Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра системного аналізу та інформаційних технологій

Бакалаврська дипломна робота на тему:

«АНАЛІЗ ТА ВІЗУАЛІАЦІЯ ДАНИХ МОНІТОРИНГОВИХ IOT-СИСТЕМ НА БАЗІ МОБІЛЬНИХ ANDROID-ПРИСТРОЇВ»

Виконав: студент 4 курсу, групи СА-20б
спеціальності 124 – «Системний аналіз»
_____ Сергій ДЖУРА
Керівник: к.т.н., доц. каф. САІТ
_____ Олексій КОЗАЧКО
«03» 06 2024 р.
Рецензент: к.т.н., доц. каф. КН
_____ Олексій СІЛАГІН
«11» 06 2024 р.

Допущено до захисту

Завідувач кафедри САІТ
_____ д.т.н., проф. Віталій МОКІН
«04» 06 2024 р.

Вінниця ВНТУ – 2024 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра системного аналізу та інформаційних технологій
Рівень вищої освіти – перший (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність 124 – «Системний аналіз»
Освітньо-професійна програма – Системний аналіз

ЗАТВЕРДЖУЮ
Завідувач кафедри САІТ
д.т.н., проф. Віталій МОКІН
"05" 03 2024 року

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ
Джурі Сергію Вікторовичу

1. Тема роботи: «Аналіз та візуалізація даних моніторингових IoT–систем на базі мобільних Android–пристроїв»
керівник роботи: Козачко О. М., к.т.н., доц. каф. САІТ
затверджені наказом ВНТУ від «11» 03 2024 року №80
2. Термін подання студентом роботи 31.05
3. Вихідні дані до роботи:
1) Дані вимірювань CO_2 за допомогою IoT-систем.
4. Зміст текстової частини:
1) Загальна характеристика об'єкту досліджень;
2) Огляд використаних технологій;
3) Результати досліджень, аналіз і візуалізація даних;
5. Перелік ілюстративного матеріалу:
1) Аналоги IoT;
2) Хмарні сховища IoT;
3) UML діаграми;
4) Макет застосунку;
5) Загальний вигляд Android застосунку.

6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
	Горячев Г. В. доцент, к.т.н	11.03	31.05

7. Дата видачі завдання 11.03

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Аналіз предметної області	11.03	31.03	Вик
2	Загальна характеристика об'єкту досліджень	01.04	15.04	Вик
3	Вибір оптимальних інформаційних технологій	16.04	20.04	Вик
4	Результати дослідження та аналіз і візуалізація даних	01.05	15.05	Вик
5	Оформлення матеріалів до захисту БДР	16.05	31.05	Вик

Студент _____ Сергій ДЖУРА

Керівник роботи _____ Олексій КОЗАЧКО

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 83 сторінок формату А4, на яких є 24 рисунка, список використаних джерел містить 20 найменувань.

Метою роботи є розроблення андроїд застосунку для аналізу та візуалізації даних моніторингових loT-систем на базі мобільних Android-пристроїв.

В роботі наведено розроблення мобільного додатку на платформі Android Studio. Розроблено дизайн та функціонал мобільного додатку. Розроблений застосунок дає можливість моніторингу даних за вибраний користувачем період також додаток дозволяє відображати дані у вигляді діаграм ще є карта яка дозволяє користувачеві бачити локацію звідки ми отримуємо дані з датчика.

Ключові слова: інтернет речей, мобільний додаток, Android Studio, Kotlin, loT.

ABSTRACT

The bachelor's thesis consists of 83 A4 pages, which contain 24 figures and a list of references containing 20 titles.

The purpose of the work is to develop an android application for analyzing and visualizing data from monitoring loT systems based on mobile Android devices.

The paper presents the development of a mobile application on the Android Studio platform. The design and functionality of the mobile application are developed. The developed application makes it possible to monitor data for a period selected by the user, and the application also allows you to display data in the form of diagrams, and there is a map that allows the user to see the location from which we receive data from the sensor.

Keywords: Internet of Things, mobile application, Android Studio, Kotlin, loT.

ЗМІСТ

ВСТУП

Актуальність теми. У сучасному світі зростає кількість пристроїв, що підключаються до Інтернету речей (IoT). Ці пристрої генерують велику кількість даних, що потребують ефективного збору, аналізу та візуалізації для прийняття інформованих рішень. Системи моніторингу IoT використовуються у різних сферах, включаючи розумні будинки, промисловість, сільське господарство та охорону здоров'я. Мобільні пристрої на базі Android стають все більш популярними завдяки своїй доступності та функціональним можливостям. Актуальність цього дослідження полягає в необхідності створення ефективних інструментів для аналізу та візуалізації даних з IoT-систем, що дозволяють користувачам отримувати зрозумілі та корисні висновки з великих обсягів інформації.

Мета і задачі дослідження. Метою дослідження є покращення мобільних засобів аналізу та візуалізації даних вимірювань отриманих із пристроїв loT.

Для досягнення поставленої мети необхідно розв’язати такі задачі:

- провести огляд існуючих рішень для аналізу та візуалізації даних отриманих з loT-систем;
- здійснити та обґрунтувати вибір фреймворків і технологій для розробки мобільного додатку ;
- здійснити реалізацію клієнтського додатку на платформі Android для обробки та візуалізації даних з loT- пристроїв.

Об’єктом дослідження є процес аналізу та візуалізації даних вимірювань пристроїв loT.

Предметом дослідження є методи та інструменти для аналізу та візуалізації даних, отриманих з моніторингових IoT-систем на базі мобільних Android-пристроїв. Це включає в себе розробку та впровадження алгоритмів для обробки даних, створення інтуїтивно зрозумілих графічних інтерфейсів для відображення результатів, а також інтеграцію з існуючими IoT-платформами та сервісами для забезпечення безперебійної роботи системи.

Публікації. За результатами даної роботи була зроблена доповідь на тему «Аналіз та візуалізація даних моніторингових IoT-систем на базі мобільних Android-пристроїв» на Міжнародній науково-методичній інтернет-конференції «Проблеми вищої математичної освіти: виклики сучасності (Вінниця, 2024р)», де були опубліковані відповідні тези [1].

1 ЗАГАЛЬНА ХАРАКТЕРИСТИКА ОБЄ'КТУ ДОСЛІДЖЕНЬ

1.1 Опис об'єкта досліджень

Об'єктом дослідження є моніторингові системи Інтернету речей (IoT) на базі мобільних Android-пристроїв, які збирають та передають різноманітні дані з датчиків і сенсорів у реальному часі. Цей додаток має універсальні можливості для аналізу часових рядів даних з різноманітних джерел, таких як датчики вимірювання показників якості навколишнього середовища. Ці дані можуть бути використані для виявлення закономірностей та прогнозування [2]. Такі системи широко використовуються у різних галузях, таких як сільське господарство, охорона здоров'я, промисловість, моніторинг навколишнього середовища тощо. Для ефективного аналізу та візуалізації зібраних даних на мобільних пристроях під Android використовуються відповідні інструменти та бібліотеки [3]. Одним із середовищ для розробки є Android Studio, а основними бібліотеками для обробки та представлення даних є:

1. MPAndroidChart: бібліотека для створення різних видів графіків і діаграм;
2. GraphView: бібліотека для створення інтерактивних графіків;
3. AChartEngine: ще одна популярна бібліотека для створення графіків на Android;
4. TensorFlow Lite: версія TensorFlow для мобільних пристроїв, яка дозволяє здійснювати складний аналіз даних і машинне навчання;
5. Room: бібліотека для зберігання та управління даними в локальній базі даних.

Основними завданнями дослідження є:

1. Аналіз даних, що збираються IoT-система;
2. Розробка додатку для Android, який отримує дані з серверної частини додатку;
3. Розробка інтерактивного меню для більш зручного користування додатків для представлення візуалізованих даних користувачам у зручному форматі.

На рисунку 1.1 Зображена структура інформаційної системи моніторингу фізичних показників на основі Інтернету речей.

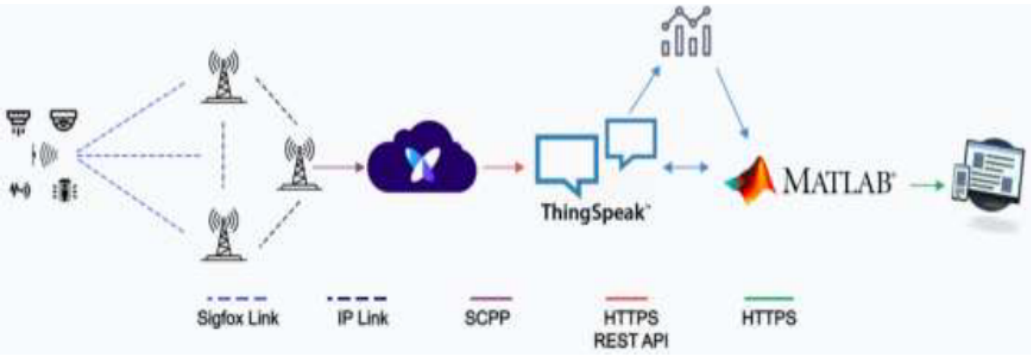

Рисунок 1.1 – Структура інформаційної системи моніторингу фізичних показників на основі Інтернету речей

Аналіз та інтерпретація візуалізованих даних для виявлення аномалій, n закономірностей та прийняття рішень щодо оптимізації роботи IoT-систем та підвищення ефективності моніторингу.

Для проведення дослідження використовуються реальні або синтетичні дані, отримані з IoT-пристроїв та датчиків. Аналіз даних виконується з використанням методів статистичного аналізу, машинного навчання та візуалізації даних [2].

Значна увага приділяється оптимізації процесів збору, передачі та обробки даних, а також забезпеченню безпеки та конфіденційності даних.

Системний аналіз та візуалізація даних моніторингових IoT-систем на базі мобільних Android-пристроїв дозволяє отримувати цінну інформацію про стан різних об’єктів та процесів у реальному часі, що сприяє прийняттю обґрунтованих рішень, підвищенню ефективності та оптимізації процесів.

1.2 Аналіз аналогічних методів та інформаційних технологій

Інтернет речей (IoT) став одним з найважливіших напрямів розвитку сучасних інформаційних технологій. Він дозволяє з'єднувати фізичні пристрої, забезпечуючи їх взаємодію та обмін даними через Інтернет. Це призводить до

створення інтелектуальних систем, здатних автоматично збирати, аналізувати та обробляти дані для прийняття рішень. У цьому розділі буде розглянуто аналогічні методи та програмні технології, що використовуються в IoT-системах. Нижче наведено аналоги додатків які використовуються в системах- loT [8] .

MiHome – це мобільний додаток, розроблений компанією Xiaomi, який дозволяє користувачам керувати та контролювати свої розумні пристрої з єдиного інтерфейсу. Основні функції інтерфейсу управління, пристроями, підключення розумних пристроїв від Xiaomi. Інтеграція з IoT: Синхронізація з іншими пристроями Інтернету речей (IoT), що підтримуються, для створення розумної екосистеми вдома [3]. На рисунку 1.2 зображено головний екран застосунку MiHome.

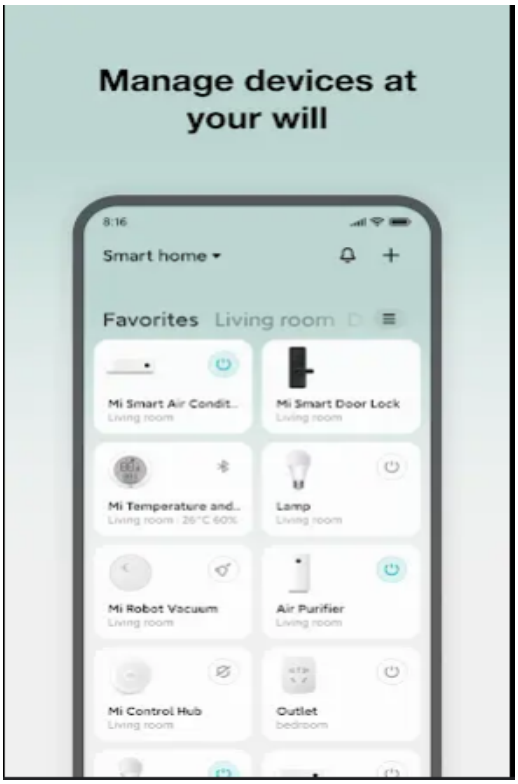

Рисунок 1.2 – Головний екран додатку MiHome

SmartThings–це платформа та екосистема "розумного" будинку, розроблена компанією Samsung. Вона активно використовує технології Інтернету речей (IoT) для інтеграції та керування різноманітними пристроями в домашніх умовах. Основний функціонал. Центральний хаб для підключення loT-

пристроїв SmartThings дозволяє підключати і керувати великою кількістю сумісних пристроїв різних виробників, таких як розумні лампи, розетки, датчики, камери безпеки, термостати та інші. Мобільний додаток та віддалене керування: Додаток SmartThings для Android та iOS дозволяє моніторити стан підключених пристроїв та керувати ними віддалено з будь-якого місця. На рисунку 1.3 зображено головний екран додатку SmartThings [4].

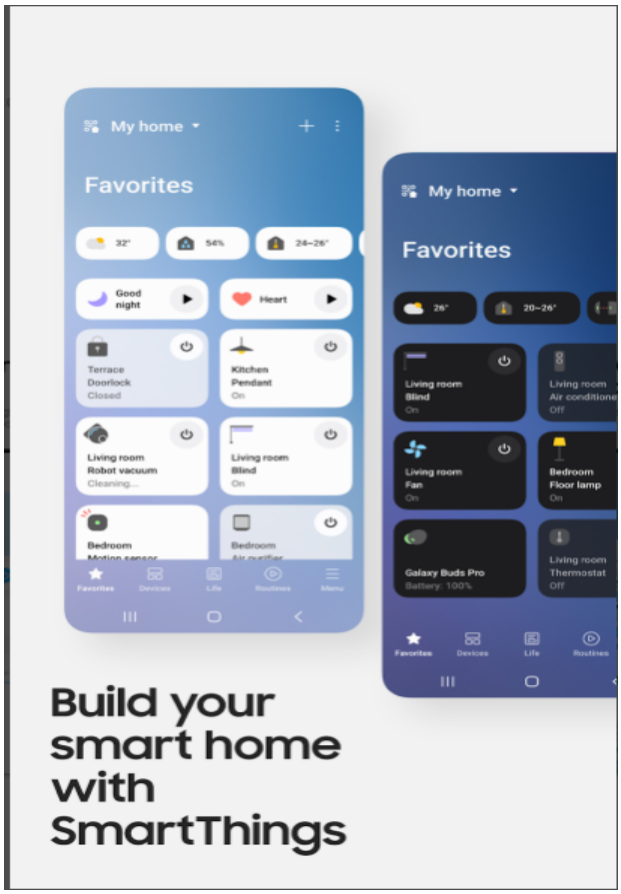

Рисунок 1.3 – Головний екран додатку SmartThings

Home Assistant — це безкоштовне програмне забезпечення з відкритим вихідним кодом для домашньої автоматизації, створене як незалежна від екосистеми інтеграційна платформа Інтернету речей (IoT) і розумний домашній центр для керування пристроями розумного дому з акцентом на локальному контролі та конфіденційності. Доступ до його інтерфейсу можна отримати через веб- інтерфейс користувача, за допомогою супутніх програм для Android та iOS або за допомогою голосових команд через підтримуваного віртуального помічника, наприклад Google Assistant , Amazon Alexa , Apple Siri та власний

«Assist» Home Assistant (вбудований місцевий голосовий помічник) з використанням природної мови. На рисунку 1.4 зображено головний екран додатку HomeAssistant [5].

