dateAdded.

Лістинг 3.37 - Гетери та сетери класу Comment

```

public int getId() {
    return id;
}

public float getRating() {
    return rating;
}

public String getReviewText() {
    return reviewText;
}

public String getDateAdded() {
    return dateAdded;
}

```

Comment клас дозволяє створювати об'єкти, які представляють окремі відгуки про зарядні заправки, зберігаючи всю необхідну інформацію для кожного відгуку.

Цей код описує адаптер для RecyclerView, який використовується для відображення списку коментарів у вигляді елементів інтерфейсу. Клас

CommentsAdapter забезпечує взаємодію між даними (списком коментарів) та уявленнями (елементами списку).

Оголошення класу та полів. Клас CommentsAdapter успадковує RecyclerView.Adapter, використовуючи CommentViewHolder.

Поля класу включають commentsList (список коментарів) та context (контекст додатку).

Лістинг 3.38 – Клас CommentsAdapter

```
package com.daniil.myapplication;

import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.RatingBar;
import android.widget.TextView;
import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;

import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.List;
import java.util.Locale;

public class CommentsAdapter extends
RecyclerView.Adapter<CommentsAdapter.CommentViewHolder> {
    private List<Comment> commentsList;
    private Context context;
```

Конструктор приймає контекст та список коментарів для ініціалізації відповідних полів.

Лістинг 3.39 – Конструктор класу CommentsAdapter

```
// Конструктор
```

```

        public CommentsAdapter(Context context, List<Comment>
commentsList) {
            this.context = context;
            this.commentsList = commentsList;
        }

```

Метод `setComments` оновлює список коментарів та повідомляє адаптер про зміни.

Лістинг 3.40 – Метод `setComments`

```

// Метод для встановлення списку коментарів
public void setComments(List<Comment> commentsList) {
    this.commentsList = commentsList;
    notifyDataSetChanged();
}

```

Оголошення `ViewHolder`: Внутрішній клас `CommentViewHolder` містить оголошення всіх елементів інтерфейсу, які використовуються для відображення коментаря.

Лістинг 3.41 - Оголошення `ViewHolder`

```

// Оголошення ViewHolder
public static class CommentViewHolder extends
RecyclerView.ViewHolder {
    public TextView textViewRating;
    public RatingBar ratingBar;
    public TextView textViewComment;
    public TextView textViewDateAddedTitle;
    public TextView textViewDateAdded;

    public CommentViewHolder(@NonNull View itemView) {
        super(itemView);
        textViewRating
        =
itemView.findViewById(R.id.textViewRating);

```

```

        ratingBar = itemView.findViewById(R.id.ratingBar);
        textViewComment
itemView.findViewById(R.id.textViewCommentText);
        textViewDateAddedTitle
itemView.findViewById(R.id.textViewDateAddedTitle);
        textViewDateAdded
itemView.findViewById(R.id.textViewDateAdded);
    }
}

```

Методи `onCreateViewHolder` та `onBindViewHolder`:

- `onCreateViewHolder` створює новий `CommentViewHolder` шляхом заповнення макету `comment_item`.
- `onBindViewHolder` встановлює дані коментаря (текст, рейтинг та дату) в елементи інтерфейсу наведено на лістингу Б.1 додатку Б.

Метод `getItemCount` повертає кількість коментарів у списку. Цей адаптер забезпечує динамічне відображення коментарів у `RecyclerView`, обробляючи налаштування інтерфейсу для кожного окремого коментаря.

Лістинг 3.42 - Метод `getItemCount`

```

@Override
public int getItemCount() {
    return commentsList.size();
}

```

`Comment_item.xml` - файл описує макет елемента інтерфейсу, який використовується для відображення коментарів з оцінкою, текстом відгуку та датою додавання. Макет побудований на основі `RelativeLayout`.

Лістинг 3.43 - розмітка `RelativeLayout`

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"

```

```

        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="16dp">

```

`RelativeLayout` використовується як основний контейнер. Встановлюються параметри ширини `match_parent` і висоти `wrap_content`. Встановлюється внутрішній відступ (`padding`) у 16dp.

`TextView` для відображення тексту "Оцінка:". Встановлюються параметри ширини та висоти `wrap_content`.

Текст встановлюється "Оцінка:", розмір тексту — 16sp, стиль — жирний (**bold**). Встановлюється верхній відступ (`marginTop`) у 8dp.

Лістинг 3.44 – Оголошення `TextView`

```

<TextView
    android:id="@+id/textViewRating"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Оцінка:"
    android:textSize="16sp"
    android:textStyle="bold"
    android:layout_marginTop="8dp" />

```

`RatingBar` для відображення рейтингу. Встановлюються параметри ширини та висоти `wrap_content`. Встановлюється розташування нижче `textViewRating` (`layout_below`). Встановлюється режим лише для читання (`isIndicator="true"`). Встановлюється максимальна кількість зірок `numStars="5"` та крок `stepSize="1"`. Встановлюється верхній відступ (`marginTop`) у 4dp.

Лістинг 3.45 – Оголошення `RatingBar`

```

<RatingBar
    android:id="@+id/ratingBar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"

```

```

        android:layout_below="@id/textViewRating"
        android:isIndicator="true"
        android:numStars="5"
        android:stepSize="1"
        android:layout_marginTop="4dp" />

```

Текстовий віджет для заголовка відгуку. `TextView` для відображення тексту "Відгук:". Встановлюються параметри ширини та висоти `wrap_content`. Текст встановлюється "Відгук:", розмір тексту — 16sp, стиль — жирний (bold). Встановлюється розташування нижче `ratingBar` (`layout_below`). Встановлюється верхній відступ (`marginTop`) у 8dp.

Лістинг 3.46 – Оголошення `TextView`

```

<TextView
    android:id="@+id/textViewCommentTitle"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Відгук:"
    android:textSize="16sp"
    android:textStyle="bold"
    android:layout_below="@id/ratingBar"
    android:layout_marginTop="8dp" />

```

Текстовий віджет для тексту відгуку. `TextView` для відображення тексту відгуку. Встановлюються параметри ширини та висоти `wrap_content`. Розмір тексту встановлюється у 14sp. Встановлюється розташування нижче `textViewCommentTitle` (`layout_below`). Вирівнювання по початку відносно `textViewCommentTitle` (`layout_alignStart`).

Лістинг 3.47 – Оголошення `TextView`

```

<TextView
    android:id="@+id/textViewCommentText"
    android:layout_width="wrap_content"

```

```

        android:layout_height="wrap_content"
        android:textSize="14sp"
        android:layout_below="@id/textViewCommentTitle"
        android:layout_alignStart="@id/textViewCommentTitle"
    />

```

Текстовий віджет для заголовка дати. `TextView` для відображення тексту "Дата:". Встановлюються параметри ширини та висоти `wrap_content`. Текст встановлюється "Дата:", розмір тексту — 14sp, стиль - курсив (*italic*). Встановлюється розташування нижче `textViewCommentText` (`layout_below`). Встановлюється верхній відступ (`marginTop`) у 8dp. Видимість встановлюється на `gone` (не відображається).

Лістинг 3.48 – Оголошення `TextView`

```

<TextView
    android:id="@+id/textViewDateAddedTitle"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Дата:"
    android:textSize="14sp"
    android:textStyle="italic"
    android:layout_below="@id/textViewCommentText"
    android:layout_marginTop="8dp"
    android:visibility="gone" />

```

Текстовий віджет для дати додавання. `TextView` для відображення дати додавання. Встановлюються параметри ширини та висоти `wrap_content`. Розмір тексту встановлюється у 14sp. Встановлюється розташування нижче `textViewDateAddedTitle` (`layout_below`). Вирівнювання по початку відносно `textViewDateAddedTitle` (`layout_alignStart`). Видимість встановлюється на `gone` (не відображається).

Лістинг 3.49 – Оголошення TextView

```

<TextView
    android:id="@+id/textViewDateAdded"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="14sp"
    android:layout_below="@id/textViewDateAddedTitle"

    android:layout_alignStart="@id/textViewDateAddedTitle"
    android:visibility="gone" />

```

Закриття тегу RelativeLayout:

```

</RelativeLayout>

```

Ця розмітка визначає, як виглядатиме кожен елемент списку коментарів, відображаючи оцінку, текст відгуку та дату додавання, з відповідними стилями та розташуванням елементів.

З метою відображення на екрані застосунку інформації про заправки електрокарів були застосовані наступні методи у класі FragmentMap:

Установка обробника натискання на маркери:

Лістинг 3.50 - Установка обробника натискання на маркери

