НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці

ДО ЗАХИСТУ ДОПУЩЕНО
В.о. завідувача кафедри
_____ Олександр КОВАЛЬ
«___»_____2023 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем і веб-технологій»
спеціальності 121 Інженерія програмного забезпечення
на тему: «Інструментальні засоби краудсорсингу в транспортних системах»

Виконала:
студентка IV курсу, групи ТІ-92
Тютюнник Олександра Григорівна _____
(прізвище, ім'я, по батькові) (підпис)

Керівник:
Доцент, к.е.н., Гусєва Ірина Ігорівна _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент:
Доцент, к.т.н., Сірий Олександр Анатолійович _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____
(підпис)

Київ – 2023

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма «Інженерія програмного забезпечення інтелектуальних кібер - фізичних систем і веб-технологій»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Олександр КОВАЛЬ
«___»_____2023 р.

ЗАВДАННЯ
на дипломну роботу студенту

_____Тютюнник Олександрі Григорівні_____
(прізвище, ім'я, по батькові)

1. Тема роботи _Інструментальні засоби краудсорсингу в транспортних системах
керівник роботи__Гусєва Ірина Ігорівна, к.е.н._____
(прізвище, ім'я, по батькові науковий ступінь, вчене звання)

затверджені наказом по університету від "29" ___травня__2023 року №_2039-с___
2. Строк подання студентом роботи _ _12.06.2023 р._____
3. Вихідні дані до роботи _мови Kotlin з використанням фреймворку Spring та Kotlin із бібліотекою Jetpack Compose. Продукт розроблено у середовищах IntellijIDEA та Android Studio, база даних PostgreSQL_____
4. Зміст (дипломної роботи) пояснювальної записки (перелік питань, які потрібно розробити) сформувати задачі, що повинна вирішувати система; проаналізувати аналоги; розробити мобільний застосунок для швидкого пошуку вільного місця на паркуваннях поблизу, використовуючи засоби краудсорсингу, а також дані з GPS та акселерометра.
5. Перелік ілюстративного матеріалу
проблематика, постановка задачі, архітектура системи, діаграма компонентів, діаграма послідовності, діаграма прецедентів, модель бази даних, схема програмної системи, інтерфейс додатку, висновки.__

6. Консультанти розділів роботи _____

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання ”30 ” _вересня_2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Отримання завдання	30.09.2022	Виконано
2.	Дослідження предметної області та аналіз задачі, дослідження існуючих рішень	02.10.2022-16.04.2023	Виконано
3.	Розробка програмного продукту, розробка окремих модулів системи, тестування ПП	17.04.2023-21.05.2023	Виконано
4.	Захист програмного продукту	15.05.2023-19.05.2023	Виконано
7.	Оформлення дипломної роботи	22.05.2023-04.06.2023	Виконано
8.	Передзахист	05.06.2023-11.06.2023	Виконано
9.	Захист	19.06.2023-23.06.2023	Виконано

Студентка _____ _____Тютюнник О.Г._____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____Гусєва І.І._____
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Структура та обсяг дипломної роботи. Робота містить 59 сторінок, 29 рисунків, 2 додатки та 17 посилань.

Метою роботи є розробка програмного забезпечення для використання краудсорсингу з метою зменшення часу на пошук місця паркування.

Для досягнення поставленої мети виконано такі завдання:

- проаналізовано наявні підходи використання краудсорсингу в транспортних системах;
- досліджено потреби транспортних систем у використанні краудсорсингу та визначено пріоритетні напрямки для впровадження цих інструментів.

Розроблено алгоритми, за допомогою яких програма здатна:

- оновлювати інформацію про зайнятість паркомісць у реальному часі за рахунок отриманих даних з GPS та акселерометра;
- додавати нові місця для паркування на мапу після певної кількості запитів від користувачів на основі вхідних даних.

Розроблено програмне забезпечення, що здатне використовувати розроблені алгоритми і яке надає користувачеві актуальну інформацію про ситуацію на паркуваннях поблизу.

Практичне значення одержаних результатів полягає в отриманні методів для зменшення часу пошуку вільного місця для паркування водіями. Реалізовано зручний та мінімалістичний інтерфейс. Зроблено огляд існуючих рішень, які розв'язують схожу задачу. Визначено подальші вектори розвитку та розширення можливостей програми.

Апробація результатів дипломної роботи

Основні положення роботи доповідалися та обговорювалися на конференції: Тютюнник О.Г., Гусєва І.І. Інструментальні засоби краудсорсингу в транспортних системах. *Сучасні проблеми наукового забезпечення енергетики: Матеріали XX Міжнародної науково-практичної конференції молодих вчених та студентів, м.*

Київ, 25-28 квітня 2023 року. Київ: КПІ ім. Ігоря Сікорського, Вид-во “Політехніка”, 2023. URL: https://iate.kpi.ua/uploads/p_21_72711255.pdf

Ключові слова: краудсорсинг, паркування, пошук вільного паркомісця, GPS, акселерометр, мобільний застосунок.

ABSTRACT

Structure and scope of the diploma thesis. The thesis consists of 59 pages, 29 figures, 2 appendices, and 17 references.

The aim of the project is to develop software for utilizing crowdsourcing to reduce the time spent on searching for parking spaces.

To achieve this goal, the following tasks have been performed:

- Analyzed existing approaches to using crowdsourcing in transportation systems;
- Investigated the needs of transportation systems in utilizing crowdsourcing and identified priority areas for implementing these tools.

Methods have been developed to:

- Update real-time information about parking space occupancy using data from GPS and accelerometer;
- Add new parking spaces to the map based on a certain number of user requests and input data.

A software has been developed that is capable of utilizing the developed algorithms and provides users with up-to-date information about parking situations nearby.

The practical significance of the obtained results lies in obtaining methods to reduce the time spent by drivers in searching for available parking spaces. A convenient and minimalist interface has been implemented. An overview of existing solutions that address similar problems has been conducted. Further directions for the development and expansion of the program's capabilities have been identified.

Approval of the Results of the Diploma Work

The main findings of the work were presented and discussed at the conference: Tiutiunnyk O.H., Husyeva I.I. Crowdsourcing Tools in Transport Systems. *Current Issues of Scientific Support for Energy: Proceedings of the XX International Scientific and Practical Conference of Young Scientists and Students, Kyiv, April 25-28, 2023.* Kyiv: Igor Sikorsky Kyiv Polytechnic Institute, “Politekhnika” Publishing House, 2023. URL: https://iate.kpi.ua/uploads/p_21_72711255.pdf

Keywords: crowdsourcing, parking, search for available parking space, GPS, accelerometer, mobile application.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

Краудсорсинг (англ. Crowdsourcing) – процес використання зовнішніх ресурсів, зокрема людей, для вирішення різних завдань. Цей підхід передбачає розподіл завдань або проектів серед великої кількості людей, які надають свій внесок у формі тестування, написання відгуків або надання своєї власної інформації.

Акселерометр - це електронний пристрій, який вимірює прискорення, що відбувається в даному об'єкті. Він виявляє зміни в руху, включаючи прискорення вздовж трьох осей. Акселерометри використовуються в різних пристроях, включаючи смартфони, планшети, спортивні трекери та інші пристрої, для визначення орієнтації, виявлення кроків, вимірювання сили гравітації та виконання різних функцій, пов'язаних з детектуванням руху.

GPS (англ. Global Positioning System) - це глобальна система позиціонування, що базується на супутниковій навігації. Вона складається з мережі супутників, які надсилають сигнали до приймачів на землі, що можуть бути вбудованими в пристрої такі як смартфони, навігаційні пристрої або автомобільні системи. Вони, у свою чергу, отримують ці сигнали та обробляють їх для визначення точного місцезнаходження об’єкту на поверхні Землі.

HTTP (англ. HyperText Transfer Protocol) - це протокол передачі гіпертексту, який використовується для здійснення комунікації між клієнтом та сервером в Інтернеті.

REST (англ. Representational State Transfer) - це архітектурний стиль для розробки веб-сервісів, який базується на стандартах протоколу HTTP. REST використовується для побудови веб-сервісів.

ВСТУП

Із зростанням індустріалізації, зростає також кількість машин у містах, через що збільшується забруднення і зменшується кількість вільних місць для паркування автомобіля. З’являється питання оптимізації часу, проведеного у пошуках місця для зупинки. Однак через велику кількість різних типів місць для паркування і їх розташування, з’являється проблема у обробці усіх цих даних і їх збереження в одному місці, а також підтримки їх актуальності в режимі реального часу. Через такі причини і з’являються інструменти для полегшення та пришвидшення пошуку вільного місця для транспортного засобу.

Актуальність таких застосунків полягає у тому, що вони здатні вирішити або, хоча б частково, покращити проблему з паркуваннями, оптимізувавши час їх пошуку і, як наслідок, зменшити викиди шкідливих речовин від транспорту в повітря або заощадити енергію електричного автомобіля для наступних поїздок.

Такі застосунки можуть допомогти державі проаналізувати, у яких районах потрібно побудувати додаткові місця для паркування, а де, навпаки, їх зменшити. Це допоможе зробити місто більш зручним для водіїв та корисним для мешканців.

Сучасні технології дозволяють робити такі інструментальні засоби, використовуючи різні методи: сканування камер з паркувальних майданчиків [1], встановлення сенсорів [2] на кожне паркомісце безпосередньо, сканування місцевості лазерами, використання краудсорсингу та інші.

В рамках цієї роботи буде визначено основні завдання та актуальність використання останнього методу, проаналізовано існуючі рішення, їхні переваги та недоліки, а також розроблено мобільний застосунок. Для досягнення цих цілей буде використано аналіз літературних джерел та тестування існуючих програм.

1 ПОСТАНОВКА ЗАДАЧІ

1.1 Мета та основні завдання

Мета роботи полягає в розробці програмного забезпечення для використання краудсорсингу з метою зменшення часу на пошук місця паркування.

Задачі, які необхідно розв’язати для досягнення мети.

1. Проаналізувати наявні підходи до використання краудсорсингу в транспортних системах, включаючи огляд наукових робіт та проектів в даній області. Пошук та тестування аналогів.

2. Розробити алгоритм, який працюватиме на основі засобів краудсорсингу. Вхідними даними буде інформація користувачів про ситуацію на паркуваннях, а також значення з GPS та акселерометра. Це дозволить у реальному часі оновлювати інформацію про зайнятість паркомісць, а також додавати нові місця для паркування на мапу після певної кількості запитів від користувачів.

3. Розробити мобільний застосунок, що буде містити карту з місцями для паркування, детальною інформацією про кожне з них, а також базуючись на даних з GPS та акселератора, буде отримувати інформацію про те, чи зайняте паркомісце та зберігати відповідні дані.

4. Розробити додаткові функції, такі як список улюблених місць для паркування, можливість надсилати запит на додавання нових місць для паркування або видалення тих, що вже неактуальні.

5. Протестувати основні функції розробленої системи.

Такий застосунок може бути особливо корисним в містах, де є складнощі з паркуванням, або у туристичних зонах, де велика кількість автомобілів збирається на одній парковці. Крім того, система може бути розширена на інші види транспорту, наприклад, на велопарковки.

1.2 Основні задачі системи

Передбачається розробка мобільного застосунку для водіїв, що допоможе зменшити час пошуку вільного місця для паркування. Для цього будуть використовуватись дані з GPS та акселерометра, а також дані від користувачів, що дозволить визначати кількість вільних місць на паркуваннях у реальному часі.

Однією з ключових функцій додатку буде використання краудсорсингу - відкритої системи збору та обробки інформації, до якої можуть долучатися користувачі додатку. Люди зможуть надавати актуальні дані про вільні місця на парковках, що дозволить системі оновлювати дані в реальному часі. Крім того, користувачі зможуть робити запити на додавання або видалення зон паркувань у додатку, що дозволить системі бути актуальною та корисною для всіх користувачів.

1.3 Задачі та вимоги для клієнтської частини

Клієнтська частина повинна відповідати найкращим практикам та рекомендаціям, що описані в офіційній документації [3].

Основні вимоги:

1) використовувати загальнодоступні іконки та компоненти інтерфейсу користувача, що покращить розуміння коду та його кількість;

2) програма має містити кнопки “Назад”, бокове та нижнє меню для зручної навігації по сторінках;

3) система повинна бути адаптованою під більшість версій операційних систем та різні розміри екрану для охоплення більшої кількості користувачів;

4) необхідно дотримуватись Android правил при використанні іконок або певних шаблонів поведінки і не змінювати їх функції аби не заплутати користувача.

Користувач застосунку повинен мати змогу переглядати існуючі паркування та їхню детальну інформацію (загальна кількість місць, кількість зайнятих місць,

вартість паркування за годину, чи наявні паркомісця для інвалідів або електричних автомобілів). Тому доцільно зробити перехід від обраної мітки на мапі головного екрану до більш детального опису зони паркування.

Користувач повинен мати можливість зберегти паркування у список своїх улюблених, а також переглянути або відредагувати його на окремому екрані системи.

Додатковою рекомендацією є створення гнучкого налаштування інтерфейсу на прикладі зміни мови інтерфейсу або ж його теми зі світлої (за замовчуванням) на темну.

1.4 Задачі та вимоги для серверної частини

Серверна частина має працювати з даними зі створеної бази даних. Для цього необхідно налаштувати підключення між сервером і базою даних, що дозволить обмінюватися даними між ними.

Для забезпечення взаємодії між сервером і базою даних можна використовувати різні технології та механізми, такі як JDBC (англ. Java Database Connectivity) для Java або Kotlin проектів або ORM (англ. Object-Relational Mapping) фреймворки, які спрощують роботу з базами даних шляхом перетворення об'єктів у записи бази даних і навпаки.

Після встановлення з'єднання з базою даних необхідно налаштувати REST операції для взаємодії з даними, такі як [4]:

- GET – отримання даних:
- POST – додавання нових даних;
- PUT/PATCH – модифікація існуючих даних;
- DELETE – видалення даних.

Дана частина запланованої системи повинна не тільки коректно та швидко відпрацьовувати запити та передавати необхідні дані на клієнтську частину, але також забезпечувати надійність та цілісність операцій з даними. Для цього може

бути застосовано принципи ACID, що надасть серверній частині системи гарантію надійності, цілісності та стійкості будь-яких операцій з даними.

1.5 Загальні вимоги

Для розробки такого застосунку необхідно мати розуміння технічних аспектів використання GPS та акселерометра для збору даних, а також розуміння того, яким чином можна використовувати дані з цих датчиків для побудови мапи вільних місць на паркуваннях.

Потрібно виконати пошук інформації з побудови коректного запиту користувачеві про використання цих даних, попередньо виводячи на екран запит про надання власником мобільного телефону необхідних метрик.

Крім того, важливо розробити механізм краудсорсингу, який дозволить користувачам надавати актуальні дані в систему та забезпечити збір та обробку цих даних і щоб вони були коректними. Важливо також розробити інтерфейс застосунку, що буде зручним та зрозумілим для користувачів, дозволяючи їм легко знаходити необхідну інформацію про вільні місця на паркуваннях. Для цього можуть використовуватись картографічні інструменти та візуалізації даних, що допоможуть користувачам швидко зорієнтуватись у місті та знайти вільне місце для паркування.

1.6 Вхідні та вихідні дані програми

Вхідними даними системи є дані з GPS та акселерометра, база даних із вже існуючими паркуваннями, а також інформація про поточну ситуацію із зайнятістю паркомісць від користувачів.

Вихідними даними є зміна кількості зайнятих місць на паркуванні, додавання або видалення зони паркомісця на мапі в інтерфейсі програми.

Висновки до розділу 1

У розділі сформульовано мету та завдання, які необхідно виконати для досягнення поставленої мети.

Запланована система має на меті вирішення завдань моніторингу наявних місць для паркування та надання актуальної інформації про вільні місця на мапі застосунку.

Для досягнення цих цілей, система повинна використовувати найпопулярніші практики при побудові мобільного застосунку та створення зрозумілого для користувачів інтерфейсу. Також для забезпечення стандартизованої та простої взаємодії з сервером необхідно використовувати REST та ACID принципи.

Краудсорсинг використовується у системі для залучення зовнішніх ресурсів, таких як експертні знання та внесок водіїв, щоб швидко отримувати актуальну інформацію про ситуацію на дорозі. Зокрема, система залучає дані з акселерометра та GPS для моніторингу та оновлення зайнятості паркомісць у реальному часі.