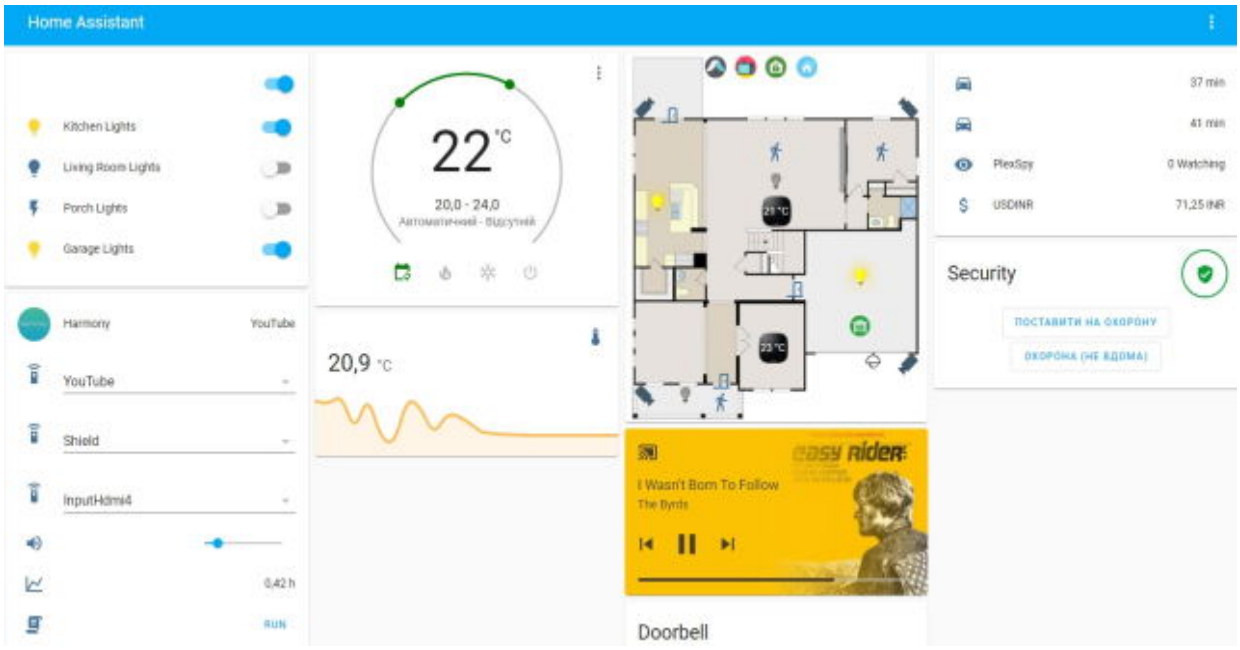

Рисунок 1.4 – Головний екран додатку Home Assistant

loT ThingSpeak Monitor Widget — це додаток для Android-пристроїв, який дозволяє візуалізувати та відстежувати дані, отримані від пристроїв та датчиків в Інтернеті речей (IoT). Він інтегрується з платформою ThingSpeak, яка є аналітичним інструментом для збору, візуалізації та аналізу потоків даних в реальному часі від пристроїв IoT. На рисунку 1.5 зображений інтерфейс додатку.

Основні функції додатку IoT ThingSpeak Monitor Widget:

1. Візуалізація даних: Додаток дозволяє відображати дані, отримані від IoT-пристроїв та датчиків, у вигляді графіків, діаграм та інших візуальних елементів. Це полегшує інтерпретацію та аналіз даних;
2. Віджити на головному екрані: Користувачі можуть розміщувати віджети прямо на головному екрані свого Android-пристрою, що забезпечує зручний доступ до найважливіших даних без необхідності відкривати сам додаток;

3. Моніторинг у реальному часі: Додаток дозволяє відстежувати потоки даних від IoT-пристроїв у режимі реального часу, що є корисним для моніторингу критично важливих процесів або виявлення аномалій;
4. Підтримка декількох каналів та пристроїв: Користувачі можуть підключатися до різних каналів даних на платформі ThingSpeak та відстежувати дані з різноманітних IoT-пристроїв та датчиків;
5. Налаштування сповіщень: Додаток пропонує функцію налаштування сповіщень, які можуть повідомляти користувачів про певні події або умови, пов'язані з отриманими даними;
6. Інтеграція з ThingSpeak: IoT ThingSpeak Monitor Widget тісно інтегрований з платформою ThingSpeak, що забезпечує безперебійний обмін даними та можливість використання додаткових функцій, таких як аналіз даних та створення візуалізацій на веб-платформі [6].

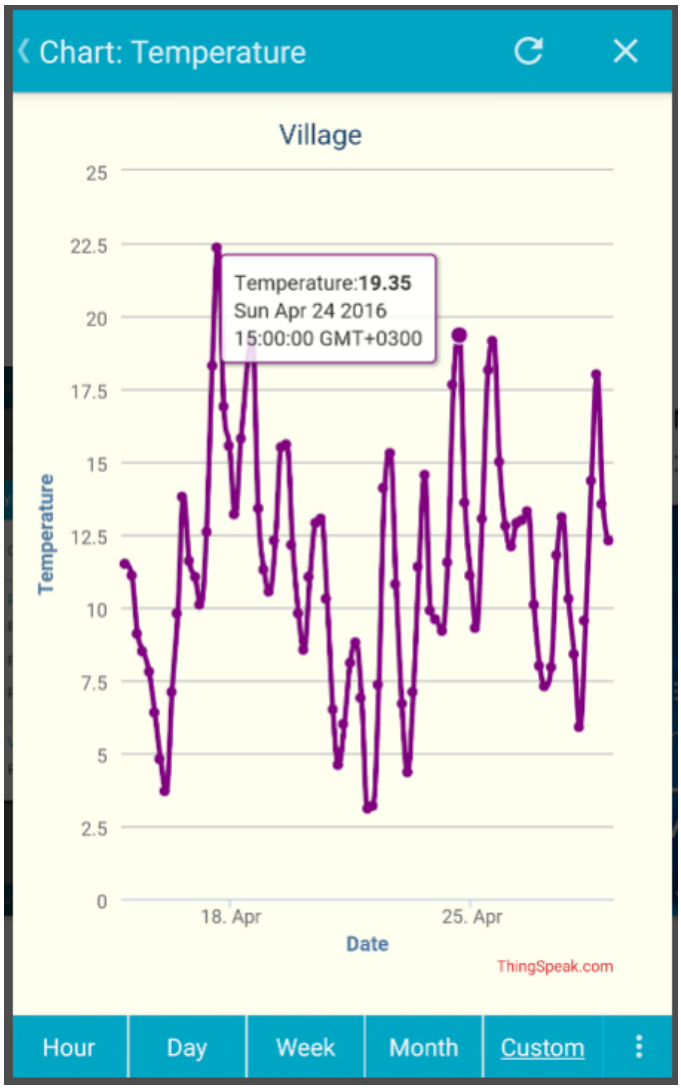

Рисунок 1.5 –Інтерфейс додатку IoT ThingSpeak Monitor Widget

Головною відмінністю IoT ThingSpeak Monitor Widget від мого додатка є відсутність функціоналу для можливості аналізу просторових даних. Мій додаток забезпечує більш розширену аналітику, включаючи аналіз та візуалізацію просторових даних, що значно підвищує корисність додатка для користувачів, яким потрібно не лише відслідковувати дані, але й аналізувати їх у географічному контексті. Крім того, аналітичні можливості IoT ThingSpeak Monitor Widget є досить простими порівняно з моїм додатком, що пропонує більш просунуті засоби аналізу та візуалізації.

ThingView - ThingSpeak viewer — це додаток для Android-пристроїв, який виступає у ролі переглядача для платформи ThingSpeak, яка є аналітичним інструментом для збору, візуалізації та аналізу даних, отриманих від пристроїв Інтернету речей (IoT). На рисунку 1.6 зображений інтерфейс додатку.

Основні функції додатку ThingView - ThingSpeak viewer:

1. Перегляд і моніторинг каналів даних ThingSpeak: Додаток дозволяє переглядати дані з публічних та приватних каналів ThingSpeak у вигляді графіків, діаграм та візуалізацій у реальному часі.
2. Підтримка різних типів візуалізацій: ThingView підтримує різноманітні типи візуалізацій даних, такі як лінійні графіки, гістограми, діаграми розсіювання, індикатори, картографічні візуалізації тощо.
3. Налаштування відображення даних: Користувачі можуть налаштовувати інтервал часу, масштаб, колірну гаму та інші параметри відображення даних відповідно до своїх потреб.
4. Експорт даних: Додаток дозволяє експортувати дані з каналів ThingSpeak у різних форматах, таких як CSV, JSON або XML, для подальшого аналізу або інтеграції з іншими системами.
5. Пошук та відстеження каналів: Користувачі можуть шукати канали ThingSpeak за ключовими словами, тегами або іншими критеріями, а також додавати канали до списку відстеження для зручного доступу.

6. Сповіщення: ThingView може надсилати сповіщення на Android-пристрій, коли дані в каналі ThingSpeak перевищують певні порогові значення або відбуваються певні події.
7. Підтримка автентифікації: Додаток підтримує автентифікацію для доступу до приватних каналів ThingSpeak за допомогою облікових записів ThingSpeak або через інтеграцію з провайдерами автентифікації, такими як Google, Facebook тощо [6].

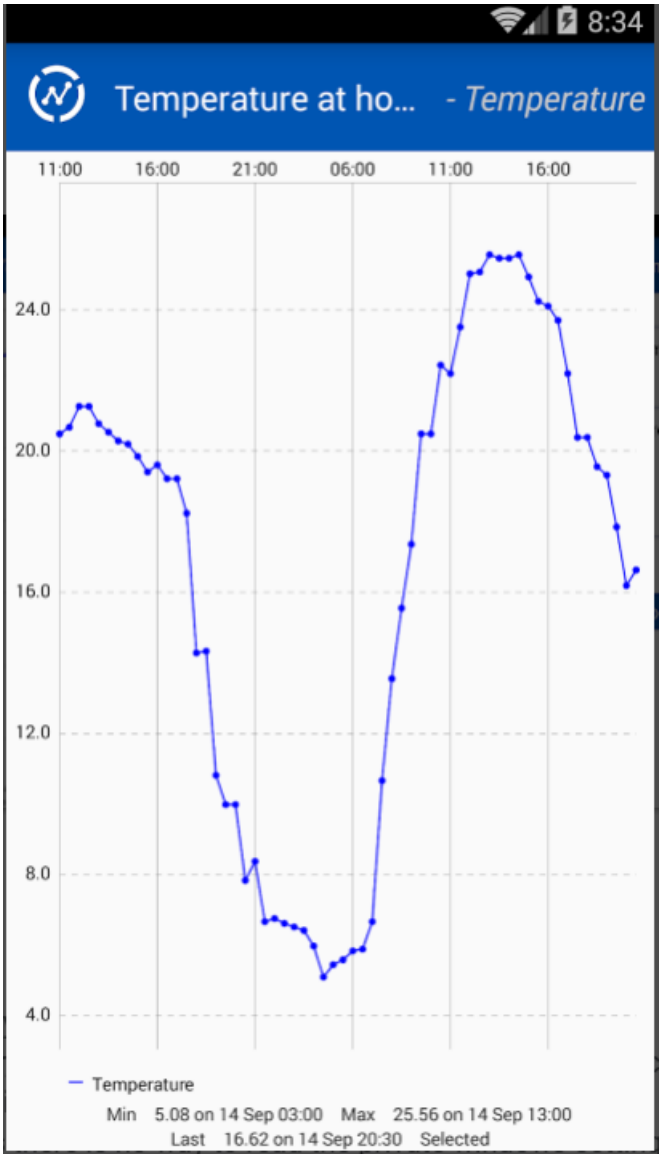

Рисунок 1.6 – Інтерфейс додатку ThingView - ThingSpeak viewer

Головною відмінністю ThingView - ThingSpeak viewer від мого додатку є відсутність функціоналу для можливості аналізу просторових даних. Мій додаток забезпечує більш розширену аналітику, включаючи аналіз та

візуалізацію просторових даних, що значно підвищує корисність додатка для користувачів, яким потрібно не лише відслідковувати дані, але й аналізувати їх у географічному контексті. Крім того, аналітичні можливості IoT ThingSpeak Monitor Widget є досить простими порівняно з моїм додатком, що пропонує більш просунуті засоби аналізу та візуалізації.

1.3 Переваги та недоліки впровадження IoT-систем на Android-пристроях

Переваги:

1. Широке розповсюдження Android-пристроїв Android є найпоширенішою мобільною операційною системою у світі, що забезпечує велику кількість потенційних користувачів для IoT-рішень. Це відкриває можливості для широкого застосування IoT-технологій у повсякденному житті, дозволяючи підключати до мережі численні пристрої, від смартфонів до побутової техніки та автомобілів.

2. Відкрита платформа Android є відкритою платформою, що дозволяє розробникам створювати та модифікувати застосунки без обмежень. Це сприяє інноваціям, даючи можливість експериментувати з новими функціями та технологіями. Розробники мають повний доступ до коду та інструментів, що дозволяє їм швидко реагувати на потреби ринку та вдосконалювати свої продукти.

3. Велика кількість доступних бібліотек та SDK Android надає розробникам широкий спектр бібліотек та SDK для роботи з різноманітними сенсорами, мережевими протоколами та інтерфейсами. Це значно полегшує розробку IoT-рішень, дозволяючи швидко інтегрувати необхідні функції та забезпечити високу якість продукту.

4. Гнучкість та налаштованість Android дозволяє розробникам гнучко налаштовувати інтерфейс та функціональність застосунків під конкретні потреби користувачів. Це особливо важливо для IoT-рішень, які часто потребують індивідуального підходу та адаптації до специфічних вимог користувачів та пристроїв [7].

5. Інтеграція з різноманітними сервісами Android-пристрої можуть легко інтегруватися з іншими сервісами та платформами, такими як Google Cloud, Firebase, Amazon AWS тощо. Це дозволяє створювати комплексні IoT-рішення з використанням хмарних технологій, забезпечуючи високу надійність та масштабованість систем.