```

googleMap.setOnMarkerClickListener(new
GoogleMap.OnMarkerClickListener() {
    @Override
    public boolean onMarkerClick(Marker marker) {
        // Отримуємо ChargingStation з тега маркера
        ChargingStation station = (ChargingStation)
marker.getTag();
        if (station != null) {
            // Показуємо інформацію про станцію
            showStationInfo(station);
        }
    }
});

```

```

        return true; // Обробили подію натискання на
маркері
    }
    return false; // Не вдалося обробити подію натискання
на маркері
}
});

```

Цей код встановлює обробник події натискання на маркери на мапі Google. Якщо маркер натиснуто, з нього отримується об'єкт `ChargingStation`, інформація про який відображається.

Метод для показу інформації про заправку електрокарів наведено на лістингу A.1 додатку A.

Даний метод відображає інформацію про заправку електрокарів у вигляді діалогового вікна. Включає середню оцінку та текстове повідомлення. Після натискання на кнопку "ОК" відкривається нова активність для перегляду відгуків.

Метод для відображення діалогу з відгуком та рейтингом наведено на лістингу A.1 додатку A.

Вказаний метод перевіряє, чи є переданий ідентифікатор заправки дійсним, і якщо так, то створює інтент для запуску активності `ReviewActivity`. Якщо ідентифікатор некоректний, відображається повідомлення про помилку.

Метод для додавання відгуку та рейтингу до бази даних наведено на лістингу A.1 додатку A.

Зазначений метод додає новий відгук і рейтинг для заправки в базу даних. Використовується конструкція `try-with-resources` для автоматичного закриття бази даних після завершення операції.

Метод, який обчислює середню оцінку на основі коментарів для заправки наведено на лістингу A.1 додатку A.

Цей метод обчислює середню оцінку на основі списку коментарів. Якщо коментарі відсутні, повертається 0.

Метод отримання коментарів для певної заправки в базі даних (Лістинг А.8, Додаток А) отримує список коментарів для конкретної заправки електрокарів з бази даних. Запит виконується для отримання всіх відгуків за заданим ідентифікатором заправки. Отримані дані додаються до списку `comments`, який потім повертається.

Вихідний код `frontend` мобільного застосунку можливо переглянути за посиланням [18].

3.2 Розробка архітектури системи (клієнт-сервер)

3.2.1 Загальні відомості про клієнт-сервер

Клієнт – комп'ютер на стороні користувача, який відправляє запит до сервера для надання інформації або виконання певних дій.

Сервер – більш потужний комп'ютер або обладнання, призначене для вирішення певних завдань з виконання програмних кодів, виконання сервісних функцій за запитом клієнтів, надання користувачам доступу до певних ресурсів, зберігання інформації і баз даних [12].

Модель такої системи полягає в тому, що клієнт відправляє запит на сервер, де він обробляється, і готовий результат відправляється клієнтові. Сервер може обслуговувати кілька клієнтів одночасно.

3.2.2 Spring Framework та Spring Boot

Spring Framework та Spring Boot - це інструменти для розробки мобільних застосунків на мові Java. Spring Framework є потужною платформою, яка пропонує багато функцій та інструментів, а Spring Boot - це додатковий інструмент, що дозволяє скоротити час та зусилля, необхідні для розгортання застосунків [13].

Особливості Spring Framework:

- дозволяє розробникам розглядати бізнес-логіку додатку як набір компонентів, які можна змінювати та розширювати без необхідності вносити зміни в код;
- підтримує аспектно-орієнтоване програмування, що дозволяє відокремити крос-концернні питання (cross-cutting concerns) від основного коду додатку;
- надає підтримку ORM, що дозволяє розробникам працювати з базою даних, використовуючи об'єктно-орієнтований підхід, що спрощує роботу з базою даних та зменшує кількість повторюваного коду;
- дозволяє розробникам розділити додаток на модулі та компоненти, що забезпечує легкість розробки та розширення функціоналу додатку.

У дипломній роботі використовується Spring Boot версії 2.0. Цей фреймворк забезпечує комплексну модель розробки і конфігурації додатку, створеного на Java, який можна використовувати на будь-яких платформах.

Spring Framework складається з модулів, що можуть використовуватися для різних завдань розробки. Наприклад, модуль Spring MVC використовується для розробки веб-додатків, модуль Spring Data допомагає в роботі з базами даних, а модуль Spring Security забезпечує безпеку веб-додатків [14].

Spring - підтримка інфраструктури на рівні програми: основна увага приділяється оформленню застосунку, тому розробка зосереджується саме на логіці без зайвих налаштувань, залежно від виконуваного завдання. Можливості цього фреймворка включають:

- впровадження залежностей.
- аспектно-орієнтоване програмування, включаючи декларативне управління транзакціями.

Spring Boot володіє багатим функціоналом, але його найбільш значущими особливостями є управління залежностями, автоматична конфігурація і вбудовані контейнери сервлетів [15].

Основна ідея фреймворку Spring полягає у використанні бінів (Beans) – це звичайні Java-об’єкти, але їх легка конфігурація робить використання дуже зручним. Таким чином контейнер сервлетів Spring Boot реалізовує принцип SOLID Dependency Inversion – ми створюємо всі біни з використанням патерну Singleton, що означає наявність лише одного об’єкта кожного класу на весь додаток. Однак, Spring також підтримує створення бінів типу prototype, що є породжуючим патерном і створює схожі об’єкти, частково копіюючи їх [16].

У роботі використовуємо анотації для конфігурації впровадження залежностей. Хоча конфігурацію можна виконувати і в XML, цей підхід вважається застарілим та менш зручним.

Анотація `@Service` створює бін з іменем `CommentsService`. Щоб використовувати цей бін, нам слід оголосити відповідний клас у іншому класі та впровадити залежність. Існує два способи впровадження залежностей – через конструктор або через сеттер. У нашому випадку ми використовуємо другий варіант – анотація `@Resource` знайде в контейнері бінів потрібні singleton-біни та впровадить залежності. Таким чином, Spring Boot реалізує один з основних принципів SOLID – інверсію залежностей.

Для того щоб в дипломній роботі використовувати Spring Data JPA для доступу до бази даних, включаємо в проект залежність `spring-boot-starter-data-jpa` (рис. 3.3).

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
  <version>1.5.22.RELEASE</version>
</dependency>
```

Рисунок 3.3 – Використання pom.xml конфігурації для бібліотек

3.2.3 REST API

Для створення зв'язку в дипломній роботі використовується архітектурний стиль REST API, який забезпечує CRUD операції з базою даних (рис. 3.4). Цей підхід обрано завдяки його простоті, прозорості, легкості та зрозумілості.

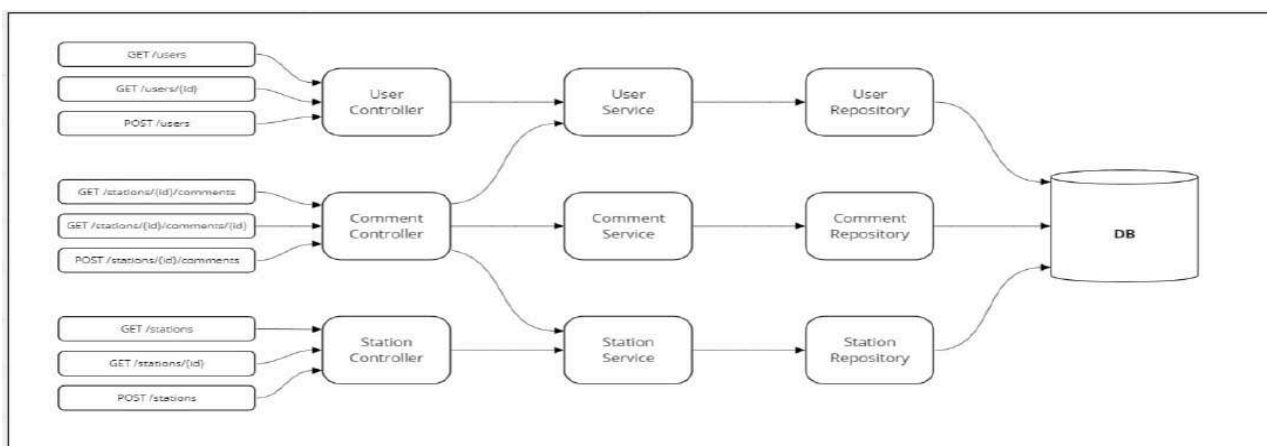


Рисунок 3.4 – Структура елементів на backend

REST (Representational State Transfer) - це архітектурний стиль для побудови розподілених веб-систем. REST API - це інтерфейс, який застосовує REST-архітектуру, що дозволяє взаємодіяти з сервером через стандартні HTTP-методи, такі як GET, POST, PUT, PATCH та DELETE [17].