2 АНАЛІЗ ІСНУЮЧИХ СИСТЕМ

2.1 Опис аналогів

Під час вибору методу реалізації для майбутньої програми, було розглянуто декілька застосунків, які використовують, наприклад, дані з камер або з сенсорів як джерело даних. Останній метод більше підходить для маленьких приватних паркувань, які обмежені в кількості місць, бо встановлення сенсору на кожне паркомісце є досить витратно.

Використання краудсорсингу є популярним для отримання великої кількості інформації за короткий час. Популярна програма WAZE майже повністю побудований на цьому, тому там завжди актуальна інформація про ситуацію на дорогах.

У багатьох містах Європи встановлені табло з інформацією про кількість вільних місць на найближчих паркуваннях вздовж дороги. Це спрощує побудування маршруту водієві та зменшує використання бензину або електроенергії, якщо автомобіль електричний. Приклад такого табло наведено на рисунку 2.1.

Рисунок 2.1 – Приклад показу кількості вільних місць на дорогах європейських міст

Щодо мобільних застосунків, існують закордонні англомовні аналоги таких систем із використанням краудсорсингу для Лос-Анджелеса та Сан-Франциско [5, 6]. Завдяки таким програмам є можливість переглянути мапу паркувань міста, відфільтрувати їх за певними характеристиками (ціна, район, тип транспорту), забронювати або ж навіть створити запит на додавання нового паркомісця на мапі.

Зовнішній вигляд програм наведено нижче на рисунках 2.2 та 2.3.

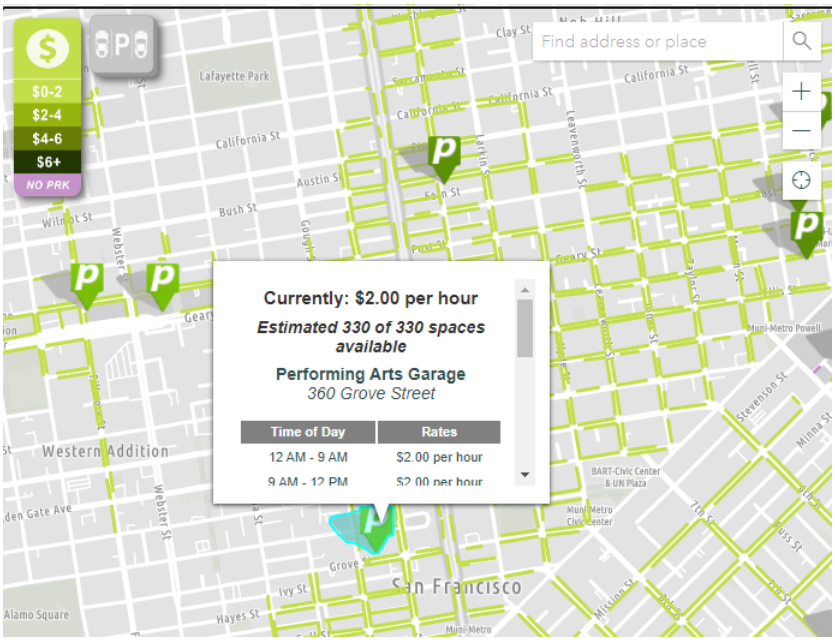

Рисунок 2.2 – Веб-застосунок з інформацією про паркування для Сан-Франциско

Переваги:

1) детальна інформація про паркування;
2) велика база даних паркомісць по всьому Сан-Франциско;
3) є певні фільтри та поле для пошуку бажаної вулиці, района.

Недоліки:

1) немає мобільної версії, тільки веб-застосунок;
2) немає прив’язки до певного користувача;
3) відсутність можливості створювати запит на додавання нових паркувань.

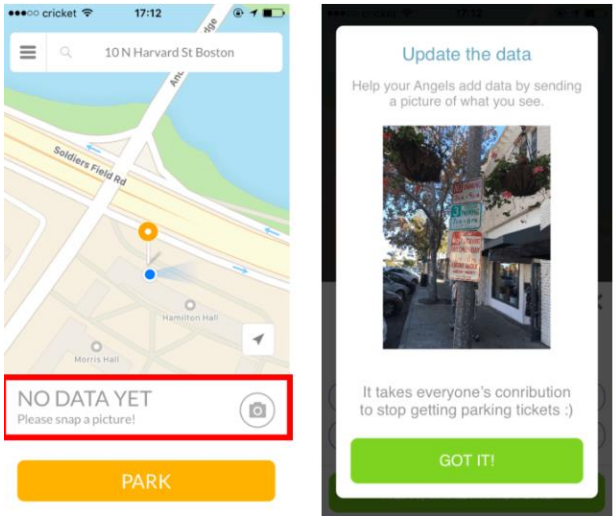

Рисунок 2.3 – Мобільний застосунок з інформацією про паркування для Лос-Анджелеса

Переваги:

1) це мобільний застосунок, що значно полегшує роботу з ним;
2) має простий та мінімалістичний дизайн.

Недоліки:

1) немає великої бази із вже існуючими паркуваннями;
2) основна функція – мануальне додавання користувачем, а згодом показ на мапі нового паркомісця;
3) немає ніякої взаємодії з користувачем.

Виходячи з даного аналізу було сформовано вимоги для майбутньої програми:

1) це має бути саме мобільний застосунок для більш зручного його використання водіями;
2) має бути вже існуюча база паркомісць;
3) окрім можливості мануально (англ. manually – вручну) створювати запити про додавання нових паркомісць або, навпаки, видалення недійсних місць, також мати взаємодію з користувачем у вигляді запитань про вільні місця поряд нього і тд у певний час, аналізуючи дані з акселерометра;
4) у фоновій роботі додатково перевіряти дані з GPS та акселерометра для

більш детальної інформації та можливості дізнаватись про нові паркомісця, яких нема на мапі та в базі даних, не забираючи час у водія; Звісно, такі дані беруться тільки за згодою користувача;

5) покращити та актуалізувати дизайн програми.

Наступним етапом був перегляд програм з мапами для затвердження зовнішнього вигляду майбутньої системи.

На рисунках 2.4, 2.5 наведено приклади відображення детальної інформації про паркомісце. У обох випадках це зроблено у вигляді вспливаючого поля на третину екрану, де зображена детальна інформація про паркування, що дає змогу краще і швидше зрозуміти що собою представляє дане паркування.

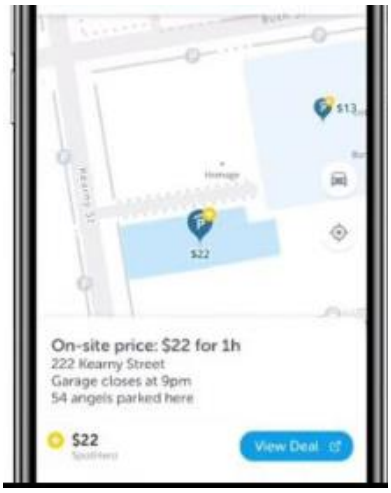

Рисунок 2.4 – Приклад показу інформації про паркування на мапі

У другому варіанті також є додаткові можливості при натисканні на кнопку “Більше” справа. На рисунку 2.6 зображено екран програми, що відкривається після натискання на іконку з трьома крапками, яка є клікабельною (англ. clickable – з можливістю натискання на елемент, у випадку програмної системи).

Користувачеві надається можливість:

- поділитися даною локацією у популярні програми для спілкування (Telegram, Viber, WhatsApp, Facebook тощо);
- зробити дану мітку як старт своєї подорожі;
- редагувати – відправити запит системі про некоректність якихось даних.

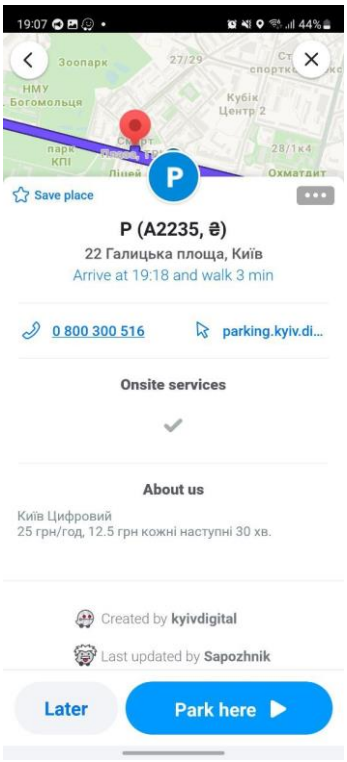

Рисунок 2.5 – Приклад виводу інформації про паркування

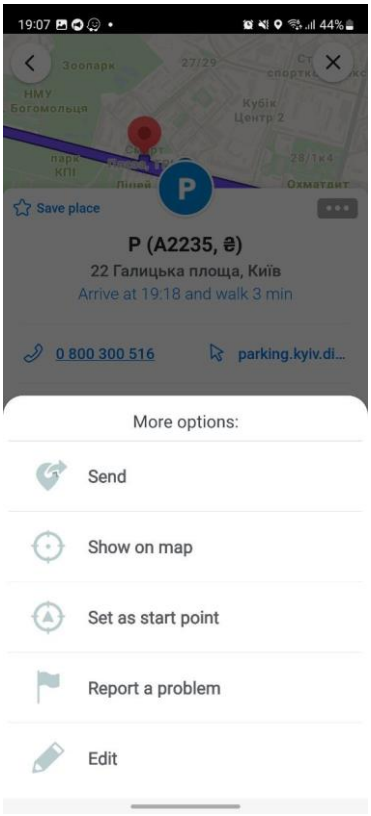

Рисунок 2.6 – Екран застосунку з додатковими можливостями

На рисунках 2.7 та 2.8 наведено проаналізовані дизайни головної сторінки додатку. Обидва варіанти мають мітки з певною інформацією. Це може бути вартість паркомісця за годину, кількість вільних місць або ж, якщо мапа сильно віддалена, то мітки відображатимуть кількість наявних зон для паркування в певному районі.

Також важливо мати кнопку “Моє місцерозташування” для переходу на свою геолокацію, додатково можна створити кнопку “Компас”, яка буде повертати мапу у правильне положення відносно сторін світу, а також кнопки “Збільшити” та “Зменшити” як можливий варіант для навігації по мапі, окрім жестів пальцями по екрану.

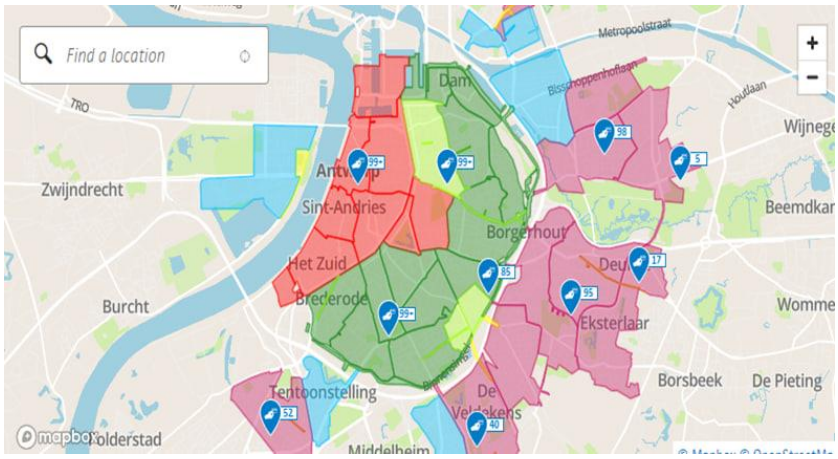

Рисунок 2.7 – Приклад полігонних позначень паркувань

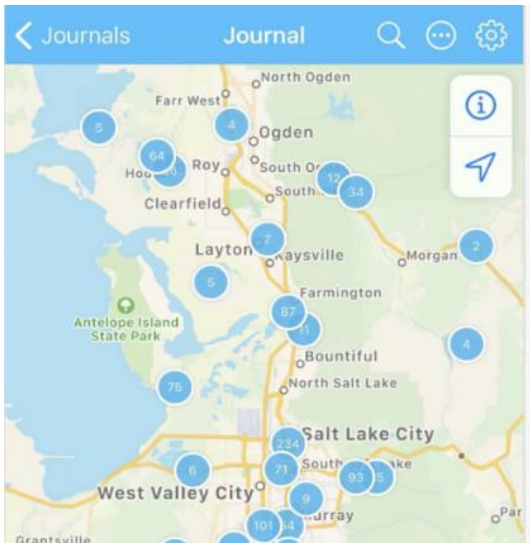

Рисунок 2.8 – Приклад позначок паркомісць у застосунку

У результаті аналізу дизайнів, було вирішено як має виглядати клієнтська частина програми:

- місця для паркування будуть зображені у вигляді полігонів (певної площі певного кольору поверх мапи) з маркером посередині, натиснувши на який можна буде отримати детальну інформацію;
- у залежності від кольору маркера, буде зрозуміло чи заповнено повністю машинами зона паркування: зелене – ще є вільні місця, червоне – усе зайнято;
- має бути кнопка “Моя локація” для швидкого переміщення до свого місцезнаходження;
- мати можливість зберегти певне паркування до свого списку улюблених;
- створити формочки для запиту на додавання нового місця паркування на мапу від користувача.

2.2 Аналіз GPS, акселерометра та засобів краудсорсингу

У наш час багато великих компаній використовують краудсорсинг для швидкого збору інформації, такі як Netflix, Uber, Amazon, Google, Starbucks та багато інших. Один з найпоширеніших видів краудсорсингу – це програмне забезпечення з відкритим кодом (англ. open-space). Розробка з відкритим кодом дозволяє розробникам отримати доступ до вихідного коду, дозволяючи їм змінювати та покращувати код, як вони вважають за потрібне.

У випадку даної роботи, краудсорсинг використовується для отримання даних від великої кількості користувачів, що надає можливість аналізувати та мати точні, деталізовані, актуальні дані про стан паркувань.

Проблемами, пов’язаними з використанням краудсорсингу, є необхідність ретельної перевірки достовірності, коректності та актуальності даних, що надходять у програму [7]. Оскільки даних багато і надсилаються від різних користувачів, необхідно встановити механізми фільтрації, видалення дублікатів та розробити алгоритми для їх порівняння.

Одна з ключових загроз полягає у можливості перетворення бази даних з реальними паркомісцями на неправдиву або непридатну через недобросовісних користувачів або хакерів. Тому важливо встановити механізми перевірки даних на достовірність перед додаванням їх в систему.

Додатково, варто розглянути введення обмежень у часі для надсилання запитів користувачами. Це дозволить контролювати частоту та обсяг даних, що приходять, запобігаючи можливості надмірного завантаження системи та забезпечуючи стабільну та швидку роботу.

GPS використовується в автомобільних навігаційних системах, мобільних пристроях, картографічних додатках та інших технологіях для визначення місцезнаходження та навігації. Такі геодані можуть використовуватись, щоб показувати користувачеві його місцезнаходження та паркування поблизу нього.

Акселерометри, у свою чергу, широко використовуються в мобільних пристроях, контролерах ігрових консолей, фітнес-трекерах та інших електронних пристроях для забезпечення більш інтерактивного та захоплюючого користувацького досвіду [8]. Вони працюють шляхом вимірювання змін у напрямку руху або прискорення в просторі. Такі дані можуть бути використані для розпізнавання різних дій та станів пристрою.

У випадку системи моніторингу паркування, що розробляється, дані з акселерометра можуть аналізуватися для визначення моменту, коли водій зупиняється. Якщо виміряне прискорення показує, що пристрій перестав рухатися певний час, більший за просто зупинку на світлофорі, то можна вважати, що автомобіль припаркувався. Це дає можливість системі змінювати кількість вільних місць у фоновому режимі, без взаємодії з користувачем.

А якщо об’єднати дані з акселерометра та GPS, то додатково буде відомо де саме зупинився водій, і тоді, у випадку якщо ця локація не занесена у систему як зона для паркування, то буде можливість створити запит на додавання цього місця в систему користувачем власноруч.

Висновки до розділу 2

У результаті вивчення існуючих аналогів для вирішення проблеми з паркуваннями, було описано вимоги до програмного продукту та його дизайну інтерфейсу. Також відбулось ознайомлення із GPS, акселерометром та засобами краудсорсингу, що допоможе мати актуальні та точні дані про стан та місцезнаходження водіїв.