6. Ефективне управління енергоспоживанням Сучасні Android-пристрої оснащені технологіями для ефективного управління енергоспоживанням, що є важливим для IoT-пристроїв, які часто працюють на батарейках. Це дозволяє забезпечити тривалу роботу пристроїв без необхідності частої заміни або підзарядки батарей.

Недоліки:

1. Фрагментація пристроїв та версій ОС Android має багато різних версій і моделей пристроїв, що ускладнює розробку універсальних рішень. Розробникам потрібно враховувати сумісність з різними версіями ОС та пристроями, що може призвести до додаткових витрат часу та ресурсів на тестування та оптимізацію.

2. Безпека та конфіденційність даних Безпека є критичним аспектом для IoT-систем. Android-пристрої можуть бути вразливими до різних видів атак, тому розробникам необхідно впроваджувати надійні механізми захисту даних та забезпечувати конфіденційність користувачів. Це включає шифрування даних, автентифікацію користувачів та захист від несанкціонованого доступу.

3. Обмежені ресурси пристроїв Багато IoT-пристроїв мають обмежені ресурси, такі як обсяг пам’яті, процесорна потужність та енергоспоживання. Це вимагає оптимізації застосунків для ефективної роботи в умовах обмежених ресурсів, що може бути складним завданням для розробників.

4. Складність налаштування та інтеграції Інтеграція IoT-систем може бути складною через необхідність налаштування різноманітних сенсорів, мережевих протоколів та інших компонентів. Це може вимагати значних технічних знань та навичок, що підвищує бар'єр для входу на ринок для нових гравців [7].

5. Вимоги до постійного підключення до Інтернету Багато IoT-рішень залежать від постійного підключення до Інтернету для обміну даними. Відсутність стабільного підключення може призвести до проблем у функціонуванні системи, що особливо важливо в умовах, де Інтернет-з'єднання може бути нестабільним або відсутнім.

6. Витрати на розробку та обслуговування Розробка та обслуговування IoT-рішень може бути дорогою, особливо якщо враховувати витрати на апаратне забезпечення, програмне забезпечення та хмарні сервіси. Це вимагає значних інвестицій на початкових етапах розробки та постійних витрат на підтримку та оновлення системи.

7. Забезпечення сумісності з різними стандартами Існує багато стандартів та протоколів для IoT, що може ускладнити забезпечення сумісності між різними пристроями та системами. Розробникам потрібно враховувати ці стандарти та забезпечувати їх підтримку у своїх продуктах, що може вимагати додаткових зусиль та ресурсів.

8. Моніторинг та управління Ефективний моніторинг та управління IoT-пристроями може бути складним завданням, особливо у випадку великої кількості пристроїв, розташованих у різних місцях. Це вимагає надійних інструментів та методів для віддаленого моніторингу, управління та оновлення пристроїв [7].

1.4 Висновки

У результаті проведеного дослідження, було визначено основні переваги та недоліки впровадження IoT-системи на Android-пристроях, проаналізував існуючі методи та інформаційні технології, а також надав опис об'єкта мого дослідження.

Проаналізувавши існуючі методи та інформаційні технології, було виявлено найбільш ефективні підходи до збирання, оброблення та візуалізації даних, а також розглянуто різні протоколи для зв'язку між пристроями. Це

дозволило вибрати оптимальні інструменти та технології для розробки додатків, що забезпечують надійну та ефективну роботу IoT-систем.

2 ОГЛЯД ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

2.1 Вибір фреймворків для розробки

У таблиці 2.1 проведено порівняння різних фреймворків для кросплатформної розробки ця таблиця допомагає розробникам мобільних додатків зробити обґрунтований вибір технологічного фреймворку для свого проекту. Таблиця порівнює популярні нативні та крос-платформні рішення, такі як Swift та Kotlin для нативної розробки, React Native, Cordova, Flutter, Ionic та Xamarin для кросплатформної розробки. Порівняння охоплює ключові аспекти, включаючи цільові платформи, мови програмування, продуктивність, користувацький інтерфейс, зручність тестування та розробки, витрати та час виходу на ринок, а також підтримку спільноти [8].

Таблиця 2.1 – Порівняння різних фреймворків

	Native IOS	Native Android	React Native	Cordova	Fluter	lonic	Xamarin
Can build apps for	Apple only	Android only	iOS, Android, Web	iOS, Android, Web	iOS, Android, Web	iOS, Android, Web	iOS, Android, Web
Language	Swift	Kotlin	JavaScript	JavaScript	Dart	JavaScript	C#
Performance	Very high	Very high	High	Low	High	Low	Aver
UX UI	Native	Native	Adapts to platforms	Adapts to platform	Adapts to platform	Adapts to platform	Adapts only partially
Testing development	Convenient and seamless	Convenient and seamless	Average	Fuster supports Live Reload	Fuster supports Live Reload	Average	Fuster supports Live Reload
Cost and time to market	More expensive and slower	More expansive and slower	Cheaper and faster	Cheaper and faster	Cheaper and faster	Cheaper and faster	Cheaper and faster
Community support	Very popular	Very popular	Very popular	Not so popular	Popular	Not so popular	Not so popular

Android забезпечує дуже високу продуктивність, оскільки додатки є нативними та компільованими безпосередньо для Android. Це перевершує більшість крос-платформних фреймворків, таких як Cordova, Ionic та Xamarin, які мають нижчу продуктивність.

Користувацький інтерфейс (UI/UX): Додатки Android мають повністю нативний користувацький інтерфейс, сумісний з керівними принципами Material Design від Google. Це забезпечує максимально природний досвід для користувачів Android.

Зручність розробки згідно з таблицею, Android має зручне та безпроблемне тестування та розробку, що робить процес створення додатків більш ефективним.

Популярність та підтримка спільноти: Android є однією з найпопулярніших і найбільш підтримуваних спільнотою мобільних платформ, що забезпечує багатий вибір ресурсів, бібліотек та підтримки від спільноти розробників.

Хоча крос-платформні фреймворки, такі як React Native та Flutter, також мають високу продуктивність та певні переваги, вони не можуть повністю конкурувати з нативною розробкою на Android з точки зору продуктивності, користувацького інтерфейсу та сумісності з вказівками платформи.

Я вибрав Android Studio для розробки свого додатку Android через його переваги у продуктивності, нативному користувацькому інтерфейсі, зручності розробки, а також величезної популярності та підтримки спільноти розробників Android. Якщо основною метою є створення високопродуктивних додатків з оптимальним користувацьким досвідом на платформі Android, то нативна розробка з Android Studio є найкращим вибором порівняно з іншими фреймворками, представленими в таблиці [9].

2.2 Інтеграція з хмарними сервісами для loT

AWS IoT Core (Amazon Web Services): Масштабована хмарна платформа для підключення та управління IoT-пристроями. Підтримує різні протоколи

передачі даних, такі як MQTT, HTTP та WebSockets. Забезпечує безпечне зберігання та обробку даних, автентифікацію пристроїв та інтеграцію з іншими AWS-сервісами. На рисунку 2.2 зображена схема роботи хмарного сховища AWS IoT Core [10].

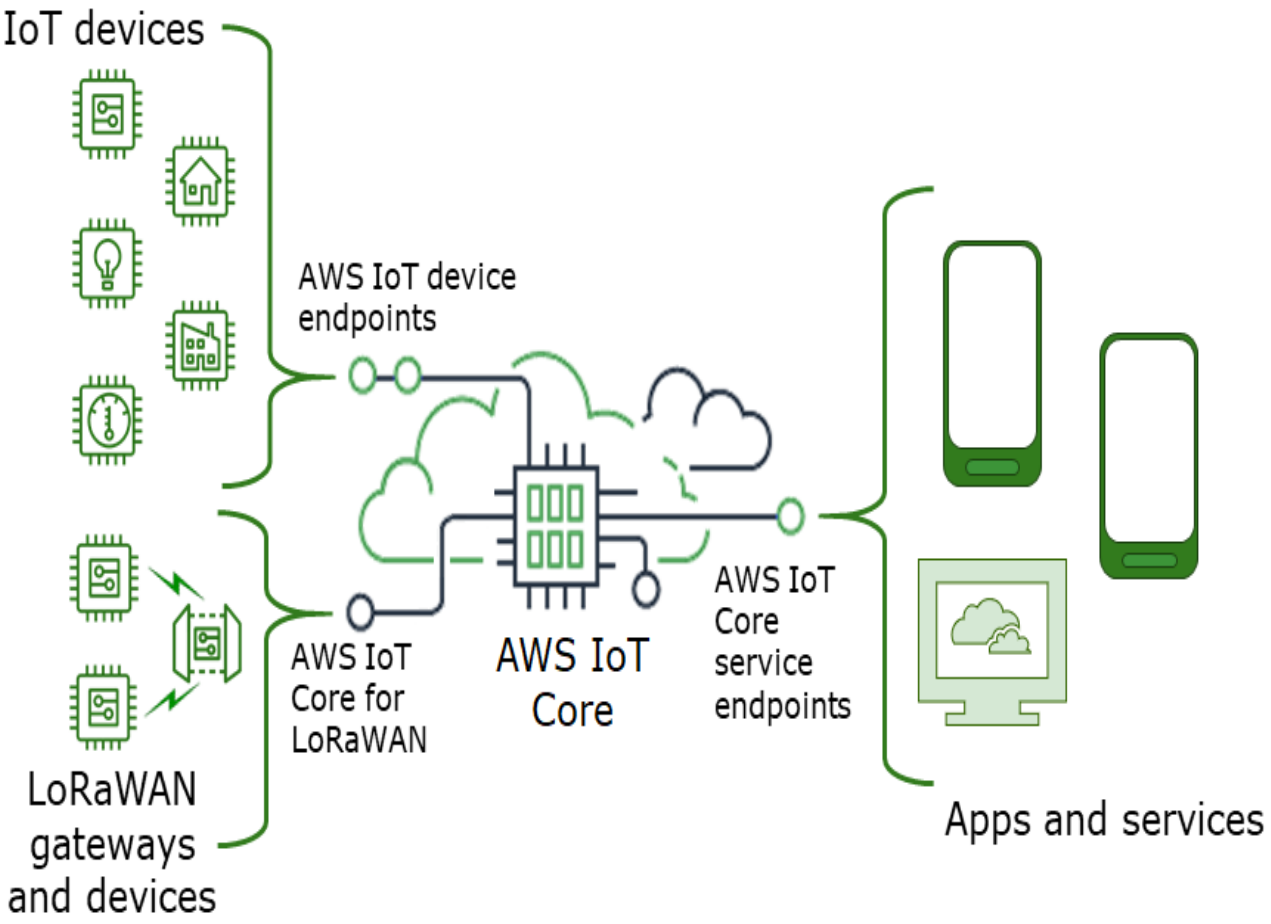

Рисунок 2.2 – Схема роботи хмарного сховища AWS IoT Core

Google Cloud IoT Core: Повністю керована хмарна платформа для IoT від Google. Підтримує протоколи MQTT та HTTP для передачі даних. Інтегрується з іншими хмарними сервісами Google, такими як Cloud Dataflow, Cloud Bigtable та BigQuery. Забезпечує безпеку та автентифікацію пристроїв за допомогою ключів та сертифікатів. На рисунку 2.3 зображена схема роботи хмарного сховища Google Cloud IoT core [11].

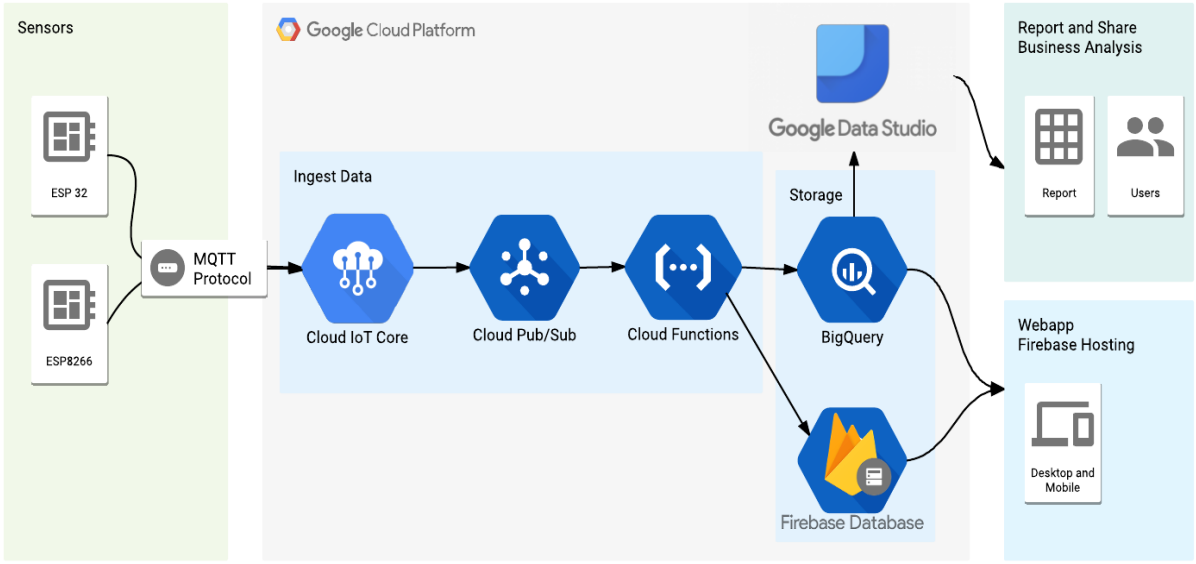

Рисунок 2.3 – Схема роботи хмарного сховища Google Cloud IoT core

Microsoft Azure IoT Hub: Хмарна платформа для IoT від Microsoft, що дозволяє безпечно підключати, моніторити та керувати IoT-пристроями. Підтримує протоколи MQTT, AMQP та HTTP. Інтегрується з іншими сервісами Azure, такими як Azure Stream Analytics, Azure Machine Learning та Azure Storage. Забезпечує автентифікацію пристроїв та безпеку за допомогою токенів доступу та сертифікатів. На рисунку 2.4 зображена схема роботи хмарного сховища Microsoft Azure IoT Hub [12].