Основні принципи REST API:

- кожен ресурс має бути доступний через унікальний URI (Uniform Resource Identifier);
- клієнт-серверна архітектура: система повинна бути поділена на клієнтів та сервери, які можуть розвиватися незалежно один від одного;
- використання стандартних HTTP-методів (GET, POST, PUT, PATCH, DELETE) для взаємодії з ресурсами;
- REST API має бути без стану, тобто сервер не зберігає жодної інформації про клієнта між запитами.

Однією з ключових переваг REST API є його простота використання та можливість інтеграції з іншими системами. REST API можна використовувати

для передачі даних між різними платформами, що робить його ідеальним для побудови мікросервісних архітектур. REST API дозволяє зменшити час розробки, спростити інтеграцію з іншими системами та забезпечити стабільну й ефективну взаємодію між різними сервісами.

Якщо розглядати детальніше, то REST (RESTful) – це архітектурний стиль, що базується на загальних принципах організації взаємодії додатків із серверами, які підтримують протокол HTTP. Особливістю REST є те, що сервер не запам'ятовує стан користувача між запитами - у кожному окремому запиті передається інформація, ідентифікована користувачем, наприклад, маркер, отриманий через OAuth-авторизацію, та всі параметри, необхідні для виконання операцій.

Взаємодія між клієнтом та сервером у REST API здійснюється за допомогою HTTP-запитів. Клієнт може робити запити на сервер, який надає необхідну інформацію у відповідь.

Для цього використовуються стандартні HTTP-методи:

- GET: запит на отримання ресурсу;
- POST: запит на створення ресурсу;
- PUT: запит на зміну ресурсу;
- PATCH: запит на часткову зміну ресурсу;
- DELETE: запит на видалення ресурсу.

У відповідь на запит сервер повертає HTTP-статус код, який вказує на стан виконання запиту, та відповідь у форматі, зазначеному в HTTP-заголовках запиту (наприклад, JSON або XML). Відповідь може містити необхідну інформацію про запрошений ресурс, його стан, а також може включати інформацію про можливі наступні кроки.

3.2.4 Реалізація серверної частини мобільного застосунку

Створення моделі для сутностей User, Station та Comment з використанням Java Persistence API (JPA) визначає три класи-моделі. Ці моделі відповідають таблицям у базі даних і включають основні поля для кожної сутності, а також необхідні анотації для роботи з ORM (Object-Relational Mapping).

Лістинг 3.51 - Клас User

```
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String email;

    // Getters and Setters
}
```

@Entity: Анотація, яка вказує, що цей клас є сутністю JPA. Це означає, що клас буде відображатися в таблицю бази даних.

@Id: Поле `id` є первинним ключем для сутності.

@GeneratedValue(strategy = GenerationType.IDENTITY): Значення первинного ключа генерується автоматично базою даних.

Поля `name` і `email`: Звичайні поля, що зберігають інформацію про ім'я та електронну пошту користувача.

Getters і Setters: Методи для доступу до полів та їх модифікації (не показані тут, але мають бути реалізовані).

Лістинг 3.52 - Клас Station

```
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
```

```

@Entity
public class Station {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;

    // Getters and Setters
}

```

@Entity: Вказує, що клас Station є сутністю JPA.

@Id: Поле id є первинним ключем для сутності.

@GeneratedValue(strategy = GenerationType.IDENTITY): Генерація значення id автоматично базою даних.

Поле name: Звичайне поле, що зберігає ім'я станції.

Getters і Setters: Методи для доступу до полів та їх модифікації.

Лістинг 3.53 - Клас Comment

```

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Comment {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private Long stationId;
    private String content;

    // Getters and Setters
}

```

@Entity: Вказує, що клас Comment є сутністю JPA.

@Id: Поле id є первинним ключем для сутності.

@GeneratedValue(strategy = GenerationType.IDENTITY): Генерація значення id автоматично базою даних.

Поле stationId: Поле, що зберігає ідентифікатор станції, до якої відноситься коментар.

Поле content: Звичайне поле, що зберігає текст коментаря.

Getters і Setters: Методи для доступу до полів та їх модифікації.

Створення репозиторіїв здійснюється за допомогою коду, який визначає три інтерфейси репозиторіїв, які використовують Spring Data JPA для взаємодії з базою даних. Репозиторії забезпечують стандартні операції CRUD (створення, читання, оновлення, видалення) для сутностей User, Station та Comment. Крім того, CommentRepository включає метод для отримання коментарів за ідентифікатором станції.

Лістинг 3.54 - Інтерфейс UserRepository

```
import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<User,
Long> {}
```

extends JpaRepository<User, Long>: Інтерфейс успадковує від JpaRepository, що надає базові методи для роботи з сутністю User.

User: Тип сутності, з якою працює цей репозиторій.

Long: Тип первинного ключа сутності User.

Основні методи, що надаються JpaRepository:

findAll(): Отримує всі записи сутності.

findById(Long id): Знаходить запис за ідентифікатором.

save(User user): Зберігає або оновлює запис.

deleteById(Long id): Видаляє запис за ідентифікатором.

Лістинг 3.55 - Інтерфейс StationRepository

```
import org.springframework.data.jpa.repository.JpaRepository;

public interface StationRepository extends
JpaRepository<Station, Long> {}
```

`extends JpaRepository<Station, Long>`: Інтерфейс успадковує від `JpaRepository`, що надає базові методи для роботи з сутністю `Station`.

`Station`: Тип сутності, з якою працює цей репозиторій.

`Long`: Тип первинного ключа сутності `Station`.

Лістинг 3.56 - Інтерфейс CommentRepository

```
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.List;

public interface CommentRepository extends
JpaRepository<Comment, Long> {
    List<Comment> findByStationId(Long stationId);
}
```

`extends JpaRepository<Comment, Long>`: Інтерфейс успадковує від `JpaRepository`, що надає базові методи для роботи з сутністю `Comment`.

`Comment`: Тип сутності, з якою працює цей репозиторій.

`Long`: Тип первинного ключа сутності `Comment`.

Додатковий метод `findByStationId(Long stationId)`:

Цей метод визначається користувачем для отримання списку коментарів, які належать до певної станції (визначається ідентифікатором станції).

`List<Comment> findByStationId(Long stationId)`: Метод знаходить всі коментарі, де поле `stationId` відповідає переданому значенню.

Створення контролерів здійснюється за допомогою коду, який визначає два контролери Spring Boot: `UserController` та `StationController`. Вони забезпечують REST API для управління користувачами, станціями та

коментарями. Контролери використовують відповідні репозиторії для взаємодії з базою даних і надають кінцеві точки (ендпоінти) для різних операцій.

Лістинг 3.57 - Контролер UserController

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Optional;

@RestController
@RequestMapping("/users")
public class UserController {
    @Autowired
    private UserRepository userRepository;

    @GetMapping
    public List<User> getAllUsers() {
        return userRepository.findAll();
    }

    @GetMapping("/{id}")
    public ResponseEntity<User> getUserById(@PathVariable Long
id) {
        Optional<User> user = userRepository.findById(id);
        return user.map(ResponseEntity::ok).orElseGet(() ->
ResponseEntity.notFound().build());
    }