Загалом, створення мобільного застосунку для системи моніторингу паркування з використанням GPS, акселерометра та краудсорсингу вимагає комплексного підходу до розробки, з урахуванням вимог до інтерфейсу, функціоналу та забезпечення достовірності та актуальності даних. Такий застосунок може забезпечувати користувачам зручну та інформативну систему для пошуку та моніторингу вільних паркомісць, поліпшуючи їх досвід паркування.

3 ЗАСОБИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Обґрунтування вибору мови програмування та бібліотек

Для серверної частини використовуватиметься Kotlin зі Spring фреймворком, а для клієнтської частини – Kotlin із Jetpack Compose.

Kotlin є новою мовою програмування для JVM, яка з'явилася в 2011 році. Вона була створена компанією JetBrains, яка є розробником популярних інтегрованих середовищ розробки, таких як Intellij IDEA [9]. Kotlin є мовою, яка є сумісною з Java, тобто код на Kotlin може співіснувати з кодом на Java, що дозволяє розробникам поступово переходити на Kotlin без потреби повного переписування коду [10].

Однією з основних переваг Kotlin є його безпека. Kotlin підтримує надійність відносно NullPointerException, завдяки тому, що він має систему типів, що може виявляти потенційні помилки перед їх виконанням.

Spring є одним з найпопулярніших фреймворків для розробки веб-додатків на Java/Kotlin. Він має велику спільноту розробників, що дозволяє швидко отримувати підтримку та рішення проблем. Spring дозволяє вирішувати проблеми, пов'язані зі складністю розробки веб-додатків, шляхом надання простих інструментів та функцій для розробників. Усі зв’язування сутностей відбуваються автоматично завдяки можливостям фреймворка, а програмістові достатньо просто додати потрібну анотацію на потрібний клас [11].

Для розробки клієнтської частини програми, яка буде використовувати Jetpack Compose, Kotlin є однією з найкращих мов програмування, особливо для платформи Android. Jetpack Compose представляє собою новий набір бібліотек, який змінює підхід до створення користувацького інтерфейсу шляхом впровадження декларативного програмування [12]. Замість використання складних вкладених макетів, як це робиться в XML, розробники можуть застосовувати

анотацію “@Composable” до функцій, що описують окремі компоненти інтерфейсу користувача.

Такий підхід дозволяє описувати зовнішній вигляд програми, використовуючи звичайні Kotlin функції, що полегшує розробку і покращує зрозумілість коду. Компоненти, створені за допомогою Jetpack Compose, можна повторно використовувати в інших місцях коду, а також їх можна комбінувати для створення складних інтерфейсів.

Крім того, Jetpack Compose пропонує багато вже існуючих вбудованих компонентів і функцій, які спрощують розробку різноманітних елементів, включаючи кнопки, тексти, списки, анімацію та багато іншого. Це зменшує кількість витраченого часу на написання великої кількості коду.

3.2 Обґрунтування використання Google Maps Api

Для можливості працювати з мапами у мобільному застосунку, необхідно використовувати Google Maps Sdk (англ. Software Development Kit) – набір інструментів, бібліотек, документації та прикладних програм, які надаються розробникам для побудови програмного забезпечення та Google Maps API (англ. Application Programming Interface) – набір визначених правил та протоколів, які дозволяють програмним компонентам взаємодіяти один з одним [13]. API визначає, які функції та процедури доступні для використання, як обмінюватись даними та яких правил слід дотримуватись під час взаємодії. Вони дозволяють інтегрувати картографічні можливості Google Maps у власні мобільні або веб-додатки, розширюючи їх функціональність та надаючи користувачам багатоцільовий інтерфейс [14].

Основні можливості Google Maps SDK та API включають:

1) геокодування та зворотнє геокодування – API надає можливість перетворювати адреси в географічні координати (геокодування) та навпаки (зворотнє геокодування). Це дозволяє знаходити місцезнаходження на карті за

адресою або отримувати адресу за координатами;

2) карти, відображення та маркери – Google Maps SDK надає можливість відображати карти різного розміру та рівня деталізації, включаючи 2D та 3D відображення. Розробники можуть змінювати масштаб, поворот та нахил карти, а також встановлювати маркери, лінії, полігони та інші елементи на карті. Це дозволяє створювати інтерактивні картографічні додатки з можливістю навігації та маркування спеціальних місць;

3) відстеження місцезнаходження – за допомогою Google Maps SDK можна отримувати доступ до інформації про поточне місцезнаходження користувача. Це може бути використано для створення функцій служби місцезнаходження або розрахунку відстані від користувача до певного місця;

4) візуалізація даних – SDK надає розробникам зручні інструменти для відображення географічних даних у вигляді шарів, маршрутів, полігонів та інших елементів. Це дозволяє створювати кастомні (англ. custom) – індивідуальні картографічні візуалізації та інтерактивні елементи на мапі.

Перш за все, для початку роботи з Google Maps, необхідно отримати API ключ безпосередньо з платформи Google Maps. Цей ключ потрібний для підключення карти до системи, що розробляється. Також необхідно мати конфігурацію для платформи Android, де вказується мінімальна версія Android SDK у файлі build.gradle.

Отриманий API ключ треба додати у файл конфігурації AndroidManifest.xml, після чого з’являється можливість взаємодіяти з Google Maps API і використовувати всі його функції. Цей процес забезпечує належне підключення та інтеграцію картографічної функціональності до мобільного застосунку.

У Google Maps SDK для Android, який використовується у Kotlin Android проекті з Jetpack Compose, є кілька популярних методів, які надають розширені можливості для взаємодії з картами.

1. MapView – метод дозволяє відображати карту у вікні або контейнері екрану мобільного телефону.

2. MapFragment – метод дозволяє вбудовувати карту у фрагмент.

3. GoogleMap – метод надає доступ до основного об'єкту GoogleMap, який представляє собою карту. Можна використовувати його для виконання різноманітних операцій з картою, таких як додавання маркерів, відображення шарів, зміна розмірів карти тощо.

4. Marker – метод дозволяє створювати та налаштовувати маркери на карті. Можна встановлювати позицію маркера, додавати текстову інформацію, зображення та власні елементи візуалізації.

5. Polyline – метод дозволяє створювати та керувати полілініями на карті. Це надає можливість відображати лінії шляхів, маршрутів або будь-яких інших лінійних об'єктів.

6. Polygon – метод дозволяє створювати та керувати полігонами на карті. Використовується для відображення закритих географічних областей, таких як парків, вулиць, паркувань або будь-яких інших багатокутників.

7. CameraPosition – цей клас представляє позицію камери на карті. Його можна використовувати для визначення положення та орієнтації камери. CameraPosition дозволяє змінювати позицію та налаштовувати параметри камери для відображення областей карти з різних перспектив.

8. LatLng – цей клас представляє географічну широту та довготу на карті. Він використовується для встановлення точок на карті, таких як позиція маркерів, поліліній та полігонів. LatLng дозволяє точно визначити географічні координати і використовувати їх для розміщення об'єктів на карті.

Ці методи, разом з іншими доступними функціями Google Maps SDK, надають потужні інструменти для створення і взаємодії з картографічними компонентами у Kotlin Android проекті з використанням бібліотек Jetpack Compose.

3.3 Обґрунтування використання даних GPS

У Kotlin Android проекті з використанням Jetpack Compose є можливість взаємодіяти з даними з GPS, щоб отримувати інформацію про місцезнаходження пристрою. Для цього необхідно додати дозволи у файл AndroidManifest.xml, а також у інтерфейсі спитати про це користувача.

Головним класом цієї частини системи є клас-слухач, який реалізує інтерфейс LocationListener і відповідає за прослуховування подій GPS і отримання даних про місцезнаходження власника телефону.

Метод onLocationChanged буде викликатися у випадку, коли координати будуть змінюватися. Далі ці координати можна використовувати на інтерфейсі, оновлюючи його з новим місцезнаходженням або виконувати різні обчислення у шарі бізнес-логіки.

Також використовується клас FusedLocationProviderClient для спрощеного доступу до різних джерел місцезнаходження, таких як GPS, мережа мобільного оператора або Wi-Fi, забезпечуючи точне місцезнаходження.

Основні функціональні можливості FusedLocationProviderClient включають:

1) отримання поточного місцезнаходження користувача з використанням методу getLastLocation. Цей метод повертає об'єкт типу Location, який містить координати широти та довготи, а також інші додаткові дані про місцезнаходження;

2) отримання оновлень місцезнаходження в режимі реального часу за допомогою методу requestLocationUpdates. Це дозволяє отримувати оновлення про місцезнаходження в режимі реального часу і використовувати їх для відображення на мапі або виконання інших дій;

3) управління налаштуваннями місцезнаходження за допомогою методів LocationRequest.setPriority та LocationRequest.setInterval, можна встановити вимоги щодо точності місцезнаходження та інтервалів оновлення;

4) обробка помилок та статусів місцезнаходження. За допомогою методу LocationAvailability можна перевірити наявність помилок з'єднання, недоступності місцезнаходження або зміни стану послуги місцезнаходження.

3.4 Обґрунтування використання даних акселерометра

Акселерометр є одним з датчиків, які доступні на багатьох сучасних смартфонах і планшетах. Для роботи з даними з акселерометра у мобільному застосунку Kotlin з використанням Jetpack Compose необхідно також додати залежність на нього у AndroidManifest.xml файлі, створити клас-слухач, що реалізує інтерфейс SensorEventListener. Метод onSensorChanged буде викликатись, коли пристрій починає рухатись. Нові значення можна використовувати наприклад для оновлення інтерфейсу додатку.

3.5 Обґрунтування вибору системи управління базою даних

SQL (англ. Structured Query Language) - це мова запитів до систем управління базами даних (СУБД), яка використовується для створення, модифікації та управління реляційними базами даних. SQL дозволяє виконувати різноманітні запити до бази даних, включаючи створення таблиць, вставку даних, оновлення, видалення, агрегування даних та злиття таблиць.

Системою для управління бази даних було обрано PostgreSQL, адже це потужна система з широким спектром функцій. Вона відома своєю надійністю та стійкістю, може обробляти великі обсяги даних та забезпечує стійку роботу в умовах високого навантаження, що є ідеальним для даної програми та щоб задовольнити усі вимоги ACID (англ. Atomicity, Consistency, Isolation, Durability) – набір властивостей, які гарантують надійність транзакцій і збереження цілісності даних в базі даних [15, 16].

1. Атомарність (Atomicity) – усі операції в транзакції розглядаються як одна

недільна одиниця. Це означає, що всі операції транзакції виконуються повністю або жодна з них не виконується.

2. Співмірність (Consistency) – транзакція забезпечує перехід бази даних з одного консистентного стану в інший. Це означає, що після виконання транзакції дані повинні залишатися у валідному і коректному стані.

3. Ізоляція (Isolation) – кожна транзакція виконується ізольовано від інших транзакцій. Це означає, що внутрішні зміни, внесені в рамках однієї транзакції, не впливають на інші транзакції, що виконуються паралельно.

4. Стійкість (Durability) – після успішного виконання транзакції та фіксування змін у постійному сховищі, дані повинні залишатися стійкими та доступними, навіть у випадку відмови апаратного чи програмного забезпечення.

PostgreSQL має вбудовану підтримку різних типів даних, включаючи геодані, графи та JSON. Вона також має велику кількість розширень та додатків, що дозволяє додавати нові функції та можливості в базу даних.

Таким чином, налаштування підключення між серверною частиною і базою даних дозволяють серверу працювати з даними програми, забезпечуючи їх доступність, цілісність та можливість виконання різних операцій над ними.

3.6 Обґрунтування вибору архітектури системи

Для серверної частини мобільного застосунку було обрано архітектуру MVC (Model-View-Controller), що є однією з найпоширеніших архітектурних підходів для розробки серверних додатків.

У MVC архітектурі класи поділяються на три основні компоненти:

1) модель (Model) – представляє бізнес-логіку програми та дані, з якими працює додаток. Це можуть бути класи, що взаємодіють з базою даних (repository) відображають сутності додатку (data classes) або зовнішніми сервісами;

2) представлення (View) – відповідає за відображення даних користувачеві і взаємодію з ним. Це може бути реалізовано у вигляді HTML-шаблонів, сторінок

JSP (JavaServer Pages) або за допомогою сучасних шаблонів Thymeleaf або ReactJS;

контролер (Controller) – є посередником між моделлю та представленням. Ця частина обробляє запити від користувача, взаємодіє з моделлю для отримання або оновлення даних, і передає ці дані відповідному представленню. У Spring фреймворці контролери можуть бути реалізовані за допомогою анотацій “@Controller” або “@RestController”, які вказують, що клас є контролером і обробляє HTTP-запити. Для перевірки коректності надісланих даних використовується програма Postman – це програмне забезпечення для тестування та розробки API. Postman дозволяє розробникам легко створювати, тестувати та документувати API виклики своєї системи.

На рисунку 3.1 зображено архітектуру серверної частини.

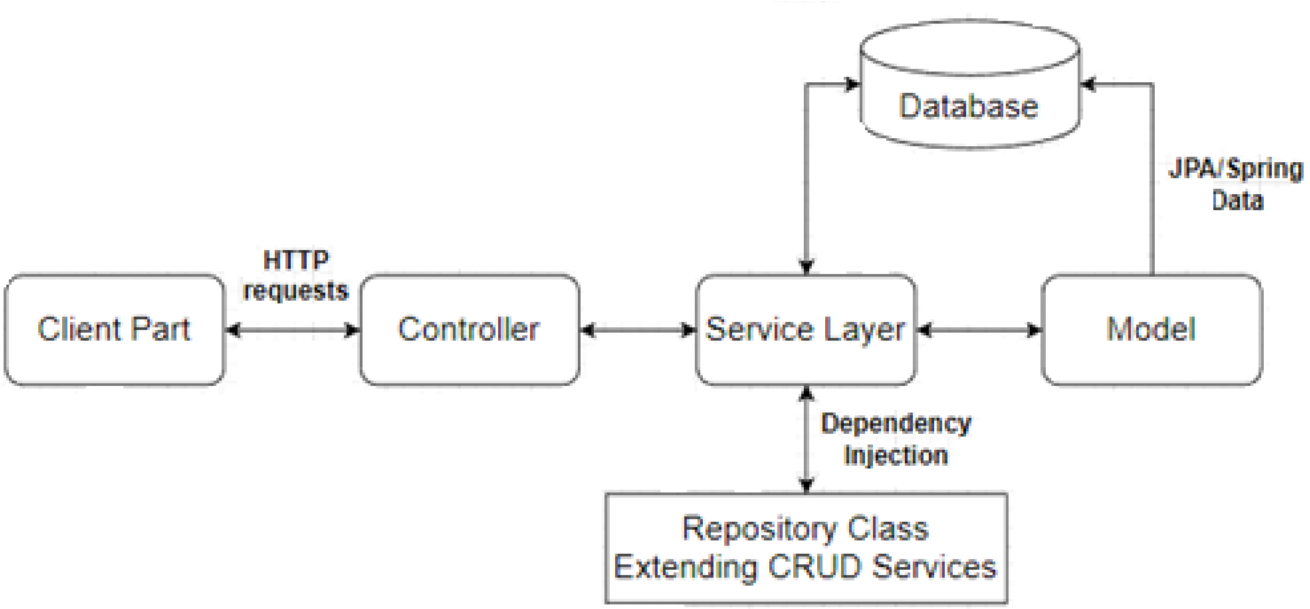

Рисунок 3.1 – Діаграма архітектури серверної частини системи

Архітектурою для клієнтської частини було обрано Clean Architecture для мобільних застосунків – це підхід, який фокусується на розділенні обов’язків. Його використання дозволяє зробити код більш організованим, розширюваним та простим для тестування.

На рисунку 3.2 зображено діаграму компонентів усієї системи.