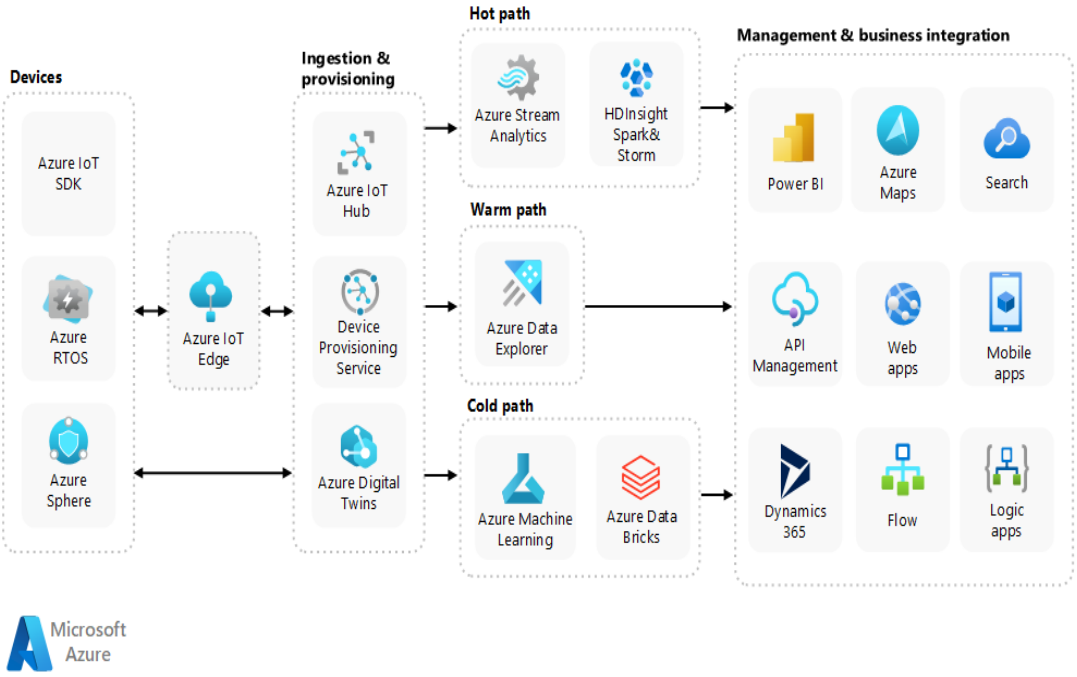

Рисунок 2.4 – Схема роботи хмарного сховища Microsoft Azure IoT Hub

ThingSpeak:Відкрита платформа для збору та аналізу даних IoT від компанії MathWorks. Підтримує протоколи HTTP та MQTT для передачі даних. Пропонує вбудовані інструменти для візуалізації та аналізу даних в режимі реального часу. Забезпечує простий і швидкий спосіб прототипування та розробки IoT-додатків. На рисунок 2.5 зображено побудову графіку в хмарному середовищі ThingSpeak. На рисунку 2.6 зображена схема роботи ThingSpeak. Завдяки відкритості та гнучкості, ThingSpeak дозволяє легко інтегрувати різноманітні IoT-системи та компоненти [13].

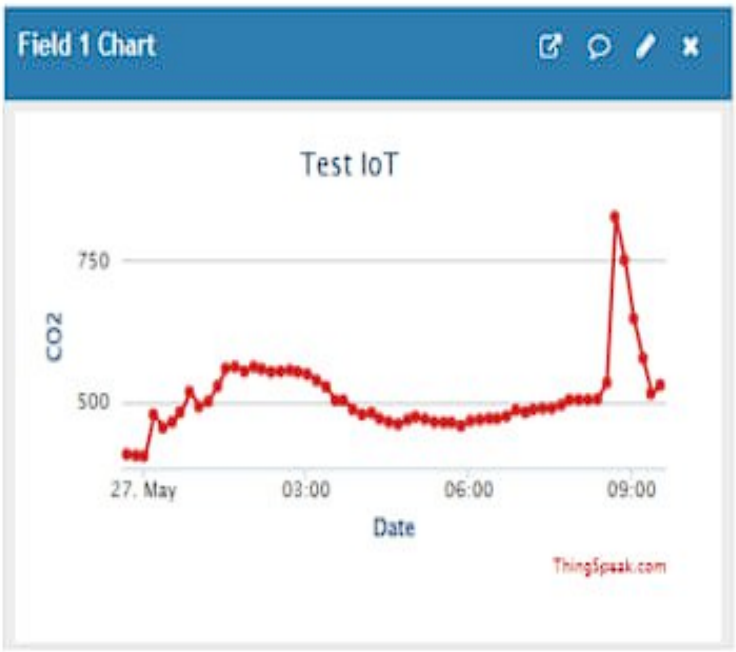

Рисунок 2.5 Вигляд графіків всередині хмарного сховища ThingSpeak

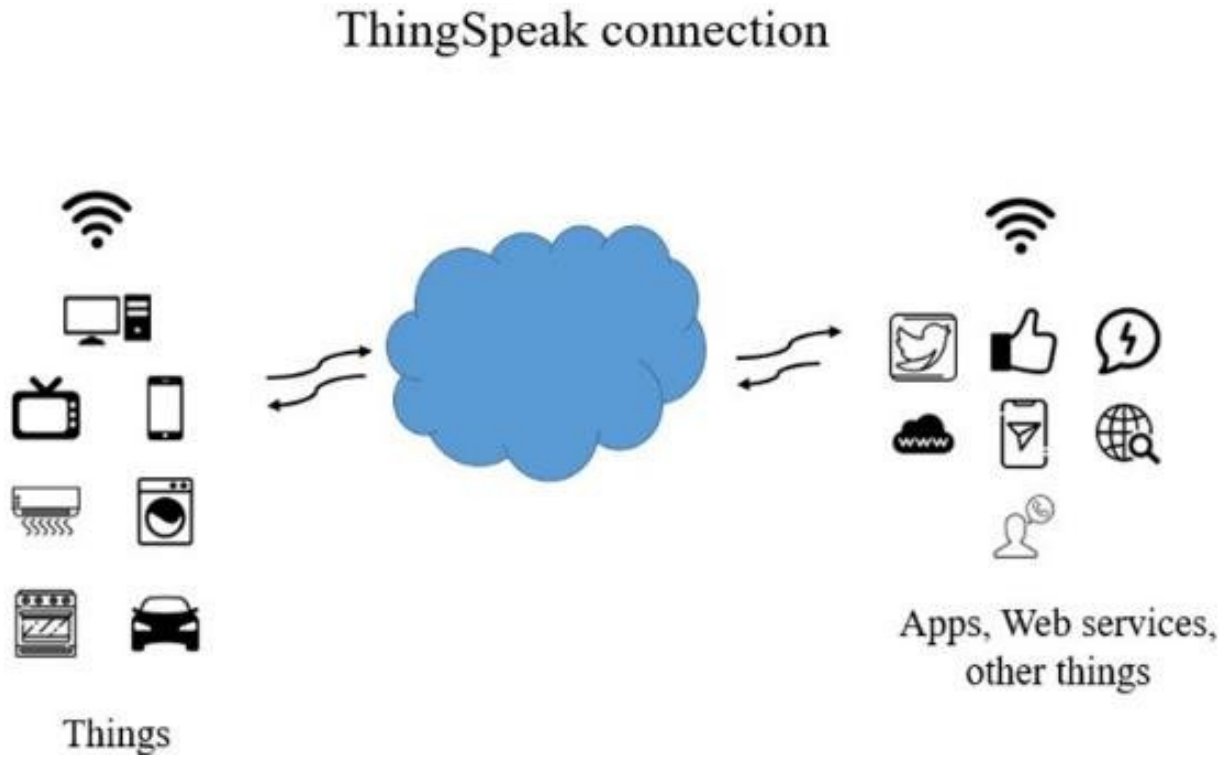

Рисунок 2.6 Схема роботи хмарного сховища ThingSpeak

Одна з ключових функцій ThingSpeak – візуалізація даних у режимі реального часу. Платформа пропонує низку інструментів для створення графіків та діаграм різних типів, зокрема лінійних графіків, гістограм, графіків розсіювання тощо. Ці візуалізації автоматично оновлюються при надходженні нових даних, забезпечуючи зручний моніторинг та аналіз змінних параметрів.

Крім візуалізації, ThingSpeak також забезпечує потужні аналітичні можливості для обробки та аналізу потоків даних. Користувачі можуть застосовувати різноманітні математичні функції та статистичні обчислення, здійснювати аналіз часових рядів, виявляти тренди та закономірності. Це дозволяє отримувати глибоке розуміння зібраних даних та приймати обґрунтовані рішення на їх основі [13].

Важливою перевагою ThingSpeak є її тісна інтеграція з MATLAB – середовищем для чисельних обчислень та аналізу даних. Користувачі можуть імпортувати дані з ThingSpeak безпосередньо в MATLAB для подальшого аналізу, моделювання та візуалізації за допомогою широкого спектру інструментів, доступних у цьому програмному забезпеченні. Це робить

ThingSpeak особливо корисною для науковців, дослідників та інженерів, які працюють з великими об'ємами даних.

ThingSpeak також дозволяє налаштовувати сповіщення та тривоги на основі заданих користувачем критеріїв. Наприклад, можна встановити граничні значення для певних метрик, і отримувати сповіщення при їх перевищенні. Це забезпечує своєчасне виявлення критичних ситуацій та дозволяє вживати необхідних заходів.

Одним з ключових аспектів ThingSpeak є її відкритий програмний інтерфейс (API), який дозволяє розробникам створювати власні додатки для взаємодії з даними платформи або інтегрувати ThingSpeak з існуючими системами моніторингу, контролю та аналізу даних. Це робить платформу ідеальним рішенням для розробки складних IoT-додатків та систем, забезпечуючи гнучкість та масштабованість.

ThingSpeak пропонує безкоштовний рівень доступу для швидкого старту прототипування та тестування IoT-проектів без значних витрат. Однак, платформа також надає різні платні плани, які забезпечують більшу пропускну здатність, додаткові функції та розширені можливості аналізу даних, задовольняючи потреби комерційних та промислових застосувань, великих підприємств та критично важливих систем моніторингу.

Загалом, ThingSpeak є потужною та всеохоплюючою платформою для роботи з даними Інтернету речей. Завдяки своїй відкритості, потужним інструментом візуалізації та аналізу, легкості у використанні та масштабованості, вона стала популярним вибором для розробників, дослідників та ентузіастів IoT у всьому світі. Платформа постійно вдосконалюється, пропонуючи нові функції та можливості, що забезпечують ефективне управління та отримання цінних знань з даних, зібраних з різноманітних пристроїв IoT.

AWS IoT Core, Google Cloud IoT Core та Microsoft Azure IoT Hub є потужними корпоративними рішеннями, які пропонують масштабованість, безпеку та інтеграцію з іншими хмарними сервісами своїх екосистем. ThingSpeak є більш простою та доступною платформою, орієнтованою на прототипування та швидку розробку IoT-додатків. Я обрав ThingSpeak для інтеграції зі своїм

проектом. Тому що вона простота у використанні та швидке впровадження. Вбудовані функції візуалізації даних, що полегшують аналіз та візуалізацію моніторингових даних. Безкоштовний тарифний план для невеликих проєктів, що робить його доступним для мого проєкту. Відкритий вихідний код, що дозволяє вивчати та модифікувати платформу за потреби.

2.3 Розробка клієнтського API

Для отримання даних з віддалених серверів та хмарних сервісів було необхідно розробити клієнтське API, яке б забезпечило безпечну та ефективну взаємодію між мобільним додатком та серверною частиною системи (рис. 2.7).

Рисунок 2.7 – Запити з вибором дати для користувача

API реалізовано у вигляді окремого модуля, який взаємодіє з віддаленими ресурсами через інтерфейс репозиторію. Основні компоненти API включають інтерфейси для визначення запитів, реалізації запитів, які обробляють взаємодію з мережею, та моделі даних для представлення відповідей від сервера. для розробки клієнтського API було використано такі технології та бібліотеки:

Retrofit: потужна бібліотека для створення клієнтських HTTP-API на Android. Вона забезпечує зручний інтерфейс для визначення запитів та обробки відповідей.

OkHttp: ефективна клієнтська бібліотека HTTP, яка використовується Retrofit для виконання мережевих запитів.

Gson: бібліотека для серіалізації та десеріалізації JSON-даних, яка використовується для перетворення відповідей сервера на об'єкти моделей даних.

Ці бібліотеки були обрані завдяки їх широкому використанню в Android-розробці, простоті налаштування та інтеграції, а також високій продуктивності та можливостям налаштування.

Для взаємодії з віддаленими серверами та хмарними сервісами використовувався протокол HTTP. Запити виконувалися за допомогою різних HTTP-методів (GET, POST, PUT, DELETE) залежно від операції, яка потрібна. Формат даних для обміну даними з сервером був JSON, який є широко розповсюдженим та легко обробляється на стороні клієнта та сервера.

2.4 Висновки

В цьому розділі було здійснено опис та обґрунтування вибору оптимальних засобів та технологій для вирішення поставленої задачі. А саме, обґрунтовано вибір необхідного фреймворку для розробки мобільних додатків Android, а також було розглянуто важливість використання хмарних платформ для ефективного зберігання та обробки даних. Вибір фреймворку для розробки Android-додатків базувався на кількох ключових критеріях, таких як популярність, наявність великої спільноти розробників, наявність документації та підтримка необхідних функцій. Зокрема, було обрано Android Studio як інтегроване середовище розробки (IDE) завдяки його потужним інструментам для розробки, тестування та деплою Android-додатків. Щодо хмарних платформ, було здійснено аналіз кількох провідних рішень на ринку, таких як Google Cloud IoT Core, Microsoft Azure IoT Hub, AWS IoT Core та ThingSpeak. Використання хмарних платформ дозволяє не тільки зберігати великі обсяги даних, але й забезпечувати їх безпечну передачу та обробку в режимі реального часу. Зокрема, Google Cloud IoT Core забезпечує надійну інтеграцію з іншими сервісами Google Cloud, що дозволяє здійснювати масштабовану обробку даних,

а ThingSpeak надає зручні інструменти для візуалізації та аналізу даних. Крім того, у розділі було розроблено клієнтське API для взаємодії з ThingSpeak.

3 РЕЗУЛЬТАТИ ДОСЛІДЖЕНЬ, АНАЛІЗ ТА ВІЗУАЛІЗАЦІЯ ДАНИХ

3.1 Аналіз бібліотека для візуалізація даних в Android

MPAndroidChart - це потужна бібліотека з відкритим вихідним кодом для малювання графіків в Android застосунках, написана на Java. Вона підтримує різноманітні типи графіків, включаючи лінійні, стовпчикові, кругові діаграми та інші.

Переваги:

- Широкий спектр можливостей для налаштування та персоналізації графіків (легенди, підписи, анімації, взаємодія з користувачем та ін.);
- Проста інтеграція та зручний API;
- Підтримка масштабування, прокрутки та збереження стану графіків;
- Активна спільнота розробників та постійна підтримка бібліотеки;
- Добра документація та приклади використання.

Недоліки:

- Відносно великий розмір бібліотеки, що може вплинути на розмір додатку;
- Деякі функції можуть бути недостатньо гнучкими або складними для налаштування;
- Можливі проблеми з продуктивністю при малюванні великої кількості даних [14].

AChartEngine - це відкрита бібліотека для створення графіків та діаграм в Android застосунках. Вона підтримує різні типи графіків, такі як лінійні, стовпчикові, кругові та інші.

Переваги:

- Легка інтеграція та простий API.
- Підтримка анімацій та інтерактивної взаємодії з графіками.
- Можливість зберігати графіки у форматах PNG та JPEG.
- Достатньо гнучка для налаштування.