    @PostMapping
    public User createUser(@RequestBody User user) {
        return userRepository.save(user);
    }
}
```

- `@RestController`: Анотація, що вказує, що цей клас є контролером REST;
- `@RequestMapping("/users")`: Вказує базовий шлях для всіх кінцевих точок у цьому контролері;

- `@Autowired`: Анотація для автоматичного впровадження залежностей.

Методи:

- `getAllUsers()`: Обробляє GET-запити на `/users` і повертає список всіх користувачів;

- `getUserById(Long id)`: Обробляє GET-запити на `/users/{id}` і повертає користувача за ідентифікатором або 404, якщо користувача не знайдено;

- `createUser(User user)`: Обробляє POST-запити на `/users` і створює нового користувача.

Контролер `StationController`:

- `@RestController`: Анотація, що вказує, що цей клас є контролером REST;
- `@RequestMapping("/stations")`: Вказує базовий шлях для всіх кінцевих точок у цьому контролері;

- `@Autowired`: Анотація для автоматичного впровадження залежностей.

наведено на лістингу Б.2 додатку Б.

Методи:

- `getAllStations()`: Обробляє GET-запити на `/stations` і повертає список всіх станцій;

- `getStationById(Long id)`: Обробляє GET-запити на `/stations/{id}` і повертає станцію за ідентифікатором або 404, якщо станцію не знайдено;

- `createStation(Station station)`: Обробляє POST-запити на `/stations` і створює нову станцію;

- `getCommentsByStationId(Long id)`: Обробляє GET-запити на `/stations/{id}/comments` і повертає список коментарів для вказаної станції;

- `getCommentById(Long stationId, Long id)`: Обробляє GET-запити на `/stations/{stationId}/comments/{id}` і повертає коментар за ідентифікатором або 404, якщо коментар не знайдено;

- `createComment(Long id, Comment comment)`: Обробляє POST-запити на `/stations/{id}/comments` і створює новий коментар для вказаної станції.

Spring Boot REST Controllers це контролери, які забезпечують RESTful API, що дозволяє клієнтам (наприклад, веб-додаткам або мобільним додаткам) взаємодіяти з сервером через HTTP-запити.

`ResponseEntity`: Клас для роботи з HTTP-відповідями, дозволяє налаштовувати статус код і тіло відповіді.

`@PathVariable`: Анотація для отримання змінних з URL-адреси.

`@RequestBody`: Анотація для отримання даних з тіла HTTP-запиту.

Ці контролери використовують репозиторії для отримання даних з бази даних, забезпечуючи виконання CRUD операцій для користувачів, заправок та коментарів.

Запуск додатку є основною точкою входу для Spring Boot додатка. Він визначає клас `Application`, який використовується для запуску додатка.

Лістинг 3.58 - Клас `Application`

```
import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

Імпортовані пакети:

- `import org.springframework.boot.SpringApplication;`
- `import org.springframework.boot.autoconfigure.SpringBootApplication;`
- `springApplication`: Клас, який забезпечує зручний спосіб запуску додатків Spring.

- `springBootApplication`: Анотація, яка об'єднує кілька ключових анотацій для налаштування Spring Boot додатка.

Анотація `@SpringBootApplication`:

- `@SpringBootApplication`;
- `@SpringBootApplication`.

Це складна анотація, яка включає в себе:

- `@Configuration`: Вказує, що клас містить конфігурацію для Spring контексту;

- `@EnableAutoConfiguration`: Дозволяє Spring Boot автоматично налаштовувати контекст додатка на основі залежностей, які визначені в classpath;

- `@ComponentScan`: Дозволяє Spring сканувати інші компоненти, конфігурації та сервіси у поточному пакеті та підпакетах.

`public class Application`: Основний клас додатка.

`public static void main(String[] args)`: Головний метод, який використовується JVM для запуску програми.

`SpringApplication.run(Application.class, args)` - метод, який запускає Spring Boot додаток.

3.2.5 Інтерфейс користувача мобільного застосунку

В рамках розробленого проекту було реалізовано інтерфейс застосунку, який передбачає взаємодію з користувачем через вікна входу, відображення мапи, вибору заправки та можливості залишити відгук і рецензію.

Після запуску мобільного застосунку користувач потрапляє на сторінку входу, де розміщена необхідна інформація про застосунок та кнопка «Start» (рис. 3.5).

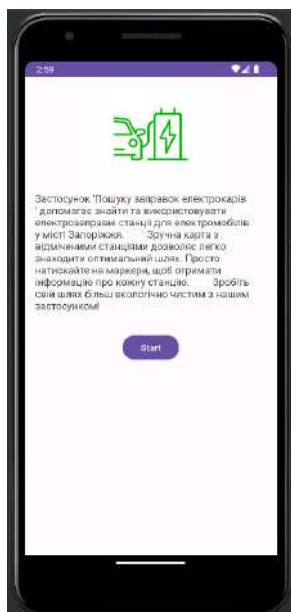


Рисунок 3.5 – Вхід користувача до мобільного застосунку

Коли користувач натискає кнопку «Start», він переходить на екран з мапою, де він бачить маркери місцезнаходження заправок електрокарів (рис. 3.6).

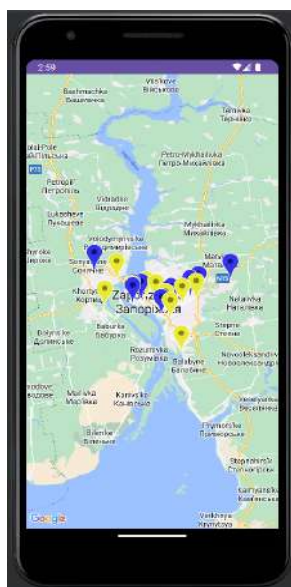


Рисунок 3.6 – Вхід користувача до мапи розташування заправок

Визначившись з потрібною заправкою, користувач має натиснути на маркер заправки, після чого з'являється вікно з детальною інформацією про заправку електрокарів та середню оцінку користувачів (рис. 3.7).

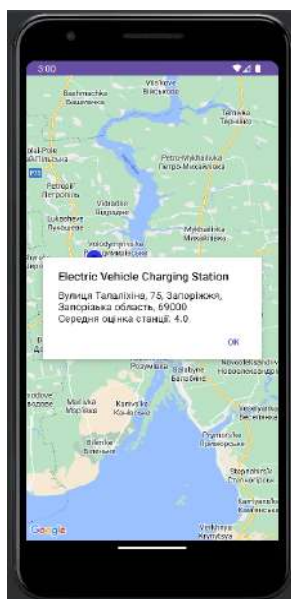


Рисунок 3.7 – Вибір заправки користувачем

В інформаційному вікні мобільного застосунку користувач повинен натиснути кнопку «Ок». Після цього він переходить до екрану, де може залишити особистий відгук про заправку та поставити оцінку.

Для цього потрібно заповнити форму «Зірочки» та поле «Введіть Ваш відгук» та натиснути кнопку «Відправити відгук».

Якщо користувач не має бажання залишати відгук, він натискає кнопку «Вийти» та повертається до початкового екрану з мапою розташування заправок електрокарів (рис. 3.8).

За допомоги кнопки «Переглянути попередні коментарі» користувач має можливість переглянути попередні коментарі користувачів та їхні оцінки.

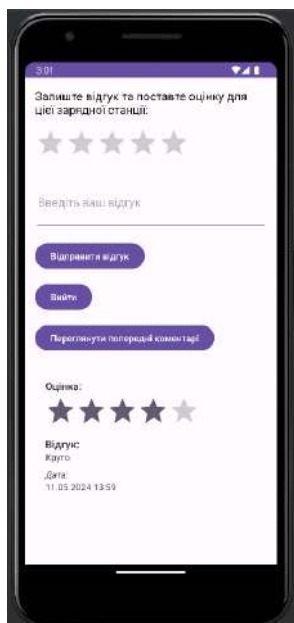


Рисунок 3.8 – Екран для відгуків та оцінок

У підсумку, маємо плакат інтерфейсу мобільного застосунку для пошуку заправок електрокарів.

Розроблений застосунок повністю відповідає реалізації поставленої задачі та має зручний, інтуїтивно зрозумілий інтерфейс.

ВИСНОВКИ

Зарядні заправки (заправки) представляють собою повноцінних учасників ринку надання послуг, для споживачів, які є власниками електромобілей (електрокарів) та потребують постійного оновлення, введення різних оптимізуючих ресурсів та новацій.

Сучасні гаджети (смартфони, планшети) стали незамінними учасниками загальної життя кожної людини, включаючи власників електротранспорту.

Розробка та створення мобільного застосунку (застосунку) – це один із способів залучення уваги до мереж заправок електрокарів, поліпшення позицій бренду на ринку, збільшення довіри з боку старих клієнтів та залучення нових.

Також, це гарна можливість використовувати інформаційну середу, для вдосконалення спілкування з відповідною аудиторією споживачів.

Об'єкт розробки дипломної роботи бакалавра – це розробка мобільного застосунку для пошуку заправок електрокарів.