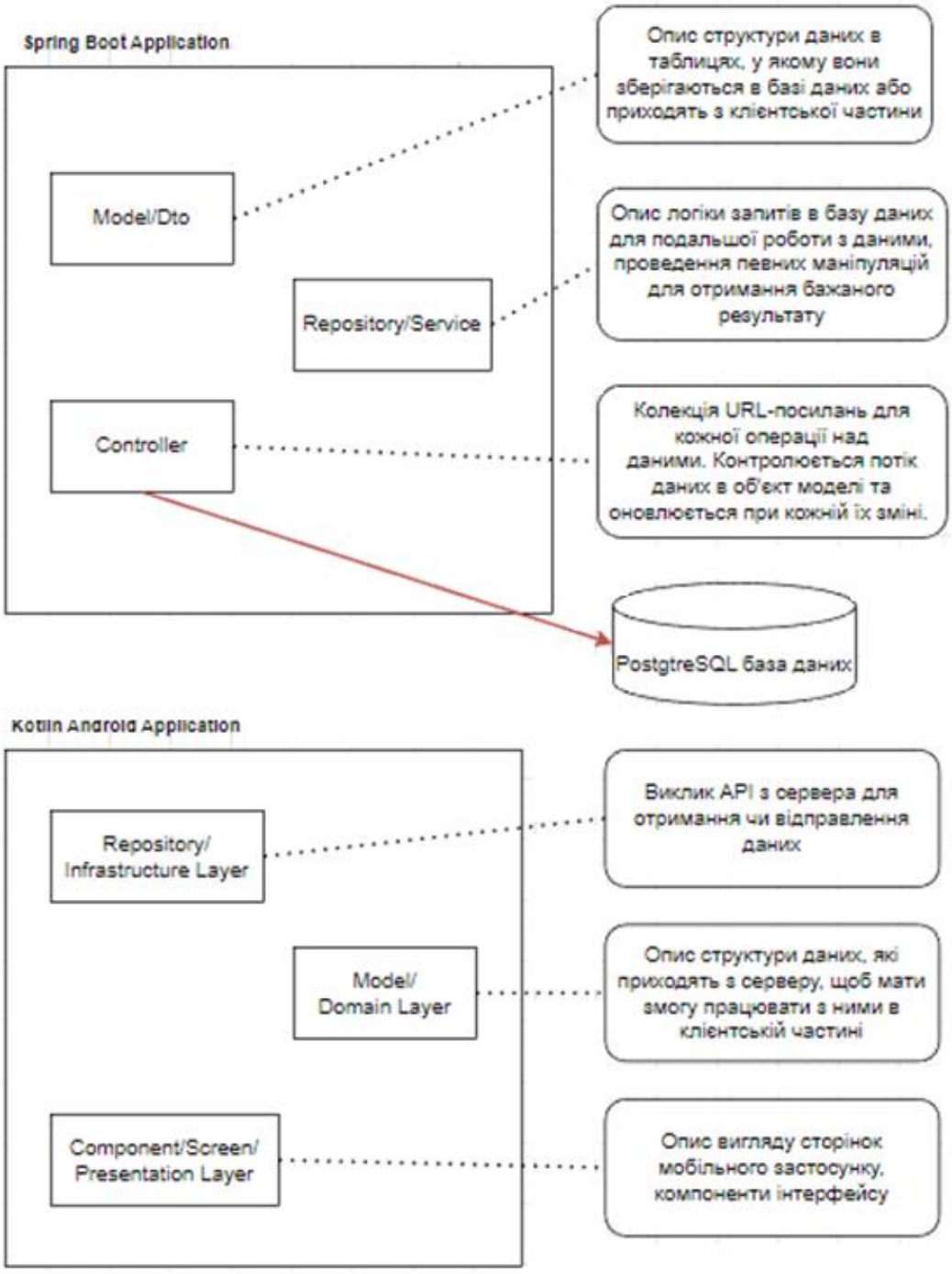

Рисунок 3.2 – Діаграма компонентів системи

Основні компоненти Clean Architecture:

1) доменний шар (Domain Layer) – визначає бізнес-логіку додатку. Він містить моделі даних, визначення правила бізнес-процесів. У випадку Kotlin та

Jetpack Compose, цей шар може містити Kotlin-класи, що представляють доменну логіку додатку;

2) шар представлення (Presentation Layer) – відповідає за відображення даних та взаємодію з користувачем. Використовуючи Jetpack Compose, цей шар може включати Composable-функції, що створюють візуальний інтерфейс користувача. Шар представлення взаємодіє з доменним шаром для отримання необхідних даних та виконання бізнес-логіки;

3) шар інфраструктури (Infrastructure Layer) – відповідає за взаємодію з зовнішніми ресурсами, такими як база даних, мережа або зовнішні сервіси. Він містить реалізації репозиторіїв, мережевих клієнтів та інших компонентів, що забезпечують доступ до зовнішніх джерел даних. У випадку мобільних додатків на Kotlin з Jetpack Compose, цей шар може включати реалізації взаємодії з базою даних, роботу з API та інші зовнішні ресурси.

3.7 Обґрунтування вибору середовища розробки

Вибір середовища розробки залежить від функціональних можливостей, продуктивності, комфортності роботи з середовищем, наявність плагінів та інших інструментів для полегшення розробки програм. Також на вибір середовища розробки впливає і оперативна система та можливості робочої машини, комп’ютера.

Найбільш популярним та зручним середовищем для розробки на мові Java/Kotlin є Intellij IDEA – інтегроване середовище розробки. А зі студентською підпискою цей сервіс дає багато функцій для створення якісного продукту.

Для написання серверної частини буде використовуватись Kotlin зі Spring фреймворком, тому Intellij IDEA ідеально підходить, адже має його підтримку, що дозволяє з легкістю створювати проекти на цих технологіях. Він автоматично визначає залежності та конфігурації Spring, що допомагає економити час розробки. Також це середовище має автодоповнення та кодогенерацію, що пришвидшує

процес написання програми, а із багатофункціональним дебагером можна легко відстежувати помилки в коді. Сюди також вбудовано систему контролю версій і це дозволяє легко працювати з Git, щоб зберігати зміни в коді в репозиторії.

Для написання клієнтської частини також буде використовуватись Kotlin. Android програми зручно створювати у Android Studio – одному з найпоширеніших середовищ для розробки мобільних додатків, що розробляється компанією Google.

Це середовище надає широкі можливості для мобільної розробки, включає в себе різноманітні інструменти для створення інтерфейсу користувача, збірки та тестування додатків, роботи з базами даних та іншими функціями.

Обидва середовища мають багато розширень та плагінів, що дозволяють налаштовувати робоче середовище для потреб конкретного проекту. Наприклад було використано Linter, який перевіряв правопис та вигляд файлу з кодом загалом, щоб всюди мати однаковий стиль та зовнішній вигляд, що покращує читабельність коду [17].

Висновки до розділу 3

Для розробки мобільного застосунку було обрано мови Kotlin з використанням фреймворку Spring для написання серверу та набору бібліотек Jetpack Compose для клієнтської частини. Розробка відбуватиметься у середовищах Android Studio та Intellij IDEA, базою даних буде виступати PostgreSQL. Було також описано функції, що можливі для використання з Google Maps SDK для роботи з мапою, GPS та акселерометром.

Використання засобів розробки, описаних вище, є обґрунтованим і зручним у контексті даного проекту. Вони дозволяють реалізувати потрібну функціональність системи, забезпечують швидкість та надійність, спрощують розробку та підвищують продуктивність розробника.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

4.1 Архітектура програмного забезпечення

Розроблений мобільний додаток має модульну архітектуру, що сприяє зручному розподілу коду на відповідні компоненти. На рисунку 4.1 показана повна архітектура програми.

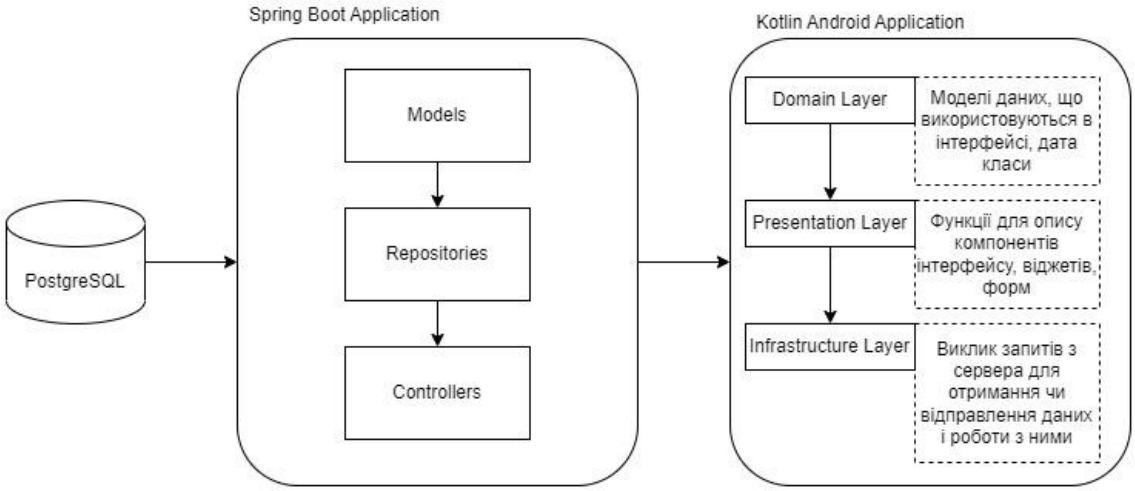

Рисунок 4.1 – Схема бази даних системи

База даних підключена до серверної частини, яка робить виклики для отримання потрібних даних. Класи-контролери мають в собі URL-строки для отримання певних даних. Рівень репозиторіїв потрібен для опису бізнес-логіки та алгоритму обробки даних, що надходять. Саме тут розраховується логіка краудсорсингу. Класи-моделі дублюють структуру таблиць у базі даних, накладають певні обмеження на значення та перевіряють чи правильні взагалі дані приходять.

Клієнтська частина відповідає за інтерфейс та правильне відображення даних у мобільному застосунку. Цим займається шар представлення (Presentation Layer). Доменний шар, знову ж таки, дублює структуру таблиць, та описує які саме дані та в якому форматі мають надходити системі. Програма має це значити, щоб в подальшому правильно їх обробити. Шар інфраструктури (Infrastructure Layer)

виконує роль репозиторію, роблячи запити на сервер для відправки або отримання потрібних даних.

4.2 Структура бази даних

Розроблений мобільний застосунок використовує реляційну базу даних PostgreSQL, де зберігається уся інформація програми. Схема бази даних наведена на рисунку 4.2.

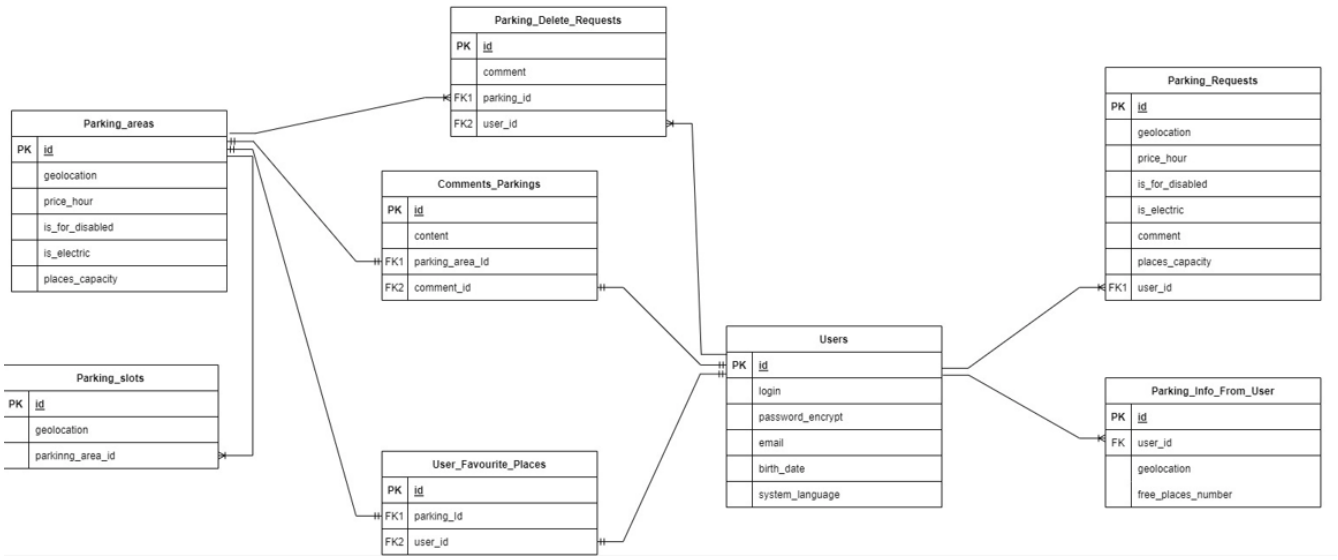

Рисунок 4.2 – Схема бази даних системи

База даних складається з 8 таблиць, основними є Parking_areas та Users, на базі яких створюються усі інші. Між цими двома таблицями зв’язок багато до багатьох, адже у проміжку між ними створюються також таблиці для коментарів від користувачів, список улюблених паркувань користувачів та запит на видалення паркозони з певних причин.

Таблиці Parking_Requests (запит на додавання нового паркомісця на мапу) та Parking_Delete_Requests (запит на видалення паркомісця) винесені окремо, адже обробляються за допомогою краудсорсингу, тож ці дані необхідно мати окремо від основної таблиці з паркуваннями, що зображені на мапі в інтерфейсі мобільного застосунку.

4.3 Діаграма прецедентів

Перед початком написання логіки програми необхідно визначити функціональні можливості, які будуть доступні користувачам у системі. На рисунку 4.3 зображено діаграму прецедентів системи.

Основна мета діаграми прецедентів полягає в ідентифікації акторів (користувачів системи) та прецедентів (функціональних вимог), а також у візуалізації зв'язків між ними.

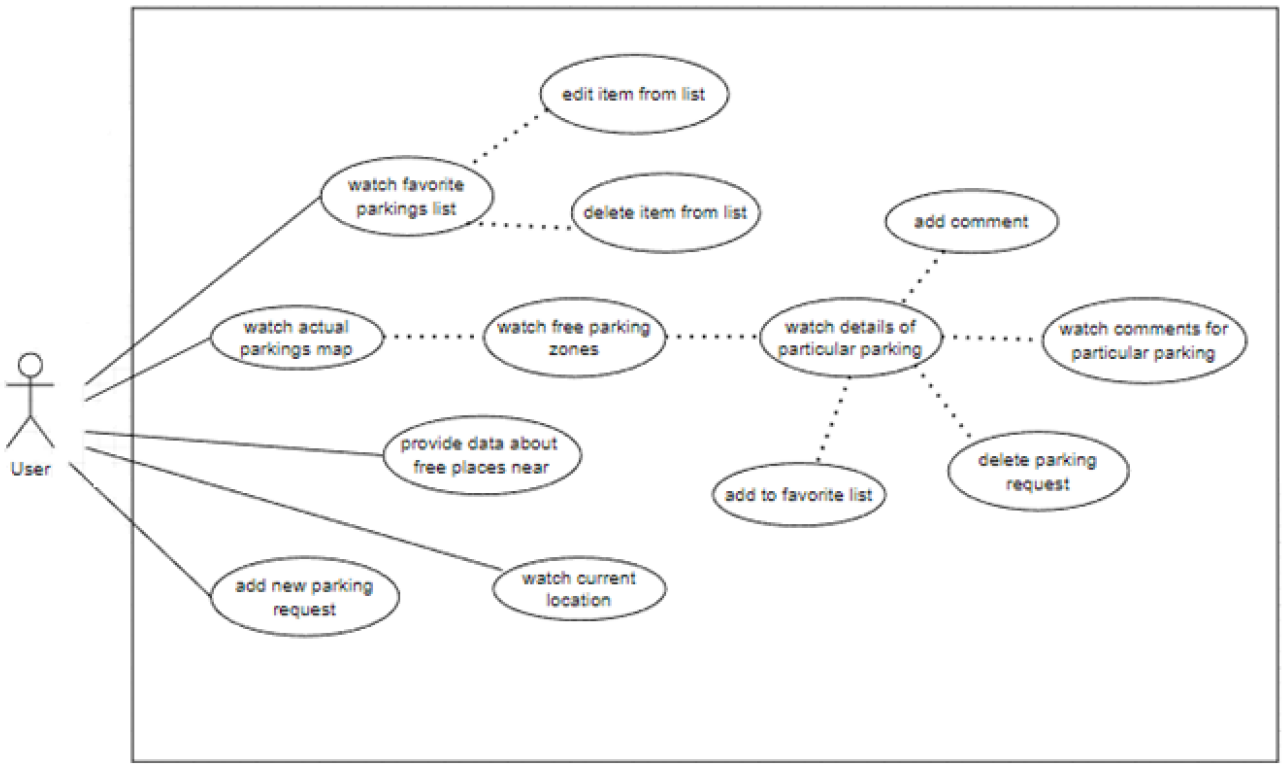

Рисунок 4.3 – Діаграма прецедентів мобільного застосунку

Аналізуючи діаграму, можна зазначити наступні можливості, що доступні користувачеві у створеній програмі:

1) перегляд усіх наявних паркувань на мапі застосунку, можливість відкрити детальну інформацію кожного з них в новому екрані, де додаються функції перегляду коментарів або ж додавання власного, створення запиту на видалення паркування або додавання його у свій список улюблених;

2) перегляд списку своїх улюблених паркувань, можливість редагувати;

3) самостійно надати інформацію про вільні паркування поблизу;

4) перейти до своєї поточної локації на мапі;

5) створення запиту на додавання нового паркування на мапу.