Недоліки:

- Обмежені можливості для персоналізації графіків.
- Менш активна спільнота розробників та підтримка порівняно з іншими бібліотеками.
- Можливі проблеми з продуктивністю при відображенні великої кількості даних.
- Відсутність підтримки деяких нових функцій Android, таких як вектори та анімації.

HelloCharts - це бібліотека з відкритим вихідним кодом для створення діаграм та графіків в Android застосунках. Вона підтримує різні типи графіків, такі як лінійні, стовпчикові, кругові та інші.

Переваги:

- Простий та зрозумілий API.
- Легка інтеграція в застосунки.
- Підтримка анімацій та інтерактивної взаємодії з графіками.
- Гнучкість налаштувань та персоналізації графіків.

Недоліки:

- Обмежені можливості для налаштування вигляду графіків.
- Менша активність спільноти розробників порівняно з іншими бібліотеками.
- Можливі проблеми з продуктивністю при відображенні великої кількості даних.
- Відсутність деяких додаткових функцій, наявних в інших бібліотеках.

GraphView - це легка бібліотека з відкритим вихідним кодом для створення графіків та діаграм в Android застосунках. Вона підтримує лінійні та стовпчикові графіки.

Переваги:

- Простий та легкий у використанні API.
- Легка інтеграція в застосунки.
- Підтримка масштабування та прокрутки графіків.
- Невеликий розмір бібліотеки, що не збільшує значно розмір додатку.

Недоліки:

- Обмежені можливості для персоналізації графіків.
- Відсутність підтримки деяких типів графіків, таких як кругові діаграми.
- Менша активність спільноти розробників порівняно з іншими бібліотеками.
- Обмежені можливості для взаємодії з графіками та анімацій.

Обґрунтування вибору MPAndroidChart:

Після ретельного аналізу переваг та недоліків різних бібліотек для малювання графіків в Android застосунках, я обрав MPAndroidChart як найбільш оптимальне рішення для свого проекту. Основні причини вибору цієї бібліотеки полягають у наступному:

Широкий спектр можливостей та гнучкість налаштувань. MPAndroidChart пропонує різноманітні типи графіків та багато опцій для їх персоналізації, що дозволяє створювати графіки відповідно до вимог проекту.

Активна спільнота та постійна підтримка. MPAndroidChart має велику та активну спільноту розробників, що забезпечує постійну підтримку, оновлення та вирішення проблем бібліотеки.

Добра документація та приклади. Наявність детальної документації та численних прикладів використання полегшує процес інтеграції та роботи з бібліотекою.

Підтримка масштабування, прокрутки та збереження стану. Ці функції є важливими для забезпечення зручної взаємодії користувача з графіками.

Простота інтеграції та зручний API. MPAndroidChart легко інтегрується в проекти та має зрозумілий API, що спрощує процес розробки.

Хоча бібліотека має деякі недоліки, такі як відносно великий розмір та можливі проблеми з продуктивністю при роботі з великою кількістю даних, її переваги перевищують ці недоліки для цілей мого проекту. Загалом, MPAndroidChart є потужним та гнучким рішенням для візуалізації даних у вигляді графіків в Android застосунках.

3.2 Системне моделювання за допомогою UML діаграм

Цей підрозділ містить UML діаграми, які використовуються для візуалізації архітектури та функціональності системи. UML є стандартом для моделювання програмного забезпечення, який дозволяє чітко та зрозуміло представляти різні аспекти системи, такі як структура, поведінка та взаємодії між компонентами. Тут будуть представлені різні типи діаграм. На рисунку 3.1 зображена діаграма прецедентів (use case), яка ілюструє різні способи аналізу даних, що доступні користувачеві Android застосунку [15].

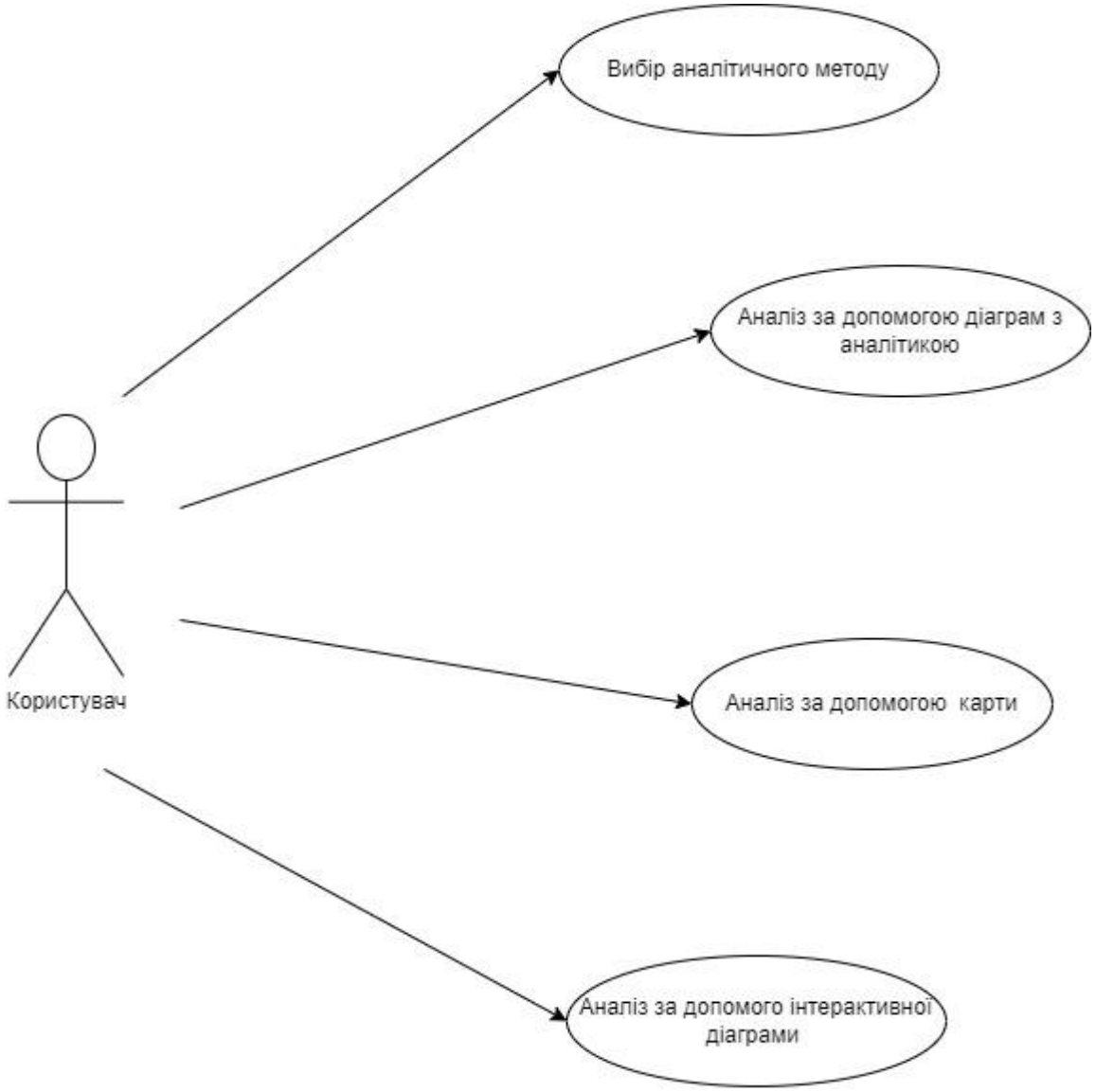

Рисунок 3.1 – Діаграма Use Case Android застосунку

Вибір аналітичного методу - це варіант використання, який дозволяє користувачеві вибрати метод або підхід для проведення аналізу даних.

– "Аналіз за допомогою діаграм з аналітикою" - цей варіант використання дає користувачеві можливість проводити аналіз даних за допомогою різних діаграм та візуалізацій з аналітичними функціями.

– "Аналіз за допомогою карти" - тут користувач може аналізувати дані з використанням географічних карт та візуалізації просторових даних.

– "Аналіз за допомогою інтерактивної діаграми" - цей варіант використання дозволяє користувачеві проводити аналіз з використанням інтерактивних діаграм, які забезпечують можливості взаємодії та маніпуляції даними.

На рисунку 3.2 зображена UML діаграма класів вона представляє структуру мобільного застосунку для аналізу даних та візуалізації.

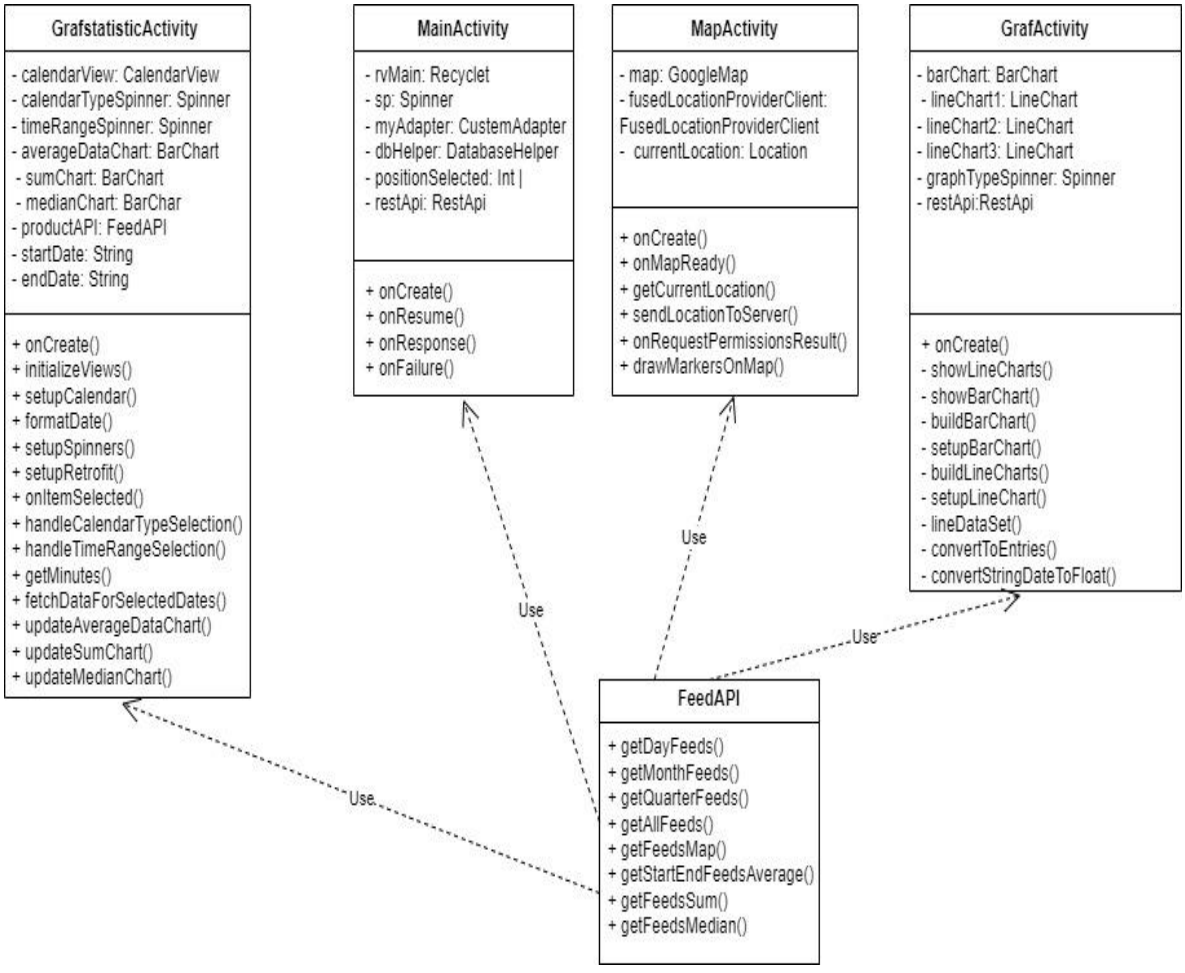

Рисунок 3.2 – Діаграма класів Android застосунку

На діаграмі зображено основні класи застосунку, їх атрибути, методи та взаємозв'язки між ними. Кожен клас відповідає за певний аспект

функціональності застосунку, а взаємодія між класами забезпечує комплексну роботу системи.

На діаграмі показано взаємодії між класами за допомогою відносин "use" (використання). Наприклад, GrafstatisticActivity використовує методи FeedAPI для отримання даних для графіків, MainActivity використовує класи RestApi, DatabaseHelper та CustomAdapter для відображення даних у RecyclerView, а MapActivity та GrafActivity використовують методи FeedAPI та RestApi для отримання даних для карти та графіків відповідно.

Ця діаграма дає чітке уявлення про структуру застосунку, взаємодію між його компонентами та основні функції кожного з класів, що є важливим для розуміння архітектури системи та її подальшого розширення або підтримки.

На рисунку 3.3 зображено діаграму взаємодії (sequence diagram) вона відображає процеси взаємодії користувача з мобільним застосунком для аналізу та візуалізації даних.

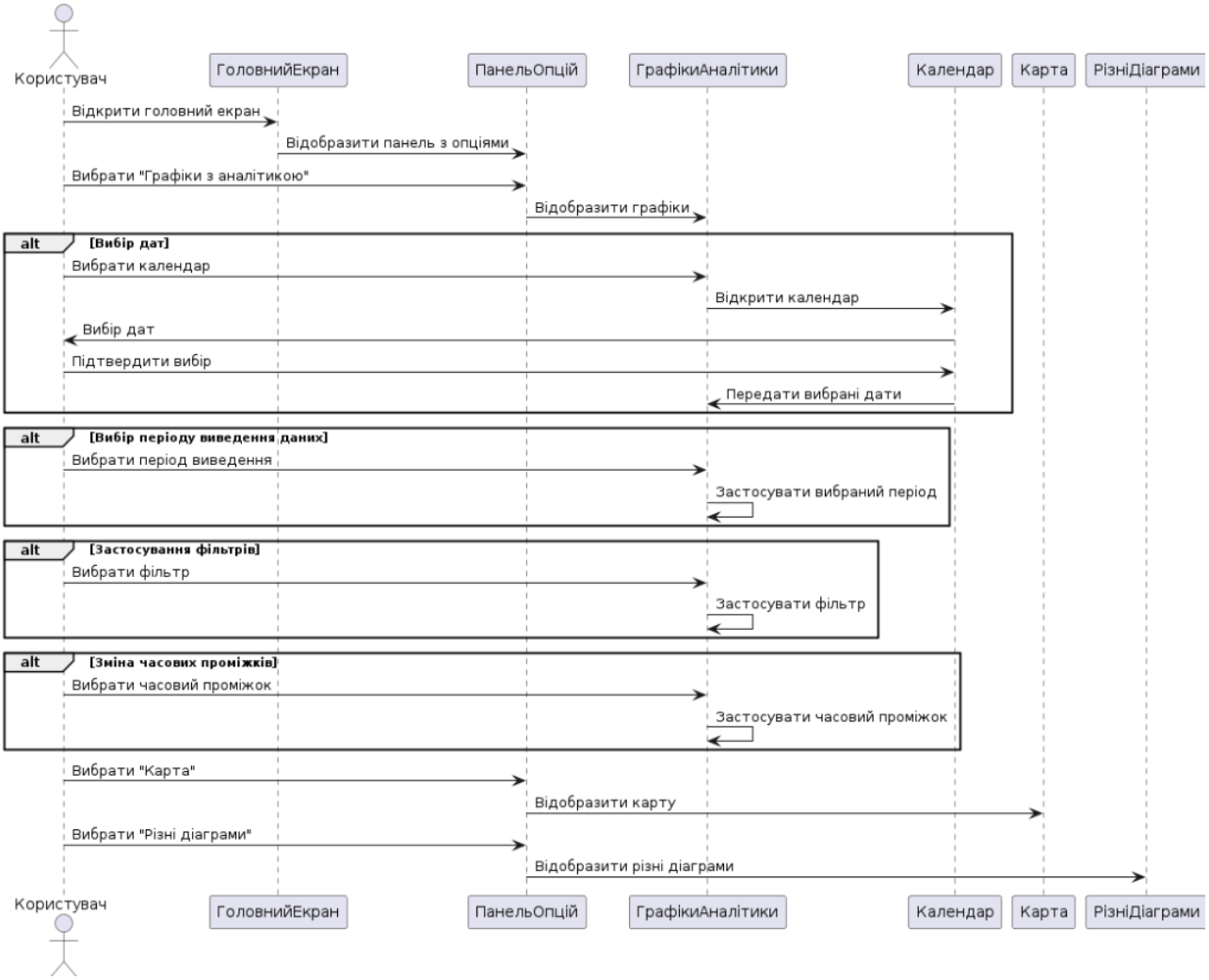

Рисунок 3.3 – Діаграма взаємодії Android застосунку

На діаграмі представлені основні сценарії використання, включаючи вибір графіків аналітики, вибір періоду виведення даних, застосування фільтрів, зміна часових проміжків, а також використання інтерактивної карти.