В результаті роботи був реалізований мобільний застосунок для пошуку заправок електрокарів. Реалізований застосунок можна використовувати для вирішення особистих задач при подорожуванні, так і в комерційних цілях.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Гід по електромобілях: усе, що тобі варто знати: вебсайт. URL: <https://bolt.eu/uk-ua/blog/electric-cars/> (дата звернення: 27.04.2024)
2. В Україні вже зараз не вистачає швидкісних зарядок для авто. Коли буде подоланий дефіцит: вебсайт. URL: <https://delo.ua/energetics/v-ukrayini-vze-zaraz-ne-vistacaje-svidkisnix-zaryadok-dlya-avto-koli-bude-podolanii-deficit-430557/> (дата звернення: 29.04.2024)
3. Мобільний застосунок це? що таке, види, типи та особливості: вебсайт. URL: <https://it-rating.ua/mobilniy-dodatok-tse-scho-take-vidi-tipi-ta-osoblivosti> (дата звернення: 29.04.2024)
4. Карта електричних заправок для електромобілів: вебсайт. URL: <https://toka.energy/uk/chargemap/> (дата звернення: 30.04.2024)
5. Мобільний застосунок Go To-U: як заряджати електромобіль з максимальним комфортом: вебсайт. URL: <https://tesla-club.com.ua/news/us/mobilniy-dodatok-go-to-u-yak-zaryadzhati-elektromobil-z-maksimalnim-komfortom> (дата звернення: 04.05.2024)
6. Мобільний застосунок UGV Chargers: вебсайт. URL: <https://ugv.ua/uk/mobilnyy-dodatok/> (дата звернення: 06.05.2024)
7. Community-based ecosystem for all your charging needs: вебсайт. URL: <https://oncharge.net/> (дата звернення: 09.05.2024)
8. Зарядні заправки ТОКА: вебсайт. URL: <https://toka.energy/uk/app/> (дата звернення: 28.04.2024)
9. Android Studio: переваги та особливості: вебсайт. URL: <https://qagroup.com.ua/publications/android-studio-perevagy-ta-osoblyvosti/> (дата звернення: 29.04.2024)
10. Що таке Java і де вона використовується: вебсайт. URL: <https://goit.global/ua/articles/shcho-take-java-i-de-vona-vykorystovuietsia/> (дата звернення: 14.05.2024)

11. Що таке Sqlite?: вебсайт. URL: <https://freehost.com.ua/ukr/faq/articles/chto-takoe-sqlite/> (дата звернення: 15.05.2024)
12. Клієнт-серверна архітектура: вебсайт. URL: <https://training.qatestlab.com/blog/technical-articles/client-server-architecture/> (дата звернення: 20.05.2024)
13. "Getting started with Spring Framework": вебсайт. URL: <https://spring.io/guides/gs/spring-framework/> (дата звернення: 21.05.2024)
14. "Spring Framework Tutorial": вебсайт. URL: <https://www.tutorialspoint.com/spring/index.htm> (дата звернення: 21.05.2024)
15. Spring Boot Documentation: вебсайт. URL: <https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/> (дата звернення: 23.05.2024)
16. "Getting started with Spring Boot" by Baeldung: вебсайт. URL: <https://www.baeldung.com/spring-boot-start> (дата звернення: 23.05.2024)
17. "What is REST API?". Mulesoft.: вебсайт. URL: <https://www.mulesoft.com/resources/api/what-is-rest-api-design> (дата звернення: 23.05.2024)
18. GitHub URL: <https://github.com/Hebidomuy/MyFirstApp> (дата звернення: 23.05.2024)

Додаток А Код головного класу FragmentMap

Лістинг А.1 – Головний клас frontend

```
package com.daniil.myapplication;

import static
com.daniil.myapplication.database.DatabaseHelper.COLUMN_DATE_ADDED;
import static
com.daniil.myapplication.database.DatabaseHelper.COLUMN_ID;
import static
com.daniil.myapplication.database.DatabaseHelper.COLUMN_RATING;
import static
com.daniil.myapplication.database.DatabaseHelper.COLUMN_REVIEW_TEX
T;
import static
com.daniil.myapplication.database.DatabaseHelper.COLUMN_STATION_ID;
import static
com.daniil.myapplication.database.DatabaseHelper.TABLE_REVIEWS;

import android.content.DialogInterface;
import android.content.Intent;
import android.database.Cursor;
import android.os.Bundle;

import androidx.appcompat.app.AlertDialog;
import androidx.fragment.app.Fragment;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Toast;

import com.daniil.myapplication.database.DatabaseHelper;
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
```

```

import com.google.android.gms.maps.MapView;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.model.BitmapDescriptorFactory;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.Marker;
import com.google.android.gms.maps.model.MarkerOptions;

import android.database.sqlite.SQLiteDatabase;

import java.util.ArrayList;
import java.util.List;
import java.util.Locale;

public class FragmentMap extends Fragment implements
OnMapReadyCallback {

    private MapView mapView;
    private GoogleMap googleMap;
    private List<ChargingStation> chargingStations = new
ArrayList<>();
    private DatabaseHelper dbHelper;
    private SQLiteDatabase db;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup
container, Bundle savedInstanceState) {
        View rootView = inflater.inflate(R.layout.fragment_map,
container, false);
        // Ініціалізація та налаштування MapView
        mapView = rootView.findViewById(R.id.mapView);
        mapView.onCreate(savedInstanceState);
        mapView.getMapAsync(this);

        // Отримання екземпляра бази даних
        dbHelper = new DatabaseHelper(getActivity());

```

```

        db = dbHelper.getWritableDatabase();

        return rootView;
    }

    @Override
    public void onMapReady(GoogleMap map) {
        googleMap = map;

        // Налаштування початкового виду карти
        float zoomLevel = 10.0f;
        LatLng location = new LatLng(47.83941, 35.14583);    //
        Координати (м. Запоріжжя)

        googleMap.moveCamera(CameraUpdateFactory.newLatLngZoom(location,
        zoomLevel));

        // Додавання зарядних станцій
        addChargingStation("station1",      47.88833,      35.02139,
        "Electric Vehicle Charging Station", "Вулиця Талаліхіна, 75,
        Запоріжжя,          Запорізька          область,          69000",
        BitmapDescriptorFactory.HUE_BLUE);
        addChargingStation("station2",      47.87497,      35.06959,
        "AutoEnterprise 271 Charging Station", "Бульвар Вінтера, 30,
        Запоріжжя,          Запорізька          область,          69061",
        BitmapDescriptorFactory.HUE_YELLOW);
        addChargingStation("station3",      47.87436,      35.31275,
        "Electric Vehicle Charging Station", "Вулиця Зразкова, 2Б, Запоріжжя,
        Запорізька область, 69000", BitmapDescriptorFactory.HUE_BLUE);
        addChargingStation("station4",      47.83258,      35.04417, "АЕ
        Charging Station", "м. Запоріжжя, вул. Заднепровская, 31 Запоріжжя
        Zaporizhia          Oblast          69000,          69499",
        BitmapDescriptorFactory.HUE_YELLOW);
    }

```

```

        addChargingStation("station5", 47.84299, 35.10380, "UGV
Chargers Charging Station", "Бульвар Шевченка, 56, Запоріжжя,
Запорізька область, 69061", BitmapDescriptorFactory.HUE_BLUE);

        addChargingStation("station6", 47.84085, 35.10165, "UGV
Chargers", "Бульвар Шевченка, 71, Запоріжжя, Запорізька область,
69000", BitmapDescriptorFactory.HUE_YELLOW);

        addChargingStation("station7", 47.83792, 35.10373, "АЕ
Charging Station", "Вулиця Немировича-Данченка, 73, Запоріжжя,
Запорізька область, 69000", BitmapDescriptorFactory.HUE_BLUE);

        addChargingStation("station8", 47.84713, 35.13265,
"Elecargroup", "Проспект Соборний, 147, Запоріжжя, Запорізька
область, 69000", BitmapDescriptorFactory.HUE_YELLOW);

        addChargingStation("station9", 47.84386, 35.12180, "АЕ
Charging Station", "Вулиця Перемоги, Запоріжжя, Запорізька область,
69000, 69499", BitmapDescriptorFactory.HUE_BLUE);

        addChargingStation("station10", 47.84305, 35.15343,
"Electric Vehicle Charging Station", "Вулиця Перемоги, 90, Запоріжжя,
Запорізька область, 69000", BitmapDescriptorFactory.HUE_YELLOW);

        addChargingStation("station11", 47.85546, 35.24466,
"Electric Vehicle Charging Station", "Донецьке шосе, 7, Запоріжжя,
Запорізька область, 69061", BitmapDescriptorFactory.HUE_BLUE);

        addChargingStation("station12", 47.84439, 35.23981,
"Electric Vehicle Charging Station", "Автомагістраль Харків-
Сімферополь, Запоріжжя, Запорізька область, 69000",
BitmapDescriptorFactory.HUE_YELLOW);

        addChargingStation("station13", 47.84927, 35.22670,
"AutoEnterprise 329", "155А,, Чарівна вулиця, 155А, Запоріжжя,
Запорізька область", BitmapDescriptorFactory.HUE_BLUE);

        addChargingStation("station14", 47.83656, 35.20971,
"Elecargroup Charging Station", "Вулиця Іванова, 24, Запоріжжя,
Запорізька область, 69000", BitmapDescriptorFactory.HUE_YELLOW);

        addChargingStation("station15", 47.83689, 35.18591,
"AutoEnterprise 272", "Вулиця Іванова, 1, Запоріжжя, Запорізька
область, 69061", BitmapDescriptorFactory.HUE_BLUE);