User Flow діаграма створена для більш детального розуміння того, які кроки та дії користувач може виконувати в системі (рис 4.4).

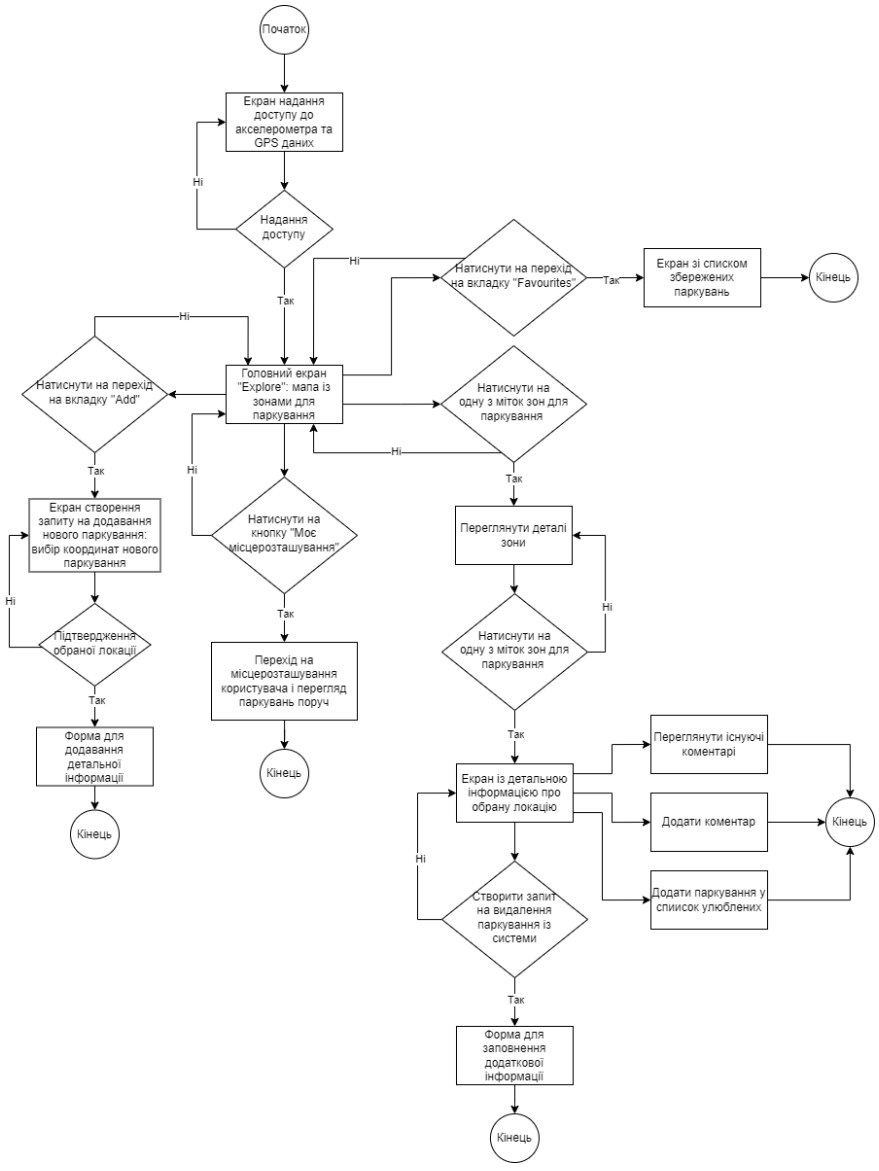

Рисунок 4.4 – Діаграма екранів User Flow

Вона допомагає розкрити потенційні шляхи користувачів у системі, виявити можливі проблеми або недоліки в проектуванні інтерфейсу та покращити загальний досвід користувача.

Робота користувача з програмою починається з форми із проханням надати доступ до даних з GPS та акселерометра. Головним екраном мобільного застосунку є мапа з відображеними мітками та полігонами, що символізують зони для паркування. Користувачеві надаються функції перегляду усіх наявних локацій, детальної інформації про кожну з них, переглянути коментарі та додати свій власний, можна перейти до свого поточного місцезнаходження для пошуку паркомісця поряд, а також створити запит на видалення з мапи обраного паркування.

Наступний екран надає можливість створювати запит на додавання нової зони для паркування на мапу. Для цього користувач спершу обирає координати нової локації, після чого відкривається форма для заповнення детальної інформації.

Ще один екран системи містить список збережених улюблених місць для паркування, які можна редагувати, змінюючи ім’я, або ж видалити з цього списку.

4.4 Алгоритм обробки даних на клієнтській частині

На клієнтську частину безперервно отримуються дані з GPS та акселерометра. Ці значення аналізуються для того, щоб у фоновому режимі змінювати кількість вільних/зайнятих місць у зонах для паркування.

Коли водій зупиняється, взаємодія з GPS та акселерометром може бути використана для виявлення цього стану та подальшої обробки.

На рисунку 4.5 наведено приклад алгоритму аналізу поведінки водія.

Основними варіантами подій після зупинки водія є наступні:

1) якщо водій зупинився у межах відомої для системи зони паркування і не змінює місцезнаходження протягом 1-2 хвилин, то кількість вільних місць на цій парковці зменшується. Це означає, що система відстежує тривалість зупинки і автоматично оновлює інформацію про вільні місця на мапі у фоновому режимі без взаємодії з водієм;

2) якщо зупинка здійснена у невідомій локації і триває більше 10 хвилин без зміни координат, система виведе спливаюче повідомлення із запитанням користувачеві чи не бажає він створити запит на додавання цієї локації як нової зони для паркування. Користувач може надати позитивну відповідь, що сприятиме додаванню нової зони на мапу, що потенційно стане доступною для інших користувачів або ж нічого не робити.

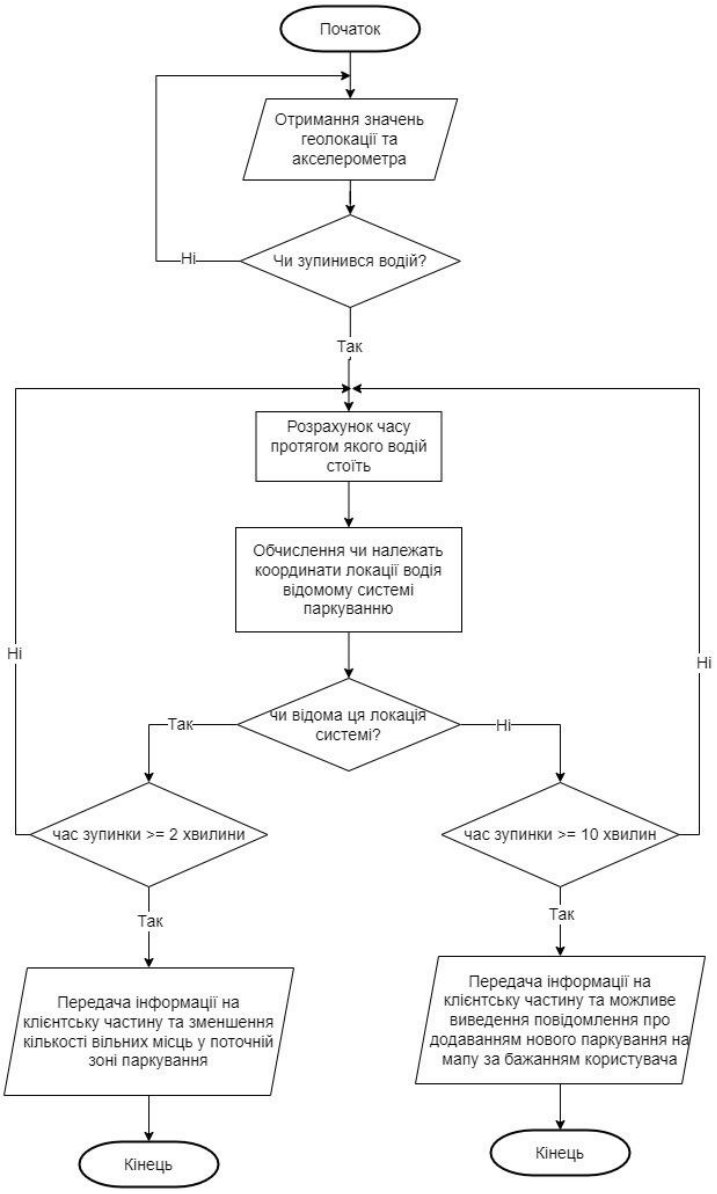

Рисунок 4.5 – Алгоритм аналізу поведінки водія

Поведінка системи коли водій починає рухатися після зупинки схожа і наведена нижче:

1) якщо місце, де була виконана зупинка, входить в одну з відомих зон паркування на мапі, система автоматично оновлює стан цієї зони, вказуючи, що місце для паркування стало вільним. Це дозволяє іншим користувачам бачити актуальну інформацію про вільні паркомісця;

2) якщо місце, де була виконана зупинка, є невідомим системі, тоді програма може повторно запитати користувача, чи бажає він додати цю нову локацію як зону для паркування.

Таким чином, система використовує дані з GPS та акселерометра, а також взаємодію з користувачем, щоб ефективно відстежувати зупинки та рух автомобіля, оновлювати інформацію про вільні місця на мапі та надавати можливість користувачам додавати нові зони для паркування, без суттєвої взаємодії з водієм.

Проте все ще існує окрема форма на додавання нового паркування або видалення існуючого, яку можна заповнити незалежно від місцезнаходження.

4.5 Алгоритм обробки даних на сервері

Для збору даних від користувачів у програмі є кілька варіантів:

1) створення запиту на додавання нового паркомісця на мапі;

2) створення запиту на видалення певного паркомісця з мапи;

3) діалогове вікно для введення кількості вільних місць поруч з водієм на паркуванні.

У випадку надходження запиту на додавання чи видалення паркомісця, дані з форми надходять на сервер, де перевіряється геолокація, відбувається пошук схожих запитів для подальшого аналізу, порівняння та прийняття рішення чи змінювати базу даних додаванням або видаленням зони паркування.

Так як програма отримує велику кількість даних, то їх треба коректно обробляти, робити безліч перевірок для того, щоб у базу даних не потрапляли

невірні дані. Для цього на серверній частині реалізовано декілька етапів роботи з даними (рис. 4.6).

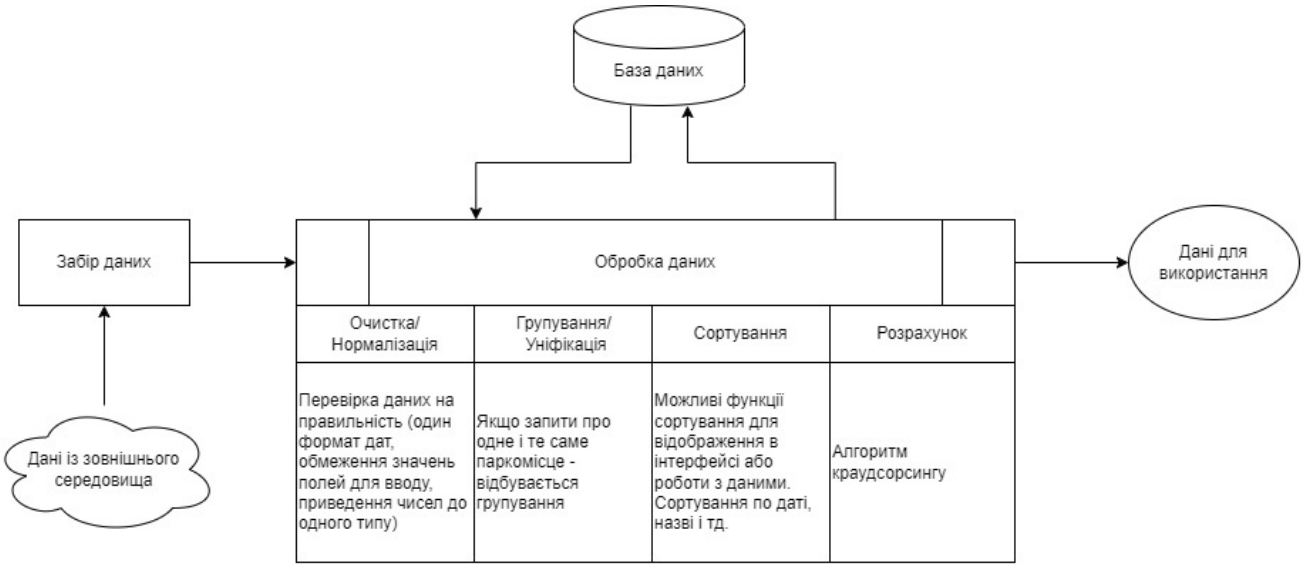

Рисунок 4.6 – Етапи роботи з даними

Аналізуючи рисунок 4.6, можна виділити такі етапи обробки даних:

1) очистка/нормалізація включає в себе видалення непотрібних символів, пробілів або спеціальних символів з даних. Також для певних методів на сервері можливе перетворення тексту у нижній регістр та приведення дат до одного стандарту. Це полегшує подальшу роботу з даними та їх аналіз;

2) групування (уніфікація) зменшує кількість даних для майбутнього зберігання, обробки. Відбувається групування за певними характеристиками або параметрами. Наприклад, групування запитів за локацією, за певною зоною паркування. Уніфікація ж означає об’єднання схожих або даних, що дублюються, щоб забезпечити їх консистентність;

3) сортування впорядковує дані за певними критеріями, наприклад алфавітний порядок чи дата. Такі операції полегшують подальшу обробку даних та пошук по них або навіть їх відображення в інтерфейсі;

4) алгоритм краудсорсингу описано детальніше у наступному підрозділі.

4.6 Алгоритми краудсорсингу

Дані, що приходять з клієнтської частини, тобто з мобільного застосунку, оброблюються на сервері.

Одним з важливих аспектів серверної обробки є алгоритм краудсорсингу. На сервері реалізується алгоритм, який збирає та обробляє запити користувачів на додавання або видалення паркування в системі. На рисунку 4.7 наведено алгоритм видалення паркування з бази даних.

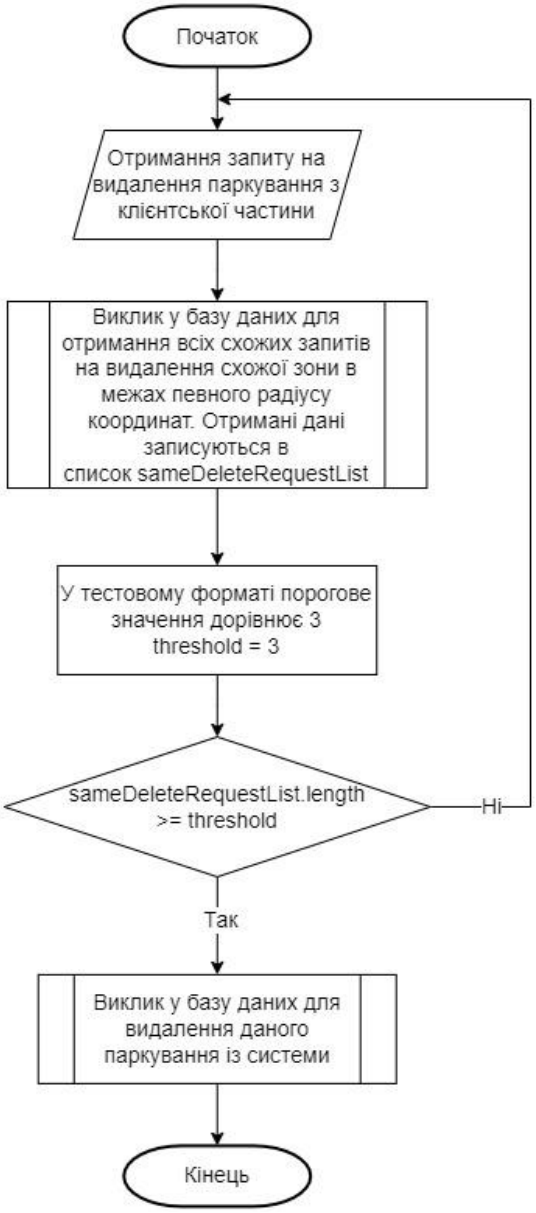

Рисунок 4.7 – Алгоритм обробки запитів від користувачів на видалення паркування

Відбувається аналіз даних, перевіряючи чи існують вже схожі запити у таблиці із запитами про видалення або додавання певної зони паркування.