Відкриття Головного Екрану. Користувач відкриває головний екран застосунку, після чого система відображає панель з доступними опціями для аналізу даних. Це є початковим етапом взаємодії користувача із застосунком.

Вибір Опції "Графіки з Аналітикою:

Користувач вибирає опцію "Графіки з аналітикою". Система відкриває екран, де відображаються графіки з аналітичними даними. Цей функціонал дозволяє користувачу бачити візуалізацію даних і взаємодіяти з ними.

Вибір Дат:

У разі необхідності вибору дат, користувач відкриває календар та вибирає необхідні дати. Після підтвердження вибору, обрані дати передаються системі, яка оновлює графіки відповідно до нового діапазону дат.

Вибір Періоду Виведення Даних:

Користувач може вибрати період виведення даних (наприклад, за день, місяць, квартал тощо). Після вибору, система застосовує вибраний період і оновлює відображення графіків.

Застосування Фільтрів:

Користувач може застосовувати різні фільтри для деталізації відображуваних даних. Вибрані фільтри дозволяють користувачу бачити лише ті дані, які відповідають заданим критеріям.

Зміна Часових Проміжків:

Для більш детального аналізу, користувач може змінювати часові проміжки (наприклад, 10 хвилин, 15 хвилин, 30 хвилин тощо). Це дозволяє налаштувати відображення даних у відповідності до конкретних потреб.

Вибір Опції "Карта":

При виборі опції "Карта", система відкриває екран з інтерактивною картою. На карті відображаються маркери, які представляють дані з географічною прив'язкою (широта, довгота). Користувач може взаємодіяти з картою, змінювати масштаб, переміщуватися та отримувати додаткову інформацію про маркери.

Вибір Опції "Різні Діаграми":

При виборі опції "Різні діаграми", система відображає різні типи діаграм, що дозволяє користувачу вибирати найбільш зручний формат представлення даних.

Ця діаграма взаємодії ілюструє послідовність дій користувача при роботі з різними компонентами застосунку, забезпечуючи глибоке розуміння логіки та функціональних можливостей системи. Вона допомагає візуалізувати та зрозуміти, як різні елементи застосунку взаємодіють між собою та як користувач може ефективно використовувати ці можливості для аналізу даних.

На рисунку 3.4 зображена діаграма діяльності відображає процес взаємодії користувача з мобільним застосунком для аналізу даних. Ця діаграма детально ілюструє кроки, які виконує користувач, починаючи з відкриття головного екрану застосунку і закінчуючи завершенням взаємодії з графіками. Вона допомагає зрозуміти логіку роботи системи та взаємодію між різними компонентами застосунку.

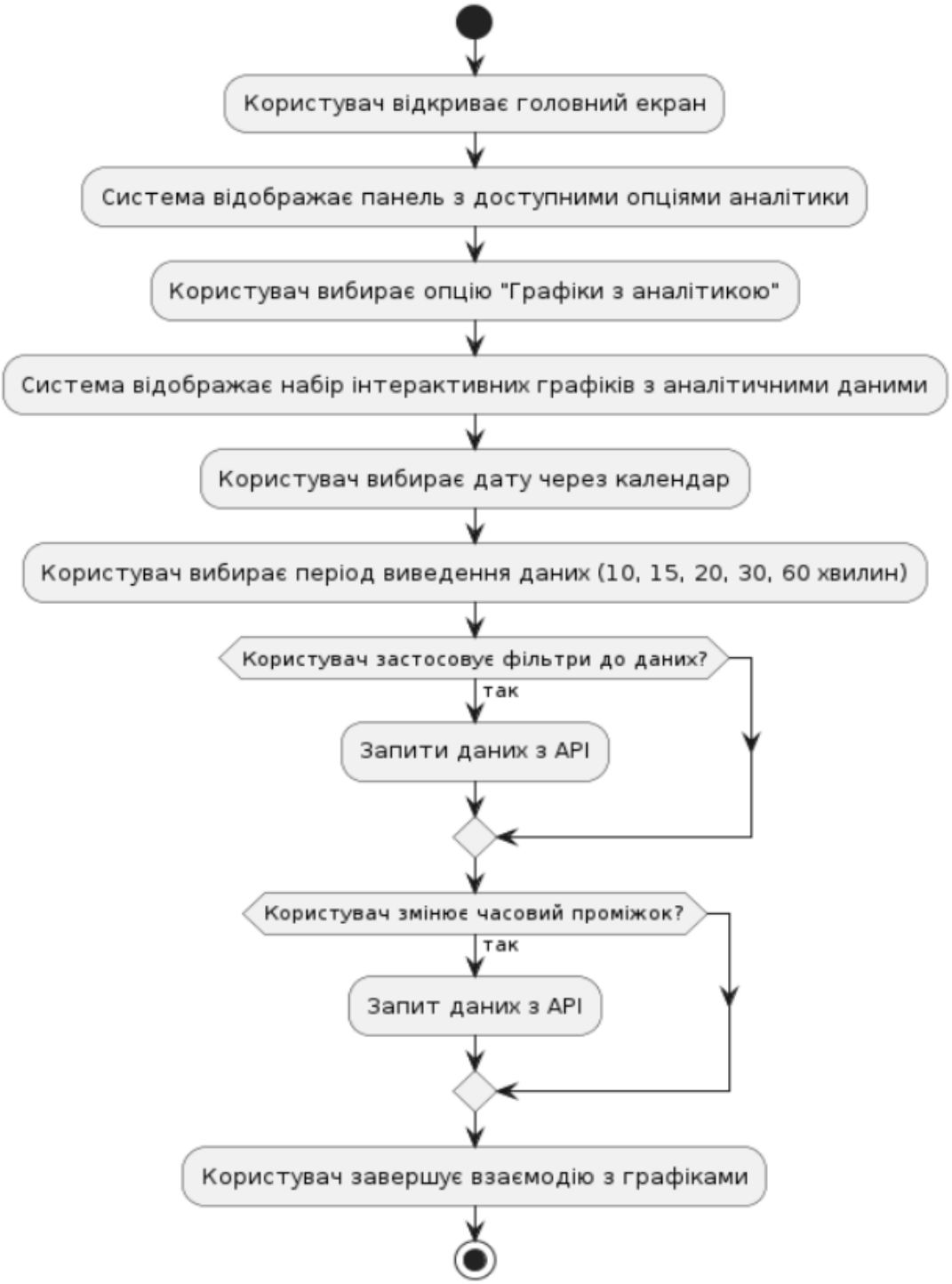

Рисунок 3.4 – Діаграма діяльності Android застосунку

Початок Взаємодії:

Процес взаємодії розпочинається з того, що користувач відкриває головний екран застосунку. Це є стартова точка на діаграмі, що позначена як чорне коло.

Відображення Панелі з Опціями Аналітики:

Після відкриття головного екрану, система автоматично відображає панель з доступними опціями аналітики. На цій панелі користувач може обрати різні функції, такі як перегляд графіків або карт.

Вибір Опції "Графіки з Аналітикою":

Користувач вибирає опцію "Графіки з аналітикою". Це призводить до переходу до наступного етапу, де система відображає набір інтерактивних графіків з аналітичними даними.

Відображення Інтерактивних Графіків:

Система відображає набір інтерактивних графіків, що дозволяє користувачу переглядати дані в різних форматах і взаємодіяти з ними.

Вибір Дати через Календар:

Користувач може вибрати конкретну дату або діапазон дат через інтегрований календар. Це дозволяє відфільтрувати дані за обраними датами, що є важливим для більш точного аналізу.

Вибір Періоду Виведення Даних:

Далі користувач вибирає період виведення даних. Доступні опції можуть включати 10, 15, 20, 30 або 60 хвилин. Це дозволяє користувачу налаштувати деталізацію відображуваних даних відповідно до своїх потреб.

Застосування Фільтрів до Даних:

Якщо користувач бажає застосувати фільтри до даних, система перевіряє цей вибір. У разі позитивного рішення, система виконує запит до API для отримання відфільтрованих даних. Фільтри можуть включати різні параметри, такі як тип даних, місцезнаходження або інші метрики.

Запити Даних з API:

Система здійснює запити до API для отримання необхідних даних. Це дозволяє застосунку динамічно оновлювати інформацію на графіках відповідно до встановлених користувачем фільтрів або обраного періоду.

Зміна Часового Проміжку:

Користувач також має можливість змінювати часовий проміжок для відображення даних. Це може бути корисно для детальнішого аналізу на коротких або довгих інтервалах часу.

Завершення Взаємодії з Графіками:

Після застосування всіх необхідних фільтрів і налаштувань користувач завершує взаємодію з графіками. Це позначено кінцевою точкою на діаграмі, що відображається як чорне коло.

Ця діаграма діяльності надає візуальне представлення логіки та послідовності дій користувача в мобільному застосунку для аналізу даних. Вона допомагає зрозуміти, як різні компоненти системи взаємодіють між собою та як користувач може ефективно використовувати цей функціонал для отримання аналітичних даних.

На рисунку 3.5 зображена діаграмі станів,, відображено процес взаємодії користувача з мобільним застосунком для аналізу даних. Ця діаграма ілюструє різні стани, через які проходить система під час роботи користувача із застосунком, починаючи з відкриття головного екрану і закінчуючи завершенням взаємодії. Вона допомагає зрозуміти логіку роботи системи, послідовність станів і переходи між ними.

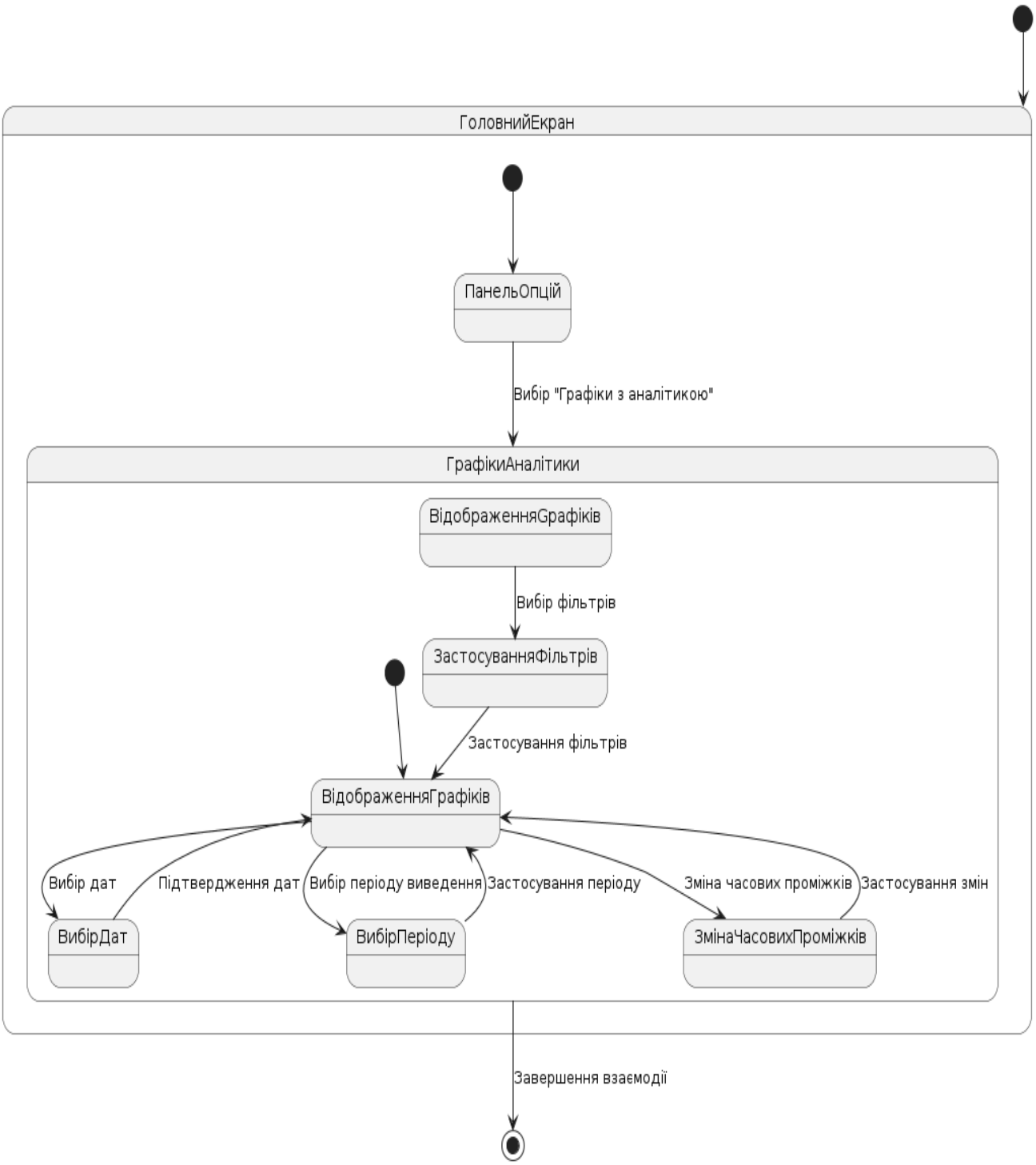

Рисунок 3.5 – Діаграма станів та переходів Android застосунку

Початок Взаємодії:

Процес взаємодії розпочинається з того, що користувач відкриває головний екран застосунку. Цей стан є початковою точкою на діаграмі і позначений як чорне коло.