```

```

        addChargingStation("station16", 47.82823, 35.17453, "АЕ
Charging Station", "м, вулиця Олександрівська, 75, Запоріжжя,
Запорізька область, 69002", BitmapDescriptorFactory.HUE_YELLOW);
        addChargingStation("station17", 47.82008, 35.16530, "UGV
Chargers. Електрозаправна станція", "Вулиця Фортечна, 2, Запоріжжя,
Запорізька область, 69001", BitmapDescriptorFactory.HUE_BLUE);
        addChargingStation("station18", 47.81486, 35.18424,
"Electric Vehicle Charging Station", "Вулиця Святого Миколая, 6,
Запоріжжя, Запорізька область, 69061",
BitmapDescriptorFactory.HUE_YELLOW);
        addChargingStation("station19", 47.76358, 35.20721,
"Зарядна станція для електромобіля", "Технікумівська вулиця, 6А,
Запоріжжя, Запорізька область, 69000",
BitmapDescriptorFactory.HUE_BLUE);
        addChargingStation("station20", 47.76348, 35.20775,
"AutoEnterprise 326", "Технікумівська вулиця, 6А, Запоріжжя,
Запорізька область, 69000", BitmapDescriptorFactory.HUE_YELLOW);

        addMarkers();

        // Установка обробника натискання на маркери
        googleMap.setOnMarkerClickListener(new
        GoogleMap.OnMarkerClickListener() {
            @Override
            public boolean onMarkerClick(Marker marker) {
                // Отримуємо ChargingStation з тега маркера
                ChargingStation station = (ChargingStation)
marker.getTag();
                if (station != null) {
                    // Показуємо інформацію про станцію
                    showStationInfo(station);
                    return true; // Обробили подію натискання на
маркери
                }
            }
        });

```

```

        return false; // Не вдалося обробити подію
натискання на маркері
    }
});
}

// Додавання інформації про зарядні станції
private void addChargingStation(String stationId, double
latitude, double longitude, String title, String snippet, float
markerColor) {
    chargingStations.add(new ChargingStation(stationId, title,
snippet, new LatLng(latitude, longitude), markerColor));
}

// Додавання маркерів на карту
private void addMarkers() {
    for (ChargingStation station : chargingStations) {
        MarkerOptions markerOptions = new MarkerOptions()
            .position(station.getLocation())
            .title(station.getTitle())
            .snippet(station.getSnippet())
            .icon(BitmapDescriptorFactory.defaultMarker(st
ation.getMarkerColor()));

        Marker marker = googleMap.addMarker(markerOptions);
        marker.setTag(station);
    }
}

// Метод для показу інформації про станцію
private void showStationInfo(ChargingStation station) {
    // Отримуємо коментарі для станції
    List<Comment> comments =
getCommentsForStation(station.getStationId());
}

```

```

// Обчислюємо середню оцінку
float averageRating = calculateAverageRating(comments);

// Створюємо рядок для відображення середньої оцінки
String ratingString = String.format(Locale.getDefault(),
"Середня оцінка станції: %.1f", averageRating);

// Створюємо діалогове вікно
AlertDialog.Builder builder = new
AlertDialog.Builder(getActivity());
builder.setTitle(station.getTitle())
.setMessage(station.getSnippet() + "\n" +
ratingString)
.setPositiveButton("OK", new
DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int
id) {
        // Після закриття діалогового вікна
створюємо інтеніт для запуску ReviewActivity
        Intent intent = new Intent(getActivity(),
ReviewActivity.class);
        intent.putExtra("selectedStationId",
station.getStationId()); // Передача ідентифікатора обраної станції
        startActivity(intent);
        dialog.dismiss();
    }
});
AlertDialog dialog = builder.create();
dialog.show();
}

// Метод для відображення діалогу з відгуком та рейтингом
private void showReviewDialog(String stationId) {
    // Проверяем, что stationId не null и не пустой
    if (stationId != null && !stationId.isEmpty()) {

```

```

        // Создание Intent для запуска ReviewActivity
        Intent intent = new Intent(getActivity(),
ReviewActivity.class);

        intent.putExtra("station_id", stationId); // Передача
ідентифікатора обраної заправки
        startActivity(intent);
    } else {
        // Якщо stationId некоректний, виводимо повідомлення про
помилку

        Toast.makeText(getActivity(), "Ошибка: Некоректний
ідентифікатор станції", Toast.LENGTH_SHORT).show();
    }
}

// Метод для додавання відгуку та рейтингу до бази даних
private void addReviewAndRating(String stationId, String
reviewText, float rating) {
    // Використання інструкції try-with-resources для
автоматичного закриття бази даних
    try (DatabaseHelper dbHelper = new
DatabaseHelper(getActivity())) {
        dbHelper.addReviewAndRating(stationId, reviewText,
rating);

        // Повідомлення користувача про успішне додавання
відгуку та рейтингу
        Toast.makeText(getActivity(), "Відгук успішно додано!",
Toast.LENGTH_SHORT).show();
    } catch (Exception e) {
        // Обробка винятків, якщо щось пішло не так
        e.printStackTrace();
    }
}

// Метод який обчислює середню оцінку на основі коментарів для
станції

```

```

private float calculateAverageRating(List<Comment> comments) {
    float sum = 0;
    if (comments != null && !comments.isEmpty()) {
        for (Comment comment : comments) {
            sum += comment.getRating();
        }
        return sum / comments.size();
    } else {
        return 0; // Повертаємо 0, якщо немає коментарів
    }
}

// Метод отримання коментарів для певної станції в базі даних.
public List<Comment> getCommentsForStation(String stationId) {
    List<Comment> comments = new ArrayList<>();
    // Отримання бази даних для читання
    SQLiteDatabase db = this.db;

    // Визначення стовпців, які необхідно отримати
    String[] columns = {COLUMN_ID, COLUMN_RATING,
        COLUMN_REVIEW_TEXT, COLUMN_DATE_ADDED};

    // Визначення умови вибірки - станція з вказаним
    ідентифікатором
    String selection = COLUMN_STATION_ID + "=?";
    String[] selectionArgs = {stationId};

    // Виконання запиту до бази даних
    Cursor cursor = db.query(TABLE_REVIEWS, columns, selection,
        selectionArgs, null, null, null);

    // Перевірка, чи є дані в курсорі
    if (cursor != null && cursor.moveToFirst()) {
        do {
            // Отримання даних з курсора

```

```

        int                id                =
cursor.getInt(cursor.getColumnIndex(COLUMN_ID));

        float              rating            =
cursor.getFloat(cursor.getColumnIndex(COLUMN_RATING));

        String             reviewText       =
cursor.getString(cursor.getColumnIndex(COLUMN_REVIEW_TEXT));

        long               timestamp        =
cursor.getLong(cursor.getColumnIndex(COLUMN_DATE_ADDED));

        // Додавання коментаря до списку
        comments.add(new Comment(id, rating, reviewText,
String.valueOf(timestamp)));
    } while (cursor.moveToNext());

    // Закриття курсора після використання
    cursor.close();
}

// Повертаємо список коментарів для станції
return comments;
}

// Реалізація методів життєвого циклу MapView
@Override
public void onResume() {
    super.onResume();
    mapView.onResume();
}

@Override
public void onPause() {
    super.onPause();
    mapView.onPause();
}

```

```
@Override
public void onDestroy() {
    super.onDestroy();
    mapView.onDestroy();
    dbHelper.close();
}

@Override
public void onLowMemory() {
    super.onLowMemory();
    mapView.onLowMemory();
}
}
```