Додатково є певна змінна в системі - число, що обмежує кількість схожих запитів і якщо кількість запитів про певне паркування перевищує це значення, то дані про паркування буде видалено або додано, в залежності від функції. У тестовому форматі під час розробки це значення дорівнює 3.

Окрім обробки даних, на серверній частині також можуть бути виконані інші завдання, такі як збереження даних у базу даних, керування роботою системи, забезпечення безпеки, виконання розрахунків, обмін даними з іншими системами, генерація репортів або запис системних логів у файли для легшого подальшого трасування програми розробниками при виникненні помилок. Усі ці операції здійснюються з використанням вибраних технологій та інструментів, які допомагають забезпечити ефективну та надійну роботу серверної частини системи.

4.7 Структура проекту

У програмі, що розробляється, замість представлення є окрема клієнтська частина мобільного додатку, тому на сервері написані лише класи для обробки інформації, що приходить з бази даних або інтерфейсу. З такою архітектурою користувач не навантажитиме телефон зайвими даними і це виглядає більш безпечно з точки зору зберігання усіх даних в одному місці.

На рисунку 4.8 зображено структуру папок серверної частини. Головним файлом для запуску програми є ProjectApplication.kt. Intellij IDEA автоматично визначає, що даний проект є Kotlin та Spring і готує потрібну конфігурацію для запуску.

Після успішного запуску, у консолі будуть відображатись повідомлення-логи про те, що сервер працює коректно і можна починати робити запити в базу даних або з неї.

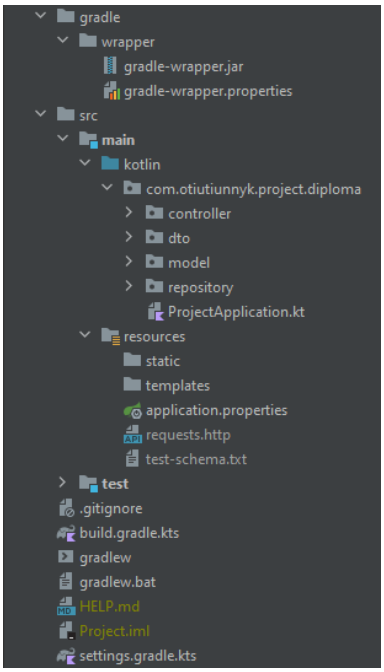

Рисунок 4.8 – Файлова структура сервера

Файли з наявним словом gradle необхідні для управління залежностями проекту, збирання проекту та налаштування його конфігурації. У свою чергу application.properties файл використовується для налаштування конфігурації програми. Цей файл містить ключ-значення (key-value) пари, де ключ представляє налаштування, а значення – проставлений програмістом параметр. Зазвичай він містить інформацію про конфігурацію бази даних, сервера, логування тощо.

Головним файлом у структурі клієнтської частини є MainActivity.kt, де в методі onCreate вказується структура інтерфейсу мобільного застосунку та можливі попередні розрахунки, наприклад отримання дозволу на обробку персональних даних про місцезнаходження, щоб на мапі у головному меню відображалась поточна локація користувача. У цьому проекті так само існують gradle файли для вказування залежностей та налаштувань проекті для його коректного запуску. На рисунку 4.9 зображено структуру папок клієнтської частини.

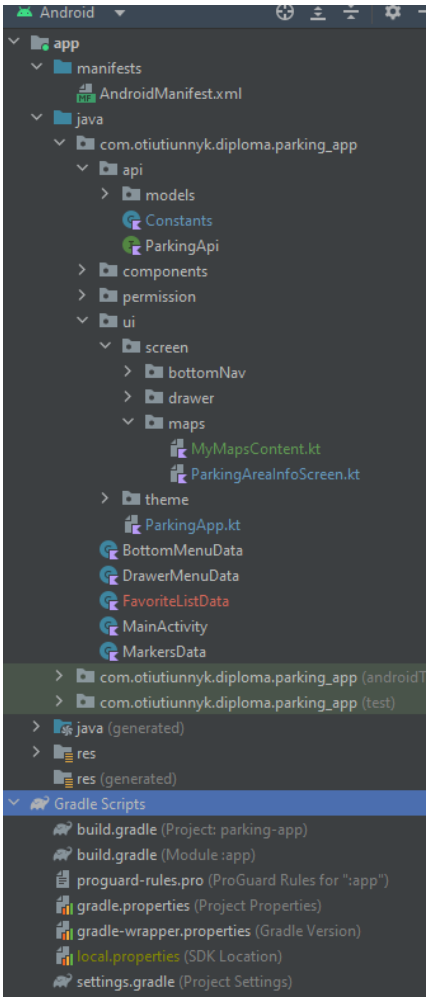

Рисунок 4.9 – Файлова структура клієнта

Також обов’язковим компонентом Android проекту є AndroidManifest.xml файл, що містить важливу інформацію про додаток.

1. Пакет (Package) – визначає унікальний ідентифікатор програми, який зазвичай відповідає ім’ям пакетів у зворотному доменному форматі.

2. Дозволи (Permissions) – визначає набір дозволів, які додаток вимагає для доступу до різних системних ресурсів або функціональності, таких як доступ до мережі, камери, GPS.

3. Компоненти додатку (Components) – визначає основні компоненти додатку, такі як Activity (екран програми), Service (фоновий процес), BroadcastReceiver (приймач відомостей від системи) та Content Provider (доступ до даних).

4. Конфігурація додатку (Application Configuration) – визначає додаткову конфігурацію додатку, таку як мінімальну та цільову версії платформи Android, потрібні бібліотеки, теми оформлення.

Висновки до розділу 4

У розділі було наведено діаграми класів, бази даних, прецедентів, обробки даних та розробленого алгоритму краудсорсингу.

Програма з мапою паркувань, що надаватиме інформацію про зайнятість місць являє собою мобільний застосунок, який складається з клієнтської частини, що знаходиться на телефоні користувача, серверної частини, що запущена на комп’ютері й напряму спілкується зі створеною базою даних, яка також знаходиться на комп’ютері.

На серверній частині системи знаходиться більша частина обробки даних, які приходять з клієнтської частини. У цьому контексті сервер може виконувати різноманітні операції, такі як обчислення, фільтрація, сортування та аналіз даних.

Клієнтська частина відповідає за побудову зручного інтерфейсу, збір даних від користувачів, їх обробку та виведення на екран мобільного телефону.

5 РОБОТА КОРИСТУВАЧА З МОБІЛЬНИМ ЗАСТОСУНКОМ

5.1 Системні вимоги

Створена система розроблена з урахуванням власників мобільних пристроїв, які працюють під управлінням операційної системи Android. Вимоги до пристроїв включають наявність ОС Android версії 11 і вище. Враховуючи статистику, ці версії охоплюють близько 91% користувачів Android-пристроїв, що забезпечує широку базу користувачів для системи.

Для належного функціонування системи необхідний дозвіл користувача на використання даних GPS та акселерометра. Ці дані використовуються для отримання інформації про місцезнаходження та рух пристрою. Дозвіл на використання цих датчиків дозволить системі отримувати необхідну інформацію для забезпечення функціональності, пов'язаної з розпізнаванням стану паркомісць та аналізом руху автомобіля.

Додатково, для коректної роботи системи необхідний доступ до Інтернету. Це дозволить системі отримувати актуальні дані, виконувати запити до сервера для оновлення інформації про зони для паркування та здійснювати інші мережеві операції.

5.2 Запуск застосунку та його основні можливості

Для того, щоб запустити застосунок, потрібно спершу відкрити Intellij IDEA, щоб запустити серверну частину, після чого у клієнтської частини з’явиться доступ до бази даних і можна буде робити запити.

Клієнтська частина запускається через Android Studio, після чого на мобільному телефоні відкривається програма. У випадку, якщо немає можливості

підключитися через USB порт до комп’ютера напряму, можна використовувати емулятор (англ. emulator) – це інструмент, який дозволяє розробникам створювати та запускати віртуальні пристрої Android на своєму комп'ютері. Емулятор дозволяє симулювати різні конфігурації та версії Android-пристроїв, що дозволяє розробникам тестувати свої додатки на різних пристроях без фізичного наявності кожного з них. Він включає в себе віртуальний екран, на якому можна відображати та тестувати інтерфейс додатків, а також симулює різні апаратні функції, такі як камера, сенсори, GPS, акселерометр і багато іншого. Можна також змінювати різні налаштування, обирати різні моделі телефону, розміри екрану для того, щоб виявити можливі помилки на певних виняткових моделях, конфігураціях.

При запуску програми, на початковому екрані з'являється запит на дозвіл на використання геолокаційних даних. На рисунках 5.1 та 5.2 наведено екрани із запитом на дозвіл оброблення даних та надання їх користувачем.

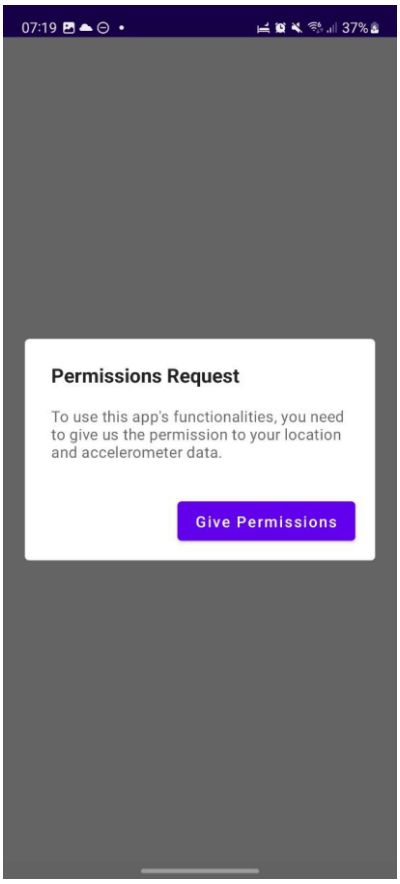

Рисунок 5.1 – Прохання надання доступу до геолокації

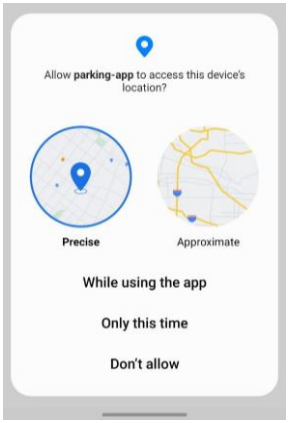

Рисунок 5.2 – Підтвердження надання доступу до геолокації

Перш за все, користувачеві відкривається сторінка “Explore” з мапою, на якій зображені усі наявні в системі зони для паркування (рис. 5.3).

Кожна зона для паркування представлена у вигляді полігону і міткою посередині. При натисканні на якусь з них, буде спливати інформація про обрану паркозону внизу екрану (рис. 5.4). Обраний маркер зони буде підсвічуватись помаранчевим кольором, повідомляючи користувачеві інформація якої зони переглядається. Червоним позначені повністю зайняті зони, а зеленим – зони, де є ще вільні місця.

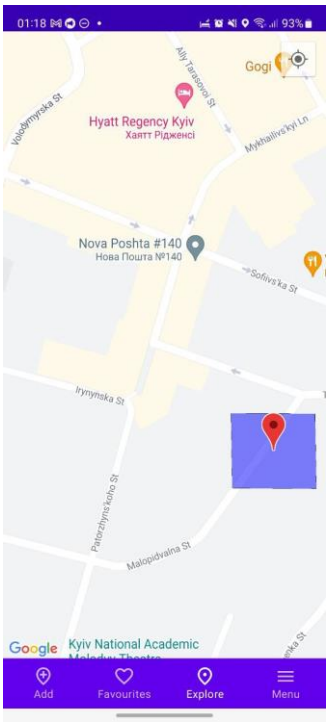

Рисунок 5.3 – Головний екран застосунку

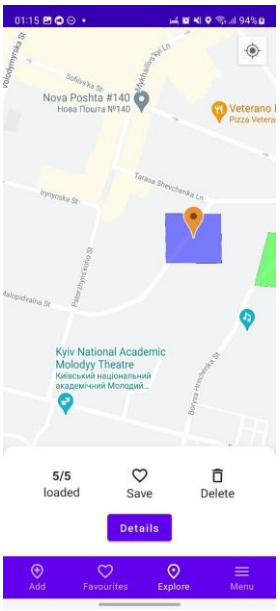

Рисунок 5.4 – Екран з детальною інформацією

На сторінці деталей паркозони користувач може ознайомитися з усіма доступними даними, такими як ціна за годину, завантаженість, інформацію про додаткові можливості (наявність місць для інвалідів, електричних машин). Користувач також може переглянути відгуки і коментарі інших користувачів про цю зону для паркування, щоб отримати додаткову інформацію та оцінку її якості та додати свій власний (рис. 5.5).

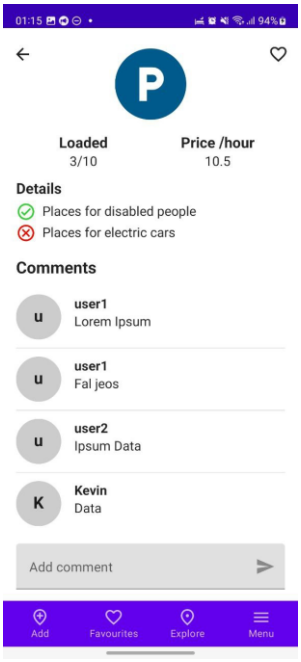

Рисунок 5.5 – Екран з детальною інформацією про паркування

При переході на вкладку “Add”, що обирається з нижнього меню програми, користувач отримує можливість створити запит на додавання нового паркомісця до бази даних.

На цій вкладці користувач бачить мапу, де маркер розташований у центрі екрану. Мапа може рухатись, що дозволяє користувачеві обрати потрібні координати. Користувач може пересувати мапу шляхом проведення пальцем по екрану або за допомогою жестів зумування (англ. zoom) – зміна масштабу зображення, мапи (рис. 5.6).

Рисунок 5.6 – Екран для додавання нового паркомісця

Коли користувач встановив мітку на бажане місце, він може натиснути на кнопку в правому нижньому кутку для підтвердження координатів. Після цього відкриється форма для додавання додаткової інформації про нове паркомісце, наприклад, назву місця, тип паркомісця, ціну та будь-які коментарі або вказівки (рис. 5.7).

Після внесення всіх необхідних даних користувач може натиснути кнопку “Підтвердити”, щоб відправити запит на додавання нового паркомісця до бази даних або відмінити його створення кнопкою “Відмінити”.

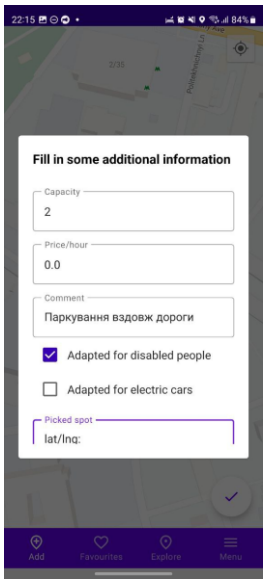

Рисунок 5.7 – Форма запиту на додавання нового паркомісця

Третьою вкладкою є “Favourites”, де відображається список збережених улюблених паркувань (рис. 5.8). Для кожного паркомісця може бути вказана його назва, яку користувач ввів при додаванні паркомісця до списку улюблених та його локація.

Користувач може прокручувати список, щоб переглянути всі збережені улюблені паркомісця. Кожен елемент списку може бути вибраний користувачем для отримання додаткової інформації або взаємодії з паркомісцем.