Відображення Панелі з Опціями Аналітики:

Після відкриття головного екрану, система переходить до стану "Панель Опцій", де автоматично відображає панель з доступними опціями аналітики. Користувач може обрати різні функції, такі як перегляд графіків або карт.

Вибір Опції "Графіки з Аналітикою":

Користувач вибирає опцію "Графіки з аналітикою", що приводить систему до переходу у стан "Графіки Аналітики". Це стан, де користувач може взаємодіяти з графіками, що відображають аналітичні дані.

Відображення Інтерактивних Графіків:

В стані "Графіки Аналітики" система відображає набір інтерактивних графіків. Користувач може переглядати ці графіки, обирати різні параметри для аналізу даних.

Вибір Дати через Календар:

Користувач може вибрати конкретну дату або діапазон дат через інтегрований календар. Це призводить до переходу в стан "ВибірДат", де користувач обирає необхідні дати і підтверджує свій вибір. Після підтвердження вибору система повертається до стану "Відображення Графіків".

Вибір Періоду Виведення Даних:

Далі користувач вибирає період виведення даних. Це може бути 10, 15, 20, 30 або 60 хвилин. Користувач переходить у стан "Вибір Періоду", де він обирає відповідний період. Після вибору система знову повертається до стану "Відображення Графіків".

Застосування Фільтрів до Даних:

Якщо користувач бажає застосувати фільтри до даних, система переходить у стан "Застосування Фільтрів". Тут користувач обирає необхідні фільтри, і система застосовує їх до даних. Після застосування фільтрів система повертається до стану "Відображення Графіків".

Зміна Часового Проміжку:

Користувач також має можливість змінювати часовий проміжок для відображення даних. Це може бути корисно для детальнішого аналізу на коротких або довгих інтервалах часу. У цьому випадку система переходить у

стан "Зміна Часових Проміжків", де користувач обирає новий часовий проміжок. Після цього система повертається до стану "Відображення Графіків".

Завершення Взаємодії з Графіками:

Після застосування всіх необхідних фільтрів і налаштувань користувач завершує взаємодію з графіками.

Ця діаграма станів надає візуальне представлення логіки та послідовності дій користувача в мобільному застосунку для аналізу даних. Вона допомагає зрозуміти, як різні компоненти системи взаємодіють між собою та як користувач може ефективно використовувати цей функціонал для отримання аналітичних даних.

3.3 Опис та функціональність Android-застосунку

На рисунку 3.6 зображено макет інтерфейсу користувача. Для створення основного макету додатку використано компонент relativeLayout. Це один з найгнучкіших типів макетів в Android, який дозволяє розміщувати елементи відносно один одного. Наприклад, один елемент можна розмістити вище, нижче, праворуч або ліворуч від іншого елемента, що забезпечує максимальну гнучкість у дизайні інтерфейсу користувача. Використання relativeLayout дозволяє легко створювати складні макети з відносним позиціонуванням елементів, що особливо корисно при створенні адаптивних інтерфейсів, які мають добре виглядати на різних розмірах екранів [16].

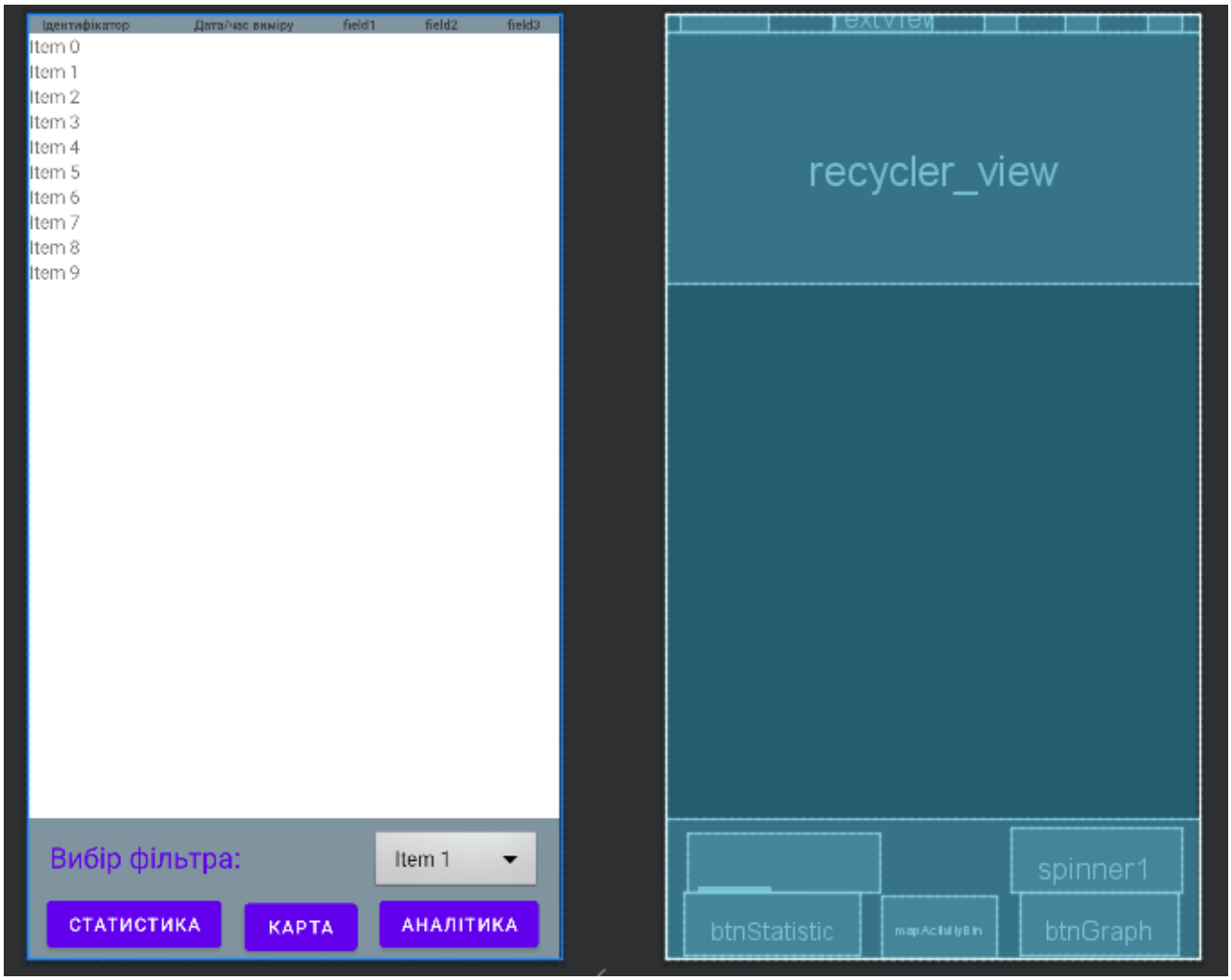

Рисунок 3.6 – Макет інтерфейсу користувача

Компонент RecyclerView:

Основним елементом відображення даних у додатку є компонент RecyclerView. Цей компонент займає більшу частину екрана і призначений для ефективного відображення великої кількості даних у вигляді списку. RecyclerView забезпечує високу продуктивність та гнучкість у порівнянні зі старим компонентом listView, завдяки використанню шаблонів ViewHolder, які зменшують кількість викликів findViewById і оптимізують процес відображення даних.

Налаштування адаптера для recyclerView дозволяє визначити, як кожен елемент списку буде відображатися на екрані. Адаптер створює нові елементи списку при необхідності та прив'язує дані до існуючих елементів, що забезпечує плавне прокручування списку та ефективне використання ресурсів.

Спінери та кнопки:

В нижній частині макету розміщено спінер , який призначений для вибору фільтрів даних. Спінер дозволяє користувачам вибирати різні параметри для фільтрації даних, що забезпечує гнучкість у налаштуванні відображення інформації. Це дозволяє користувачам швидко і зручно отримувати тільки ті дані, які їх цікавлять, що покращує користувацький досвід.

Також додано три кнопки: статистика, аналітика, карта, які відповідають за навігацію до відповідних розділів додатку – статистика, графіки та карта. Ці кнопки забезпечують швидкий доступ до основних функцій додатку, дозволяючи користувачам легко перемикатися між різними типами візуалізації даних.

Екран статистики:

Для екрана статистики використано компоненти TextView, які дозволяють відображати статистичні дані та діаграми. TextView – це базовий компонент для відображення тексту, який можна налаштовувати для відображення різноманітних стилів тексту. Впроваджено можливість взаємодії користувача з даними через компоненти Button та Spinner, що дозволяє користувачам вибирати різні параметри для відображення та аналізу даних. Це забезпечує інтерактивність та гнучкість у роботі з даними.

Екран карти:

Для відображення географічної інформації на екрані карти використано компонент MapView. Цей компонент дозволяє інтегрувати карти Google Maps у додаток і відображати місця розташування IoT-пристроїв. Використання MapView забезпечує можливість взаємодії з картою, такі як зміна масштабу, переміщення по карті та відображення маркерів, що робить додаток більш інтерактивним та зручним для користувачів.

Екран аналітики:

Для відображення графіків і діаграм на екрані аналітики додано компоненти LineChart та BarChart. Ці компоненти дозволяють відображати часові ряди та гістограми, надаючи користувачам візуальне уявлення про зібрані

дані. Використання графіків і діаграм значно полегшує аналіз даних, дозволяючи виявляти тенденції та аномалії в даних.

Опис макету в XML:

Використання XML для опису макетів дозволяє чітко визначити структуру інтерфейсу. XML-файли легко модифікувати і вони зрозумілі для інших розробників, що спрощує командну роботу над проектом. Кожен елемент інтерфейсу, такий як кнопки, спінери, текстові поля, було додано у XML-файлі, де визначено їх властивості, включаючи розмір, розташування, відступи та стилі.

Стилі і теми:

Для забезпечення уніфікованого вигляду всіх компонентів інтерфейсу використано механізм стилів і тем. Це дозволяє централізовано змінювати вигляд додатку, не редагуючи кожен елемент окремо. Стилі та теми визначаються у XML-файлах ресурсів, що дозволяє легко змінювати зовнішній вигляд додатку, забезпечуючи цілісність даних та зручність у підтримці коду.

Взаємодія з елементами

Для кожної кнопки та спінера було налаштовано обробники подій (event handlers), що дозволяє реагувати на дії користувача, такі як натискання кнопок або вибір елементів у спінері. Обробники подій реалізовано у відповідних Activity або Fragment класах, де визначено логіку роботи додатку. Це забезпечує інтерактивність і динамічність інтерфейсу, дозволяючи додатку відповідати на дії користувача в режимі реального часу.

Запити до API

При взаємодії користувача з інтерфейсом, наприклад, застосування фільтрів або вибір дати, відправляються запити до REST API сервера. Це дозволяє отримувати необхідні дані для відображення у додатку. Запити реалізовано за допомогою бібліотеки Retrofit, що спрощує роботу з HTTP-запитами в Android. Retrofit забезпечує зручний і ефективний спосіб виконання запитів та обробки відповіді сервера.

Збереження даних

Для забезпечення доступу до даних в офлайн-режимі використовується база даних SQLite. За допомогою бібліотеки Room організовано збереження та

доступ до даних, що дозволяє зменшити залежність від постійного підключення до Інтернету. Room забезпечує зручний інтерфейс для роботи з базою даних, спрощуючи операції збереження, читання та оновлення даних.

Тестування та налагодження

Макети та функціонал додатку тестувалися на різних емуляторах та реальних пристроях для забезпечення сумісності та коректної роботи на різних версіях Android та розмірах екранів. Це дозволяє виявляти та усувати проблеми, пов'язані з різними характеристиками пристроїв, такими як роздільна здатність екрана та продуктивність.

Розроблений додаток на основі Android являє собою універсальну, інтерактивну платформу для виведення, аналізу та візуалізації даних з IoT-пристроїв. Основною метою цього додатку є надання користувачам інструментів для зручного доступу до даних, їх аналізу та подальшої візуалізації, що дозволяє робити важливі висновки та приймати обґрунтовані рішення на основі отриманих даних.

Після запуску додатку користувач потрапляє на основний екран, де він може одразу ж взаємодіяти з даними. На цьому екрані представлено таблицю даних, яка дозволяє користувачеві переглядати актуальну інформацію у зручному табличному форматі. Тут також знаходяться кнопки для навігації, які допомагають користувачеві швидко переміщатися між різними секціями додатку. Крім того, на основному екрані є функціональні кнопки для отримання статистичних показників, що дає змогу користувачеві швидко отримувати важливу аналітичну інформацію.

На додаткових екранах додатку реалізовано функціонал візуалізації даних вимірювань. Ці дані відображаються у вигляді часових рядів, що дозволяє користувачам аналізувати динаміку змін параметрів у часі. Графіки мають інтерактивні елементи, що дозволяють масштабувати, переміщувати та детально аналізувати конкретні відрізки часу. Така візуалізація значно покращує розуміння складних даних та робить їх більш доступними для користувача.

Фільтрація даних у додатку реалізована шляхом зміни параметрів REST API сервера ThingSpeak. Користувач може задавати різні параметри фільтрації

для отримання необхідних даних за конкретні періоди часу. Це дозволяє налаштовувати відображення даних відповідно до потреб користувача. Наприклад, можна отримати дані за останню годину, день, тиждень або інший обраний період. Завдяки цьому функціоналу, користувачі можуть зручно і швидко налаштовувати додаток для отримання саме тих даних, які їм потрібні в даний момент.

Для забезпечення безперебійного доступу до даних навіть у разі відсутності інтернет-з'єднання, додаток підтримує режим офлайн. У цьому режимі використовуються збережені раніше дані. Актуальні дані зберігаються у вбудованій базі даних sqlite за допомогою бібліотеки Room. Це дозволяє додатку працювати автономно, що є особливо корисним для користувачів, які знаходяться в умовах обмеженого доступу до Інтернету. Крім того, збереження даних локально допомагає зменшити обсяги даних, які необхідно завантажувати з Інтернету, що, в свою чергу, знижує навантаження на мережу та прискорює роботу додатку.

Завдяки зазначеним функціональним можливостям, додаток є потужним інструментом для моніторингу та аналізу даних з IoT-пристроїв. Його універсальність та інтерактивність роблять його незамінним для користувачів, які прагнуть мати постійний доступ до аналітичних даних, незалежно від наявності інтернет-з'єднання. Вбудована можливість фільтрації та візуалізації даних допомагає користувачам зосередитися на найбільш важливих аспектах, а збереження даних у локальній базі дозволяє знизити витрати на передачу даних.