Додаток Б Фрагмент коду класу CommentsAdapter

Лістинг Б.1 -.Методи onCreateViewHolder та onBindViewHolder

```

    @NonNull
    @Override
    public CommentViewHolder onCreateViewHolder(@NonNull
    ViewGroup parent, int viewType) {
        View view =
        LayoutInflater.from(parent.getContext()).inflate(R.layout.comment_
        item, parent, false);
        return new CommentViewHolder(view);
    }

    @Override
    public void onBindViewHolder(@NonNull CommentViewHolder
    holder, int position) {
        // Отримуємо коментар за позицією
        Comment comment = commentsList.get(position);

        // Встановлюємо текст коментаря
        holder.textViewComment.setText(comment.getReviewText());

        // Встановлюємо оцінку та видимість відповідних
        елементів
        if (comment.getRating() > 0) {
            holder.textViewRating.setVisibility(View.VISIBLE);
            holder.ratingBar.setVisibility(View.VISIBLE);
            holder.ratingBar.setRating(comment.getRating());
        } else {
            holder.textViewRating.setVisibility(View.GONE);
            holder.ratingBar.setVisibility(View.GONE);
        }
    }

```

```

        // Встановлюємо дату та видимість відповідних елементів
        if (comment.getDateAdded() != null &&
!comment.getDateAdded().isEmpty()) {

holder.textViewDateAddedTitle.setVisibility(View.VISIBLE);

holder.textViewDateAdded.setVisibility(View.VISIBLE);

        // Перетворення мілісекунд в Date
        Date date = new
Date(Long.parseLong(comment.getDateAdded()));

        // Форматування дати згідно з новим форматом
        SimpleDateFormat outputFormat = new
SimpleDateFormat("dd.MM.yyyy HH:mm", Locale.getDefault());
        String formattedDate = outputFormat.format(date);

        // Встановлення форматованої дати в текстове поле
        holder.textViewDateAdded.setText(formattedDate);
    } else {

holder.textViewDateAddedTitle.setVisibility(View.GONE);

holder.textViewDateAdded.setVisibility(View.GONE);

    }
}

```

Лістинг Б.2 -. Контролер StationController

```

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Optional;

```

```

@RestController
@RequestMapping("/stations")
public class StationController {
    @Autowired
    private StationRepository stationRepository;
    @Autowired
    private CommentRepository commentRepository;

    @GetMapping
    public List<Station> getAllStations() {
        return stationRepository.findAll();
    }
    @GetMapping("/{id}")
    public ResponseEntity<Station>
getStationById(@PathVariable Long id) {
        Optional<Station> station =
stationRepository.findById(id);
        return station.map(ResponseEntity::ok).orElseGet(() ->
ResponseEntity.notFound().build());
    }

    @PostMapping
    public Station createStation(@RequestBody Station station)
{
        return stationRepository.save(station);
    }

    @GetMapping("/{id}/comments")
    public List<Comment> getCommentsByStationId(@PathVariable
Long id) {
        return commentRepository.findByStationId(id);
    }

    @GetMapping("/{stationId}/comments/{id}")

```

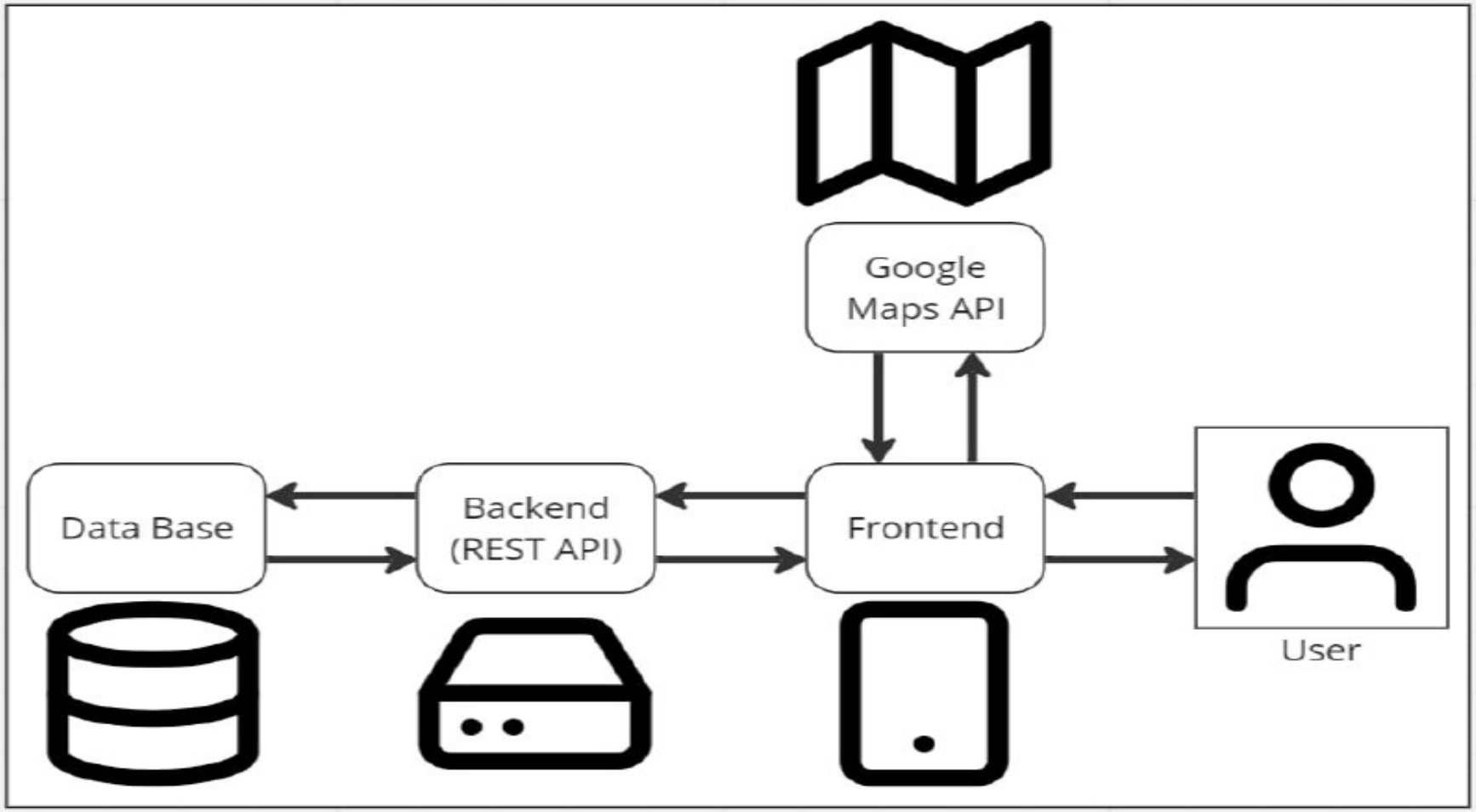
```

        public                               ResponseEntity<Comment>
getCommentById(@PathVariable Long stationId, @PathVariable Long id)
{
        Optional<Comment>                               comment =
commentRepository.findById(id);
        return comment.map(ResponseEntity::ok).orElseGet(() ->
ResponseEntity.notFound().build());
}

        @PostMapping("/{id}/comments")
        public Comment createComment(@PathVariable Long id,
@RequestBody Comment comment) {
        comment.setStationId(id);
        return commentRepository.save(comment);
}
}