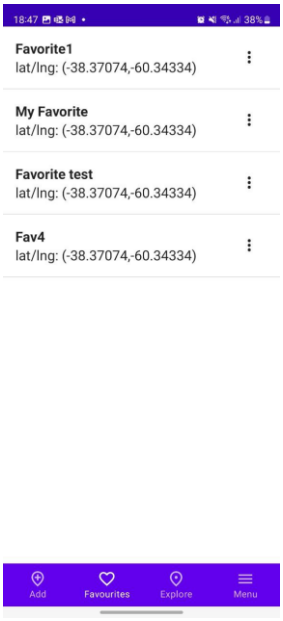

Рисунок 5.8 – Екран зі списком улюблених паркувань

Останньою вкладкою є відкриття бокового меню, у якому є різні налаштування (зміна персональних даних, конфігурації програми, інформація про розробників тощо) (рис. 5.9).

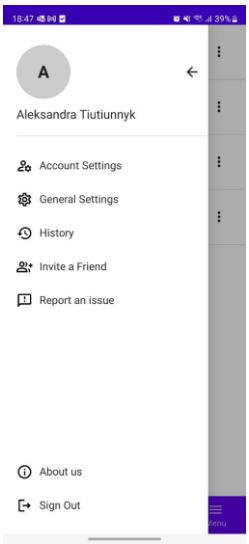

Рисунок 5.9 – Екран зі списком улюблених паркувань

Якщо водій зупиняється у невідомому системі місці, то на екрані буде з’являтися форма із пропозицією створити запит на додавання даної локації на мапу. Додатково мануально користувач може заповнити форму з вказуванням актуальної кількості вільних місць на паркуванні (рис. 5.10).

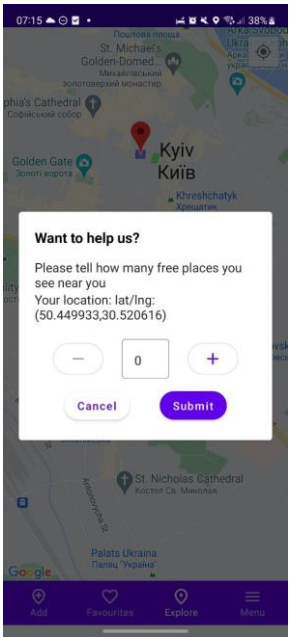

Рисунок 5.10 – Форма для отримання інформації про паркування поруч

На поле вводу є обмеження на запис від’ємних чисел, а також неможливо ввести число більше за ємкість зони паркування.

Висновки до розділу 5

У розділі було продемонстровано роботу мобільного застосунку, розроблений функціонал та варіанти взаємодії з ним. Користувач може легко взаємодіяти з додатком і здійснювати різні дії, такі як перегляд інформації про паркомісця, додавання нових місць для паркування, вибір улюблених місць, а також зміна налаштувань.

Додаток забезпечує зручний та інтуїтивно зрозумілий інтерфейс, який спрощує використання його функцій. Користувач може легко переміщатись між екранами системи, виконувати необхідні дії і отримувати актуальну інформацію про кількість вільних місць для паркування.

ВИСНОВКИ

У ході виконання роботи було проаналізовано наявні підходи до використання краудсорсингу в транспортних системах. Також було досліджено потреби транспортних систем у використанні краудсорсингу та визначення пріоритетних напрямків для впровадження цих інструментів, після чого було розроблено алгоритм, який на основі вхідних даних від користувачів здатний оновлювати інформацію про зайнятість паркомісць і додавати нові місця для паркування на мапу після отримання в обробку певної кількості схожих запитів про одне місце від користувачів.

Було виявлено, що краудсорсинг може бути використаний для розв'язання різноманітних проблем та задач у транспортному секторі, зокрема, поліпшення мобільності водіїв авто, підвищення ефективності роботи паркувань та зменшення негативного впливу на довкілля через зниження часу, проведеного у пошуках вільного місця для зупинки.

У результаті було розроблено мобільний застосунок, що містить мапу з місцями для паркування, детальною інформацією про кожне з них, а також використовує дані з GPS та акселерометра для отримання точніших даних про переміщення водіїв та зайнятість місць на паркуваннях.

Додатково створено можливість мати список улюблених паркомісць та додавання коментарів про паркування.

Даний застосунок дозволяє користувачам швидко та легко знаходити вільне паркомісце, що робить процес паркування більш ефективним та зручним.

Такі застосунки можуть допомогти державі проаналізувати у яких районах потрібно побудувати додаткові місця для паркування, а де, навпаки, їх зменшити. Це допоможе зробити місто більш зручним для водіїв та корисним для мешканців.

Дипломну роботу виконано в рамках проекту Driver's Behavior Cognition спільно із Політехнічним інститутом м. Томар, Португалія.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Deshun H., Erkang C. Automatic parking space detection method based on deep learning. Chinese journal of lasers. 2019.

2. Wu Q., Zhang Y. Parking lots space detection, machine learning. Carnegie mellon university. 2006.

3. Gupta A. Best Practices in Kotlin. https://medium.com. URL: https://medium.com/bobble-engineering/best-practices-in-kotlin-45619094d711 (дата звернення: 27.04.2023).

4. Chris K. REST API Best Practices – REST Endpoint Design Examples. freeCodeCamp.org. URL: https://www.freecodecamp.org/news/rest-api-best-practices-rest-endpoint-design-examples/ (дата звернення: 24.04.2023).

5. Alumni A. SpotAngels – Using crowds to solve the car parking problem. Digital Innovation and Transformation. URL: https://d3.harvard.edu/platform-digit/submission/spotangels-using-crowds-to-solve-the-car-parking-problem/ (дата звернення: 11.04.2023).

6. Demand-Responsive Parking Pricing. SFMTA. URL: https://www.sfmta.com/demand-responsive-parking-pricing (дата звернення: 13.04.2023).

7. The ParkMan Team. Crowdsourcing vs Sensors: How to Successfully Manage On-Street Parking. ParkMan Blog. URL: https://blog.parkman.io/benefits-of-crowdsourcing-in-parking-industry/ (дата звернення: 27.04.2023).

8. Мелешко В. В., Аврутов В. В., Бондар П. М. Мікроакселерометри та їх випробування: Навч. посіб. 2008.

9. IntelliJ IDEA – the Leading Java and Kotlin IDE. JetBrains. URL: https://www.jetbrains.com/idea/ (дата звернення: 28.04.2023).

10. Недашківський О. Л. Мова Kotlin для Розробки програмного забезпечення мобільних пристроїв. Київ : КПІ ім. Ігоря Сікорського, 2022.

11. Pramod K. S. Understanding the Basics of Spring vs. Spring Boot. 2018.

URL: https://dzone.com/articles/understanding-the-basics-of-spring-vs-spring-boot (дата звернення: 04.05.2023).

12. Get started with Jetpack Compose | Android Developers. Android Developers. URL: https://developer.android.com/jetpack/compose/documentation (дата звернення: 17.04.2023).

13. Maps SDK for Android overview | Google for Developers. Google for Developers. URL: https://developers.google.com/maps/documentation/android-sdk/overview (дата звернення: 24.04.2023).

14. Svennerberg G. Beginning Google Maps API 3. 2-ге вид. 2010. 328 с.

15. Chandrakant K. Introduction to Transactions | Baeldung on Computer Science. Baeldung on Computer Science. URL: https://www.baeldung.com/cs/transactions-intro (дата звернення: 02.05.2023).

16. Панченко І. PostgreSQL: учора, сьогодні, завтра. Відкриті системи. СКБД. 3-тє вид. СпБ : OSP, 2015.

17. Darwin I. F. Checking C Programs with Lint. O'Reilly Media, Inc., 1988. 72 с.

ДОДАТОК А

Інструментальні засоби краудсорсингу в транспортних системах

Програмний код

Аркушів 14

Київ — 2023

ParkingConroller.kt

```
@RestController
@RequestMapping("/parkings")
class ParkingController(
    private val parkingRepository: ParkingAreaRepository,
    private val newParkingRequestRepository: NewParkingRequestRepository,
    private val deleteParkingRequestRepository: DeleteRequestRepository,
    private val userRepository: UserRepository
) {
    @GetMapping("")
    fun getAllParking(): MutableIterable<ParkingArea> = parkingRepository.findAll()
    @GetMapping("/{id}")
    fun getParkingById(@PathVariable id: Long): Optional<ParkingArea> =
parkingRepository.findById(id)
    @DeleteMapping("/{id}")
    fun deleteParkingArea(@PathVariable parkingAreaId: Long) =
deleteParkingRequestRepository.deleteById(parkingAreaId)
    @GetMapping("/new")
    fun getAllNewParkingRequest(): MutableIterable<NewParkingRequest> =
newParkingRequestRepository.findAll()
    @GetMapping("/new/{id}")
    fun getNewParkingRequest(@PathVariable id:Long): Optional<NewParkingRequest> =
newParkingRequestRepository.findById(id)
    @PostMapping("/new")
    fun createNewParkingRequest(@RequestBody @NonNull newParkingRequest:
NewParkingRequestDto): NewParkingRequest {
        val user = userRepository.findById(newParkingRequest.userId)
       return newParkingRequestRepository.save(newParkingRequest.toApiEntity(user.get()))
    }
}
```

ParkingAreaDto.kt

```
data class ParkingAreaDto(
    var geolocation: String,
    var capacity: Int,
    var isForDisabledPerson: Boolean? = null,
    var isForElectricCar: Boolean? = null,
    var priceHour: Double? = null
) {
    fun toApiEntity(): ParkingArea = ParkingArea(
        geolocation,
        capacity,
        isForDisabledPerson,
```

```
        isForElectricCar,
        priceHour
    )
    fun toEntity(): ParkingAreaDto = ParkingAreaDto(
        geolocation,
        capacity
    )
}
```

ParkingArea.kt

```
@Entity
@Table(name = "parking_areas")
class ParkingArea() {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @get:Column(name = "id")
    var id: Long? = null
    @get:Column(name = "geolocation", nullable = false)
    var geolocation: String? = null
    @get:Column(name = "capacity", nullable = false)
    var capacity: Int? = null
    @get:Column(name = "is_for_disabled_person")
    var isForDisabledPerson: Boolean? = null
    @get:Column(name = "is_for_electric_car")
    var isForElectricCar: Boolean? = null
    @get:Column(name = "price_hour")
    var priceHour: Double? = null
    constructor(geolocation: String, capacity: Int) : this() {
        this.geolocation = geolocation
        this.capacity = capacity
    }
    constructor(
        geolocation: String, capacity: Int,
        isForDisabledPerson: Boolean?, isForElectricCar: Boolean?,
        priceHour: Double?
    ) : this(geolocation, capacity) {
        this.isForDisabledPerson = isForDisabledPerson
        this.isForElectricCar = isForElectricCar
        this.priceHour = priceHour
    }
}
```

MainActivity.kt

```
class MainActivity : ComponentActivity() {
    @OptIn(ExperimentalPermissionsApi::class)
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            ParkingappTheme {
                // A surface container using the 'background' color from the theme
                Surface(color = MaterialTheme.colors.background) {
                    RequestPermission(permission = ACCESS_FINE_LOCATION)
                }
            }
        }
    }
}
```

RequestPermission.kt

```
@ExperimentalPermissionsApi
@Composable
fun RequestPermission(
    permission: String,
    rationaleMessage: String = "To use this app's functionalities, you need to give us the permission.",
) {
    val permissionState = rememberPermissionState(permission)
    HandleRequest(
        permissionState = permissionState,
        content = {
            PermissionDeniedContent(
                rationaleMessage = rationaleMessage
            ) { permissionState.launchPermissionRequest() }
        }
    )
}
@ExperimentalPermissionsApi
@Composable
fun HandleRequest(
    permissionState: PermissionState,
    content: @Composable () -> Unit
) {
    when (permissionState.status) {
        is PermissionStatus.Granted -> {
            ParkingApp()
        }
```

```
        is PermissionStatus.Denied -> {
            content()
        }
    }
}
@ExperimentalPermissionsApi
@Composable
fun PermissionDeniedContent(
    showButton: Boolean = true,
    rationaleMessage: String,
    onClick: () -> Unit
) {
    if (showButton) {
        val enableLocation = remember { mutableStateOf(true) }
        if (enableLocation.value) {
            AlertDialog(
                onDismissRequest = { enableLocation.value = false },
                title = {
                    Text(
                        text = "Permission Request",
                        style = TextStyle(
                            fontWeight = FontWeight.Bold
                        ),
                        fontSize = 18.sp
                    )
                },
                text = {
                    Text(rationaleMessage)
                },
                confirmButton = {
                    Button(
                        onClick = onClick, modifier = Modifier.padding(bottom = 10.dp, end=10.dp),
                    ) {
                        Text("Give Permission")
                    }
                }
            )
        }
    }
}
```

ParkingApp.kt

```
@Composable
fun ParkingApp() {
    val scrollState = rememberScrollState()
    val navController = rememberNavController()
    val navBackStackEntry by navController.currentBackStackEntryAsState()
    val currentRoute = navBackStackEntry?.destination?.route
    MainScreen(
        navController = navController,
        scrollState = scrollState,
        currentRoute = currentRoute
    )
}
@SuppressLint("UnusedMaterialScaffoldPaddingParameter")
@Composable
fun MainScreen(
    navController: NavHostController,
    scrollState: ScrollState,
    currentRoute: String?
) {
    val scaffoldState = rememberScaffoldState()
    val coroutineScope = rememberCoroutineScope()
    val openNewParkingDialog = remember { mutableStateOf(false) }
    val openAccelerometerDialog = remember { mutableStateOf(true) }
    val freePlacesNumber = remember { mutableStateOf(0) } //to pass the value to the server
    val bottomSheetScaffoldState = rememberBottomSheetScaffoldState()

    val userViewModel: UserViewModel = viewModel()
    val userList = userViewModel.usersListResponse
    userViewModel.getUserList()
    Scaffold(
        scaffoldState = scaffoldState,
        drawerContent = {
            DrawerMenu(
                scaffoldState = scaffoldState,
                scope = coroutineScope,
                navController = navController
            )
        },
        drawerGesturesEnabled = scaffoldState.drawerState.isOpen, //allows to close gestures only if the
drawerMenu is open
        bottomBar = {
            BottomMenu(
```

```
                navController = navController,
                scaffoldState = scaffoldState,
                scope = coroutineScope,
                bottomSheetScaffoldState = bottomSheetScaffoldState
            )
        },
        floatingActionButton = {
            if (currentRoute == BottomMenuData.AddNewPlace.route)
SubmitFab(openNewParkingDialog)
        })
    {
        if (openAccelerometerDialog.value) {
            ParkingDialog(openAccelerometerDialog)
        }
        Navigation(
            navController = navController,
            openNewParkingDialog,
            userList,
            coroutineScope,
            bottomSheetScaffoldState
        )
    }
}
@Composable
fun Navigation(
    navController: NavHostController,
    openNewParkingDialog: MutableState<Boolean>,
    userList: List<User>,
    coroutineScope: CoroutineScope,
    bottomSheetScaffoldState: BottomSheetScaffoldState
) {
    NavHost(navController = navController, startDestination = BottomMenuData.Explore.route) {
        composable(BottomMenuData.Explore.route) {
            ExploreScreen(navController, coroutineScope, bottomSheetScaffoldState)
        }
        println("Test3: $userList")
        bottomNavigation(
            openNewParkingDialog,
            userList,
            navController,
            coroutineScope,
            bottomSheetScaffoldState
        )
        drawerNavigation()
```

```
        commentsNavigation(navController)
    }
}
fun NavGraphBuilder.bottomNavigation(
    openNewParkingDialog: MutableState<Boolean>,
    userList: List<User>,
    navController: NavController,
    coroutineScope: CoroutineScope,
    bottomSheetScaffoldState: BottomSheetScaffoldState
) {
    composable(BottomMenuData.AddNewPlace.route) {
        AddNewPlaceScreen(openNewParkingDialog)
    }
    composable(BottomMenuData.FavouritePlaces.route) {
        println("Test4: $userList")
        FavouritesScreen(userList = userList)
    }
    composable(BottomMenuData.Explore.route) {
        ExploreScreen(navController, coroutineScope, bottomSheetScaffoldState)
    }
    composable(BottomMenuData.Menu.route) {
    }
}

fun NavGraphBuilder.drawerNavigation() {
    composable(DrawerMenuData.AccountSettings.route!!) {
    }
    composable(DrawerMenuData.GeneralSettings.route!!) {
    }
    composable(DrawerMenuData.History.route!!) {
    }
    composable(DrawerMenuData.InviteFriend.route!!) {
    }
    composable(DrawerMenuData.ReportIssue.route!!) {
    }
    composable(DrawerMenuData.About.route!!) {
        AboutScreen()
    }
    composable(DrawerMenuData.SignOut.route!!) {
    }
}
fun NavGraphBuilder.commentsNavigation(navController:NavController) {
    composable(MarkersData.Marker1.route) {
        ParkingAreaInfo(item = MarkersData.Marker1, navController)
```

```
    }
    composable(MarkersData.Marker2.route) {
        ParkingAreaInfo(item = MarkersData.Marker2, navController)
    }
    composable(MarkersData.Marker3.route) {
        ParkingAreaInfo(item = MarkersData.Marker3, navController)
    }
}
```

BottomSheet.kt

```
@Composable
fun BottomSheet(
    navController: NavController,
    coroutineScope: CoroutineScope,
    bottomSheetScaffoldState: BottomSheetScaffoldState,
    selectedMarker: MutableState<Marker?>
) {
    BottomSheetScaffold(
        scaffoldState = bottomSheetScaffoldState,
        sheetContent = {
            BottomSheetContent(
                marker = selectedMarker.value,
                navController = navController,
                scope = coroutineScope,
                bottomSheetScaffoldState = bottomSheetScaffoldState
            )
        },
        sheetPeekHeight = 0.dp,
        sheetShape = RoundedCornerShape(topStart = 16.dp, topEnd = 16.dp),
        modifier = Modifier.padding(bottom = 50.dp)
    ) {
        GoogleMapWithBottomSheet(selectedMarker, bottomSheetScaffoldState, coroutineScope)
    }
}

@Composable
fun BottomSheetContent(
    marker: Marker?,
    navController: NavController,
    scope: CoroutineScope,
    bottomSheetScaffoldState: BottomSheetScaffoldState
) {
    Column(
```

```
    modifier = Modifier
        .fillMaxWidth()
        .padding(top = 16.dp, end = 16.dp, start = 16.dp, bottom = 20.dp),
) {
    Row(
        Modifier.fillMaxWidth().padding(10.dp),
        horizontalArrangement = Arrangement.SpaceAround,
        verticalAlignment = Alignment.CenterVertically
    ) {
        Column(
            verticalArrangement = Arrangement.Center,
            horizontalAlignment = Alignment.CenterHorizontally
        ) {
            Text(
                textAlign = TextAlign.Center,
                text = "${marker?.title}",
                fontSize = 16.sp,
                fontWeight = FontWeight.Bold
            )
            Text(textAlign = TextAlign.Center, text = "loaded", fontSize = 16.sp)
        }
        IconButton(
            onClick = {}) {
            Column(
                verticalArrangement = Arrangement.Center,
                horizontalAlignment = Alignment.CenterHorizontally
            ) {
                Icon(
                    imageVector = if (true) Icons.Outlined.FavoriteBorder else Icons.Outlined.Favorite,
                    contentDescription = ""
                )
                Text(text = "Save")
            }
        }
        IconButton(
            onClick = {}) {
            Column(
                verticalArrangement = Arrangement.Center,
                horizontalAlignment = Alignment.CenterHorizontally
            ) {
                Icon(
                    imageVector = Icons.Outlined.Delete,
                    contentDescription = ""
                )
```

```
                Text(text = "Delete")
            }
        }
    }
    Row(Modifier.fillMaxWidth(), horizontalArrangement = Arrangement.Center) {
        Button(onClick = {
            navController.navigate("marker1")
            scope.launch {
                bottomSheetScaffoldState.bottomSheetState.collapse()
            }
        }) {
            Text(text = "Details")
        }
    }
  }
}
```

BottomMenu.kt

```
@Composable
fun BottomMenu(navController: NavController, scaffoldState: ScaffoldState, scope: CoroutineScope,
bottomSheetScaffoldState: BottomSheetScaffoldState) {

  val menuItems = listOf(
    BottomMenuData.AddNewPlace,
    BottomMenuData.FavouritePlaces,
    BottomMenuData.Explore,
    BottomMenuData.Menu
  )
  BottomNavigation(contentColor = colorResource(id = R.color.white))
  {
    val navBackStackEntry by navController.currentBackStackEntryAsState()
    val currentRoute = navBackStackEntry?.destination?.route
    menuItems.forEach {
      BottomNavigationItem(
        label = { Text(text = it.title) },
        alwaysShowLabel = true,
        selectedContentColor = Color.White,
        unselectedContentColor = Color.White.copy(0.7f),
        selected = currentRoute == it.route,
        onClick = {
          if (it.route == BottomMenuData.Menu.route)
            drawerOpen(scope, scaffoldState)
          else onNavigationItemClick(it, navController, scope, bottomSheetScaffoldState)
```

```
            },
            icon = {
                Icon(
                    imageVector = it.icon,
                    contentDescription = it.title
                )
            }
        )
    }
  }
}
fun drawerOpen(scope: CoroutineScope, scaffoldState: ScaffoldState) {
  scope.launch {
    scaffoldState.drawerState.open()
  }
}
fun onNavigationItemClick(
  it: BottomMenuData,
  navController: NavController,
  scope: CoroutineScope,
  bottomSheetScaffoldState: BottomSheetScaffoldState
) {
  navController.navigate(it.route) {
    navController.graph.startDestinationRoute?.let { route ->
      popUpTo(route) {
        saveState = true
      }
    }
    scope.launch {
      bottomSheetScaffoldState.bottomSheetState.collapse()
    }
    launchSingleTop = true
    restoreState = true
  }
}
```

ExploreScreen.kt

```
@Composable
fun ExploreScreen(
  navController: NavController,
  coroutineScope: CoroutineScope,
  bottomSheetScaffoldState: BottomSheetScaffoldState
) {
```

```
    val selectedMarker = remember { mutableStateOf<Marker?>(null) }
    BottomSheet(navController, coroutineScope, bottomSheetScaffoldState, selectedMarker)
}
@SuppressLint("MissingPermission")
@Composable
fun GoogleMapWithBottomSheet(
    selectedMarker: MutableState<Marker?>,
    bottomSheetScaffoldState: BottomSheetScaffoldState,
    scope: CoroutineScope
) {
    val mapView = rememberMapViewWithLifecycle()
    val kyiv = LatLng(50.44, 30.52)
    val markerMockList = listOf(
        MarkersData.Marker1,
        MarkersData.Marker2,
        MarkersData.Marker3
    )
    val polygonOptions = listOf(
        PolygonOptions()
            .add(LatLng(50.450565, 30.520874))
            .add(LatLng(50.449933, 30.520616))
            .add(LatLng(50.449892, 30.521277))
            .add(LatLng(50.450524, 30.521553))
            .strokeWidth(0.5f)
            .fillColor(0x7F00FF00),
        PolygonOptions()
            .add(LatLng(50.450789, 30.518947))
            .add(LatLng(50.450787, 30.519986))
            .add(LatLng(50.450195, 30.520004))
            .add(LatLng(50.450200, 30.518962))
            .strokeWidth(0.5f)
            .fillColor(0x7F0000FF)
    )
    AndroidView(
        factory = {
            mapView.apply {
                getMapAsync { map ->
                    val cameraPosition = CameraPosition.Builder()
                        .target(kyiv) //make on top (upper)
                        .zoom(14f)
                        .build()
                    map.moveCamera(CameraUpdateFactory.newCameraPosition(cameraPosition))
                    map.mapType = com.google.android.gms.maps.GoogleMap.MAP_TYPE_NORMAL
                    map.isMyLocationEnabled = true
```

```
map.isTrafficEnabled = true
map.isBuildingsEnabled = true
val centerLatLngList = mutableListOf<LatLng>()
polygonOptions.forEach { polygonOption ->
    map.addPolygon(polygonOption)
    centerLatLngList.add(getPolygonCenterPoint(polygonOption.points))
}
val uiSettings = map.uiSettings
uiSettings.isMyLocationButtonEnabled = true
uiSettings.isCompassEnabled = true
val markerOptions = MarkerOptions()
    .position(centerLatLngList[1])
    .title("8/8")
    .icon(defaultMarker(HUE_RED))
map.addMarker(markerOptions)

markerMockList.forEach {
    val markerOptions = MarkerOptions()
        .position(it.position)
        .title("${it.occupied}/${it.capacity}")
        .icon(
            if (it.occupied == it.capacity) defaultMarker(HUE_RED)
            else defaultMarker(HUE_GREEN)
        )
    map.addMarker(markerOptions)
}
map.setOnMarkerClickListener { clickedMarker ->
    val values: List<String> =
        selectedMarker.value?.let { it.title?.split("/") } ?: emptyList()
    if (values.isNotEmpty()) {
        selectedMarker.value?.setIcon(
            if (values[0] == values[1]) defaultMarker(HUE_RED) else defaultMarker(
                HUE_GREEN
            )
        )
    }
    clickedMarker.setIcon(
        defaultMarker(HUE_ORANGE)
    )
    selectedMarker.value = clickedMarker
    scope.launch {
        bottomSheetScaffoldState.bottomSheetState.expand()
    }
    true
```

```
                }
            }
        }
    },
    modifier = Modifier.fillMaxSize()
  )
}
fun getPolygonCenterPoint(points: List<LatLng>): LatLng {
  var latSum = 0.0
  var lngSum = 0.0
  for (point in points) {
    latSum += point.latitude
    lngSum += point.longitude
  }
  val latCenter = latSum / points.size
  val lngCenter = lngSum / points.size
  return LatLng(latCenter, lngCenter)
}
```

ДОДАТОК Б

Інструментальні засоби краудсорсингу в транспортних системах

Публікації
XX міжнародна науково-практична конференція молодих вчених та студентів

Аркушів 3

Київ — 2023

УДК 004.78

Студент 4 курсу, гр. ТІ-92 Тютюнник О.Г.
Доц., к.е.н. Гусєва І.І.

ІНСТРУМЕНТАЛЬНІ ЗАСОБИ КРАУДСОРСИНГУ В ТРАНСПОРТНИХ СИСТЕМАХ

Із зростанням індустріалізації, зростає також кількість машин у містах, через що збільшується забруднення і зменшується кількість вільних місць для паркування автомобіля. З'являється питання оптимізації часу, проведеного у пошуках місця для зупинки. Однак через велику кількість різних типів місць для паркування і їх розташування, з'являється проблема у обробці усіх цих даних і їх збереження в одному місці, а також підтримки їх актуальності в режимі реального часу. Через такі причини і з'являються інструменти для полегшення та пришвидшення пошуку вільного місця для свого транспортного засобу.

Актуальність таких додатків полягає у тому, що вони здатні вирішити або, хоча б частково, покращити проблему з паркуваннями, оптимізувавши час їх пошуку і, як наслідок, це зменшить виділення шкідливих речовин від транспорту в повітря, а також заощадить енергію електричного автомобіля для наступних поїздок.

Такі засоби можуть допомогти державі проаналізувати у яких районах потрібно побудувати додаткові місця для паркування, а де, навпаки, їх зменшити. Це допоможе зробити місто більш зручним для водіїв та корисним для мешканців.

Сучасні технології дозволяють робити такі інструментальні засоби, використовуючи різні методи: сканування камер з паркувальних майданчиків [1], встановлення сенсорів [2] на кожне парко місце безпосередньо, сканування місцевості лазерами, використання краудсорсингу [3] та інші.

Розглядаючи останній метод, краудсорсинг – це залучення людських можливостей для спільного вирішення певних проблем чи втілення проєктів. Тобто програма надаватиме дозвіл користувачам покращувати додаток та ділитися даними (інформацією) між собою спільними силами. Такий метод використовується у багатьох сферах для отримання більшої кількості інформації від користувачів та, таким чином, покращує результати аналізу цих даних. Тобто це допоможе транслювати у застосунку більш актуальну інформацію, застосовуючи знання та дані від самих же користувачів.

Якщо об'єднати згадану вище проблему та метод краудсорсингу, то можна отримати один з можливих варіантів часткового покращення ситуації з пошуком місця для паркування. Найкраще для цього підійде розробка саме мобільного додатку, адже це дозволить водіям використовувати його в будь який час.

Після перегляду та тестування деяких аналогів [4] інших країн, було виділено низку покращень інтерфейсу для майбутньої програми, що полегшить взаємодію з користувачами. Також потрібно не забувати про перевірку зовнішнього вигляду (UI частини) програми на коректність відображення вмісту сторінок (іконки, текст, поля для вводу) на різних розмірах екрану, різних оболонках, оперативних системах та версіях прошивки.

Буде використано вже існуючу базу з місцями для паркування, що надається державою. Аналіз можна буде проводити не лише на основі цієї бази, але й вручну додати свої парко місця, яких ще немає на мапі додатку. У цій частині програми буде застосовано краудсорсинг – нове парко місце, яке було запропоноване користувачами, буде додано на загальну карту та в базу даних з парко місцями тільки після певної кількості однакових заяв. Також буде використовуватись безпосередня інформація користувача з телефону (дані з акселерометра [5], gps) для пошуку вчасної миті, щоб взаємодіяти з водієм. Тобто коли з даних акселерометра буде видно, що користувач зупиняється, то програма зможе

запитатись про вільні місця поряд з цим користувачем на паркувальному майданчику. Таким чином дані будуть оновлені до актуальних і це допоможе іншим водіям, які ще в процесі пошуку вільного місця і знаходяться поруч, швидко зайняти місце і оновити інформацію, що воно вже зайнято для наступних користувачів.

Також важливо правильно розписати інформацію, яку можна додавати користувачеві при заповненні запиту на додавання нового парко місця на мапу, а зі сторони розробників важливо її ретельно провалідувати при отриманні, щоб це не зламало систему.

Наприклад користувач зможе додавати інформацію про довжину та ширину місця, точну адресу та певні деталі для швидшої ідентифікації місця водієм під час поїздки. Можна буде додати інформацію про тариф, способи оплати, певні особливості та інше. Також у користувача буде можливість додати парко місця до списку своїх обраних, що пришвидшить навігацію.

Програмний продукт буде реалізований за допомогою Kotlin mobile [6], матиме декілька баз даних різних видів (SQL – Firebase із даними про користувачів та бази місць для паркування, тимчасова база даних із запитами на додавання нового парко місця на мапу, також у якійсь структурі даних будуть зберігатися проміжні дані з акселерометра та gps), інтерфейс буде створений за допомогою елементів, що надаються у Kotlin mobile та записані у XML-файлах. Будуть використовуватися карти від Google та Amazon, на яких будуть відображатися елементи додатку (іконки про паркувальні майданчики, де знаходиться користувач та інше).

Мобільний додаток допоможе збирати інформацію з декількох джерел одразу в одному місці:

- від держави;
- від самих користувачів на основі краудсорсингу
- від водіїв, що припаркувались про можливі вільні місця біля них для отримання максимально актуальної інформації;
- від датчиків з телефону про місцезнаходження для більш швидкого отримання інформації.

Дана програмна система суттєво зменшить час пошуку місця для паркування, збереже нерви водіям, а також планету від забруднень. Завдяки додатку отримано можливість швидшого визначення вільних місць для паркування, а також система надає підґрунтя для подальшого розвитку таких систем на основі краудсорсингу в Україні.

Перелік посилань:

1. Deshun, H.; Erkang, C. Automatic Parking Space Detection Method Based on Deep Learning, .

2. Wu, Q. and Zhang, Y. Parking Lots Space Detection, Machine Learning, Carnegie Mellon University, 2006.

3. R. Arnott and J. Rowse, “Modeling parking”, Journalof Urban Economics, vol. 45(1), pp. 97–124, 1999.

4. A. L. C. de Cerreo, “The dynamics of on-street park-ing in large central cities”, Transportation ResearchRecord, vol. 1898, pp. 130–137, 2004.

5. Аврутов В.В., Бондар П.М., Мелешко В.В. Мікроакселерометри ти їх випробування: Навчальний посібник, 2008.

6. Мова Kotlin для Розробки програмного забезпечення мобільних пристроїв: Недашківський О. Л.. - Київ.: КПІ ім. Ігоря Сікорського. 2022.