```

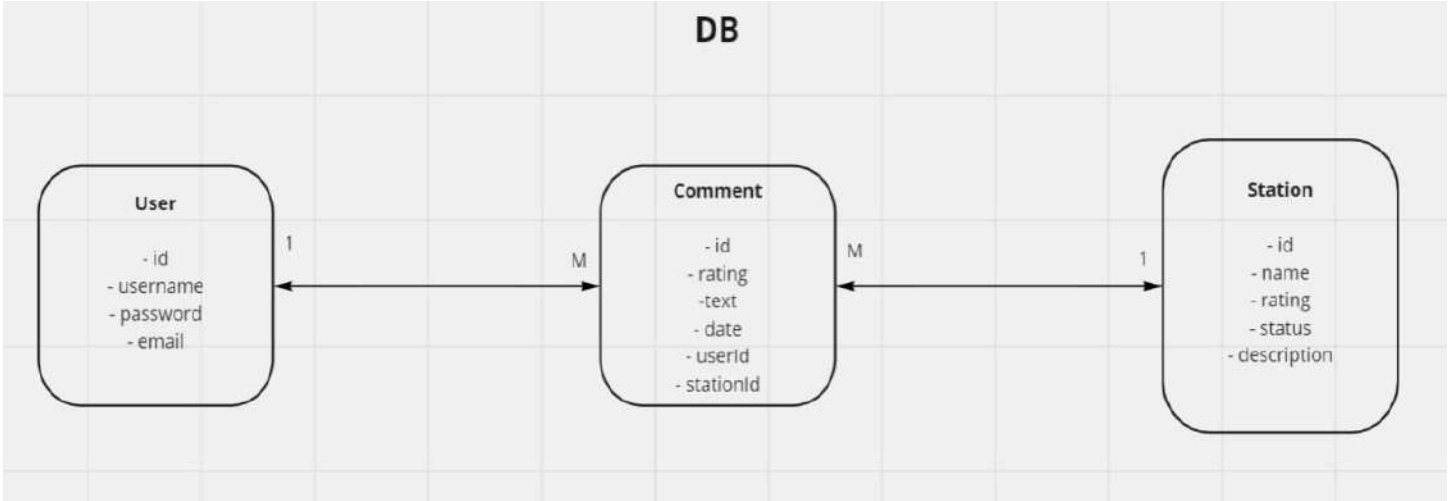
Структурна схема реалізації проекту



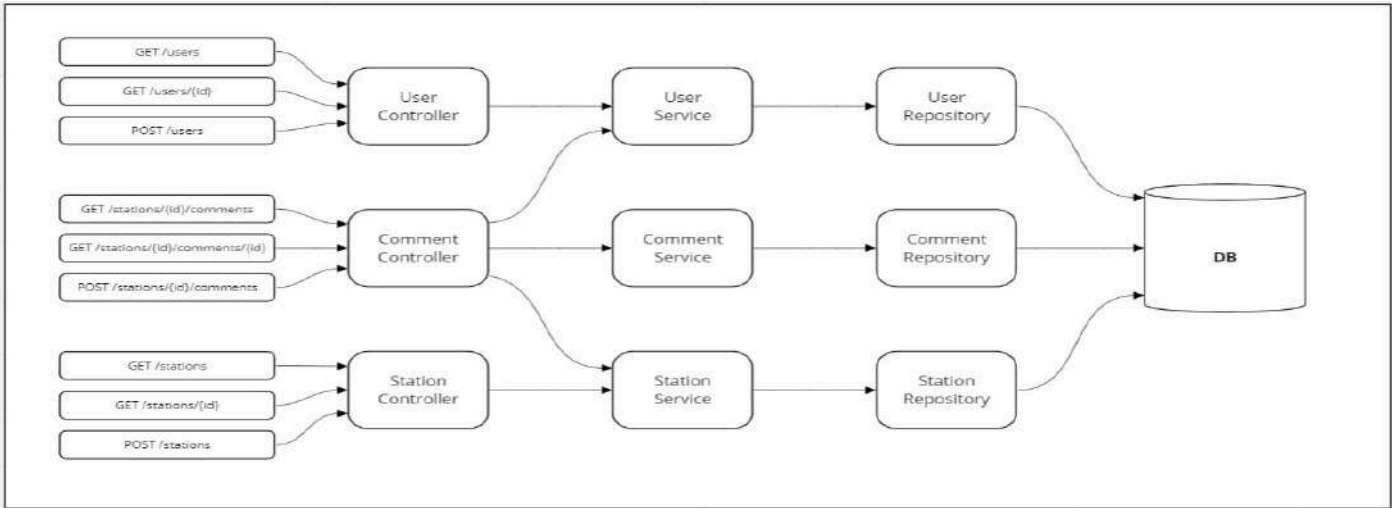
Архітектура мобільного застосунку

					13.02070849.00033 ПЛ1						
					Розробка мобільного застосунку для пошуку заправок електрокарів Структурна схема реалізації проекту	Лист.			Маса	Масшт.	
Зам.	Лист.	№ докум.	Підпис	Дата		Лист 1			Листів 1		
Розраб.		Шкода Д.С.									
Перевір.		Голуб Т.В.									
Т.контр.											
Н.контр.		Щербак Н.В.				НУ “Запорізька політехніка”					
Затвердж.		Кудерметов Р.К.				КНТ – 511сн					

Структура бази даних та серверу



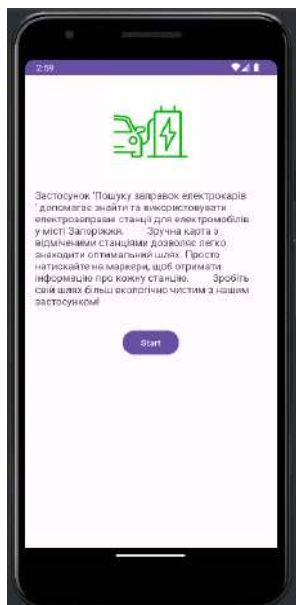
Структура бази даних



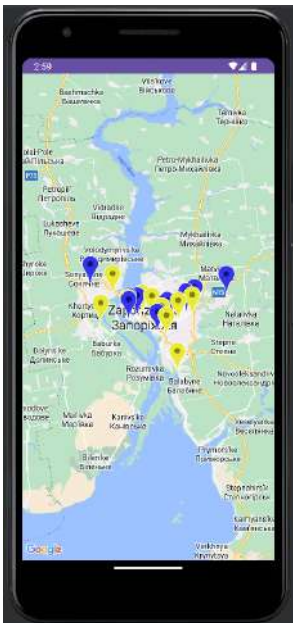
Структура елементів на backend

					13.02070849.00033 ПЛ2					
					Розробка мобільного застосунку для пошуку заправок електрокарів	Лист.		Маса	Масшт.	
Зам.	Лист.	№ докум.	Підпис	Дата	Структурна схема реалізації проекту	Лист 1		Листів 1		
Розроб.		Шкода Д.С.								
Перевір.		Голуб Т.В.								
Т.контр.										
Н.контр.		Щербак Н.В.				НУ “Запорізька політехніка”				
Затвердж.		Кудерметов Р.К.								

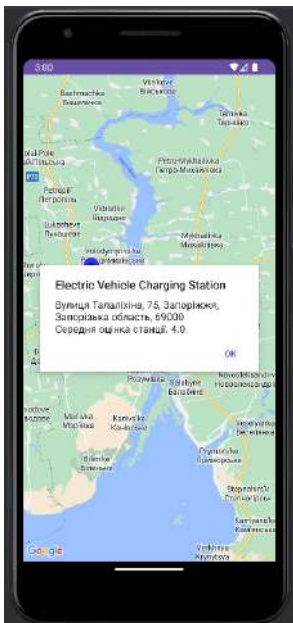
Інтерфейс мобільного застосунку



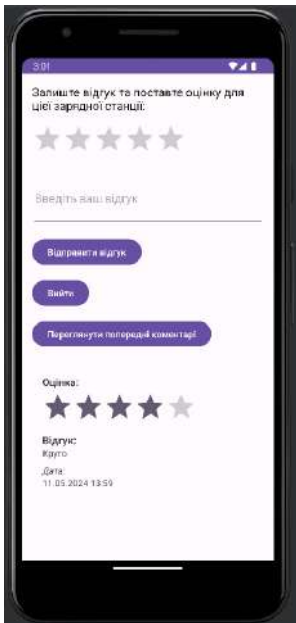
Вхід користувача до мобільного застосунку



Вхід користувача до мапи розташування заправок



Вибір заправки користувачем



Екран для відгуків та оцінок

					13.02070849.00033 ПЛЗ						
					Розробка мобільного застосунку для пошуку заправок електрокарів Структурна схема реалізації проекту	Лист.			Маса	Масшт.	
Зам.	Лист.	№ докум.	Підпис	Дата							
Розраб.		Шкода Д.С.									
Перевір.		Голуб Т.В.									
Т.контр.						Лист 1			Листів 1		
						НУ “Запорізька політехніка”					
Н.контр.		Щербак Н.В.				КНТ – 511сн					
Затвердж.		Кудерметов Р.К.									



					13.02070849.00033 АІ					
					Розробка мобільного застосунку для пошуку заправок електрокарів			Лист.	Маса	Масшт.
Зам.	Лист.	№ докум.	Підпис	Дата						
	Розраб.	Шкода Д.С.								
	Перевір.	Голуб Т.В.								
	Т.контр.									
	Н.контр.	Щербак Н.В.								
	Затвердж.	Кудерметов Р.К.								
								Лист 1		Листів 1
								НУ “Запорізька політехніка” КНТ – 511сн		