Цей додаток може бути використаний у різних сферах, включаючи промисловість, сільське господарство, енергетику та багато інших галузей, де важливо мати оперативний доступ до даних та можливість їхнього детального аналізу.

Крім зазначених функціональних можливостей, додаток має великий потенціал для подальшого розширення та інтеграції з іншими системами та платформами.

Наприклад, можна додати підтримку інших типів датчиків та IoT-пристроїв, що дозволить збирати та аналізувати ще більше різноманітних даних.

Інтеграція з популярними хмарними платформами для обробки великих даних, такими як AWS або Google Cloud, може забезпечити додаткові можливості для зберігання та аналізу даних. Крім того, розробка веб-версії додатку або його інтеграція з існуючими корпоративними системами дозволить забезпечити доступ до даних з різних пристроїв та платформ, що значно підвищить зручність використання та гнучкість додатку. Такі можливості роблять додаток не лише потужним інструментом для поточного використання, але й перспективною платформою для майбутнього розвитку та вдосконалення.

Загальний вигляд застосунку, представлений на рис. 3.7, демонструє доступний функціонал та інтерфейс додатку [17].

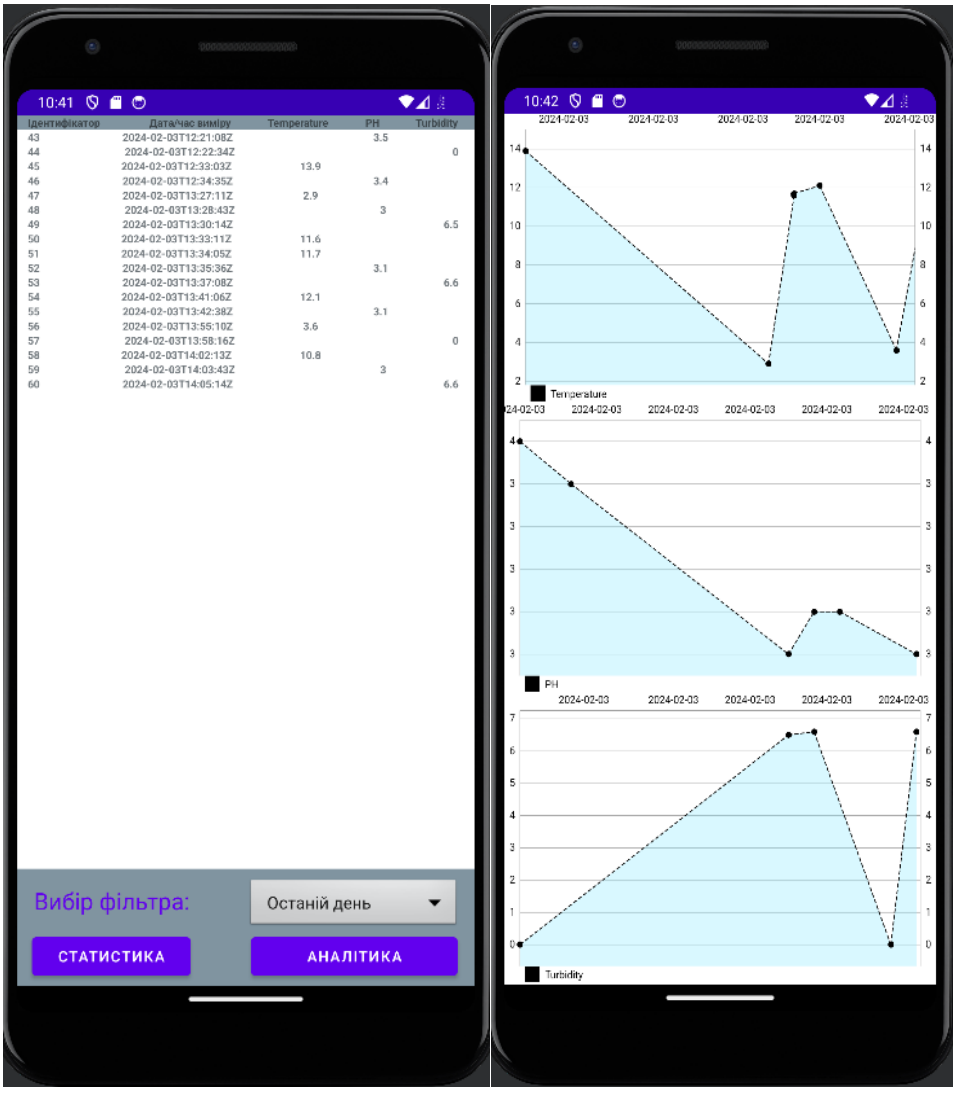

Рисунок 3.7 – Загальний вигляд застосунку

Він дозволяє користувачам швидко ознайомитися з основними можливостями та зрозуміти, як використовувати додаток для виведення та аналізу даних. На головному екрані додатку розміщені елементи навігації, такі як панелі з опціями для доступу до різних розділів аналітики, візуалізації та фільтрації даних. На рисунку 3.8 зображено код який відповідає за оновлення графіків.

```
177
        dzurasergij4@gmail.com <dzurasergij4@gmail.com>
178     private fun updateAverageDataChart(feeds: List<Feed>?) {
179         val entries = mutableListOf<BarEntry>()
180         feeds?.forEachIndexed { index, feed ->
181             feed.field1?.let { it: String
182                 entries.add(BarEntry(index.toFloat(), it.toFloat()))
183             }
184         }
185         val dataSet = BarDataSet(entries, label: "Average Data")
186         val barData = BarData(dataSet)
187         averageDataChart.data = barData
188         averageDataChart.invalidate()
189     }
190
        dzurasergij4@gmail.com <dzurasergij4@gmail.com>
191     private fun updateSumChart(feeds: List<Feed>?) {
192         val entries = mutableListOf<BarEntry>()
193         feeds?.forEachIndexed { index, feed ->
194             feed.field1?.let { it: String
195                 entries.add(BarEntry(index.toFloat(), it.toFloat()))
196             }
197         }
198
199         val dataSet = BarDataSet(entries, label: "Sum")
200         val barData = BarData(dataSet)
201         sumChart.data = barData
202         sumChart.invalidate()
203     }
204
        dzurasergij4@gmail.com <dzurasergij4@gmail.com>
205     private fun updateMedianChart(feeds: List<Feed>?) {
206         val entries = mutableListOf<BarEntry>()
207         feeds?.forEachIndexed { index, feed ->
208             feed.field1?.let { it: String
209                 entries.add(BarEntry(index.toFloat(), it.toFloat()))
210             }
211         }
212
213         val dataSet = BarDataSet(entries, label: "Median")
214         val barData = BarData(dataSet)
215         medianChart.data = barData
216         medianChart.invalidate()
217     }
218
```

Рисунок 3.8 Код оновлення графіків

Цей код відповідає за оновлення трьох різних графіків на основі отриманих даних з ThingSpeak, використовуючи середні, сумарні та медіанні значення відповідно. Функція updateAverageDataChart ітерується по отриманих даних (feeds), витягує потрібні значення (field1), формує набір даних (BarDataSet), і оновлює графік, що відображає середні значення (averageDataChart). Аналогічно, функція updateSumChart створює BarDataSet з сум значень field1 і оновлює графік sumChart. Функція updateMedianChart формує BarDataSet з медіан значень field1 та оновлює графік medianChart. Таким чином, цей код забезпечує візуалізацію різних статистичних характеристик даних на трьох окремих графіках. На рисунку 3.9 код для взаємодії з API.

```
dzurasergij4@gmail.com <dzurasergij4@gmail.com>
private fun fetchDataForSelectedDates(startDate: String, endDate: String, minutes: Int) {
    productAPI.getStartEndFeedsAverage(startDate, endDate, minutes).enqueue(object : Callback<JsonResponse> {
        override fun onResponse(call: Call<JsonResponse>, response: Response<JsonResponse>) {
            if (response.isSuccessful) {
                val data = response.body()
                updateAverageDataChart(data?.feeds)
            }
        }

        override fun onFailure(call: Call<JsonResponse>, t: Throwable) {
            // Handle failure
        }
    })
    productAPI.getFeedsSum(minutes).enqueue(object : Callback<JsonResponse> {
        override fun onResponse(call: Call<JsonResponse>, response: Response<JsonResponse>) {
            if (response.isSuccessful) {
                val data = response.body()
                updateSumChart(data?.feeds)
            }
        }

        override fun onFailure(call: Call<JsonResponse>, t: Throwable) {
            // Handle failure
        }
    })
    productAPI.getFeedsMedian(minutes).enqueue(object : Callback<JsonResponse> {
        override fun onResponse(call: Call<JsonResponse>, response: Response<JsonResponse>) {
            if (response.isSuccessful) {
                val data = response.body()
                updateMedianChart(data?.feeds)
            }
        }

        override fun onFailure(call: Call<JsonResponse>, t: Throwable) {
            // Handle failure
        }
    })
}
```

Рисунок 3.9 Код взаємодії з API

Функція fetchDataForSelectedDates отримує дані за певний період часу з ThingSpeak API і викликає функції для оновлення графіків середніх значень, сум та медіан.

− Отримання середніх значень: Викликається метод getStartEndFeedsAverage з API, щоб отримати середні значення за вказаний період.При успішному отриманні даних викликається функція updateAverageDataChart, щоб оновити графік середніх значень.

− Отримання сумарних значень: викликається метод getFeedsSum з API для отримання сум значень за вказаний період. При успішному отриманні даних викликається функція updateSumChart, щоб оновити графік сумарних значень.

− Отримання медіанних значень: викликається метод getFeedsMedian з API для отримання медіан значень за вказаний період. При успішному отриманні даних викликається функція updateMedianChart, щоб оновити графік медіанних значень.

На рисунку 3.10 представлений екран зі статистикою, де можна побачити інтегрований календар, що дозволяє обирати конкретні дати для відображення отриманих даних із сервера. Це дуже зручно для аналізу даних за певний період. Після вибору дати, система надає можливість переглядати дані у вигляді діаграм, що допомагає візуалізувати та аналізувати інформацію з IoT-пристроїв. Використання календаря спрощує процес вибору потрібного періоду, а відображення даних у вигляді діаграм забезпечує їх наочність і полегшує процес прийняття рішень на основі зібраної інформації.

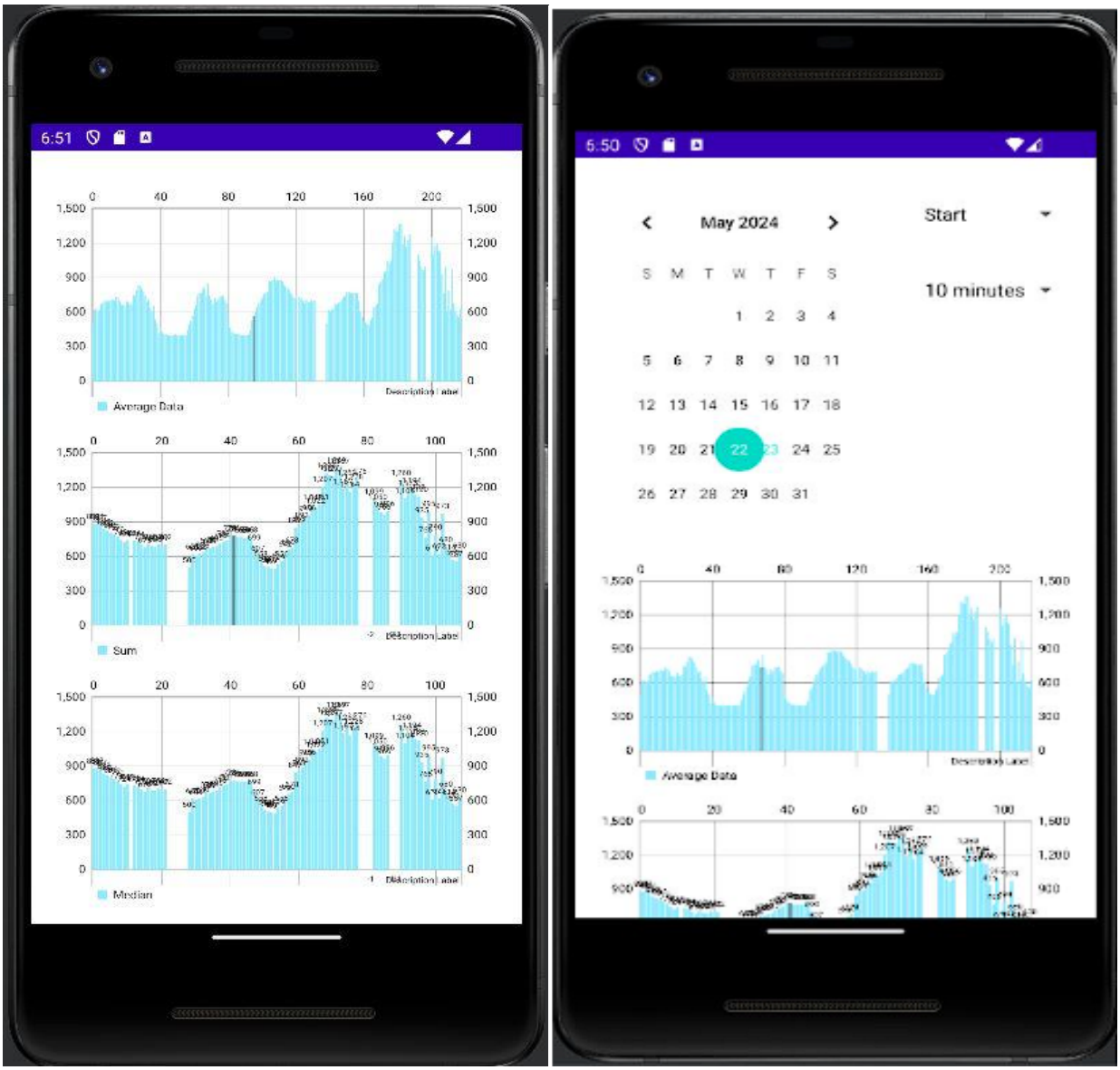

Рисунок 3.10 – Статистика за вибраний користувачем період

На рис 3.10 під кожним графіком рівнів CO_2 є додаткові рядки з числовими значеннями, які, є:

– Середнє - середнє арифметичне значення всіх даних за цей період.

– Сума - сума всіх значень даних за вказаний період.

– Медіана - середнє значення у впорядкованому наборі даних, коли однакова кількість значень менша і більша цього значення.

Ці статистичні показники дозволяють швидко оцінити загальний рівень та розподіл концентрацій CO_2 протягом кожного періоду, не вдаючись до аналізу всіх точок даних графіка також є календар вгорі для вибору початкової та кінцевої дати, що дозволяє переглядати ці статистичні дані разом із графіком для кожного дня окремо. Таким чином, ці зображення поєднують графічну

візуалізацію даних рівнів CO_2 із ключовими числовими статистичними показниками - медіаною, сумою та середнім, що допомагає узагальнювати дані вимірювань. На рисунку 3.11 зображена візуалізація даних вимірювань пристроїв IoT на карті

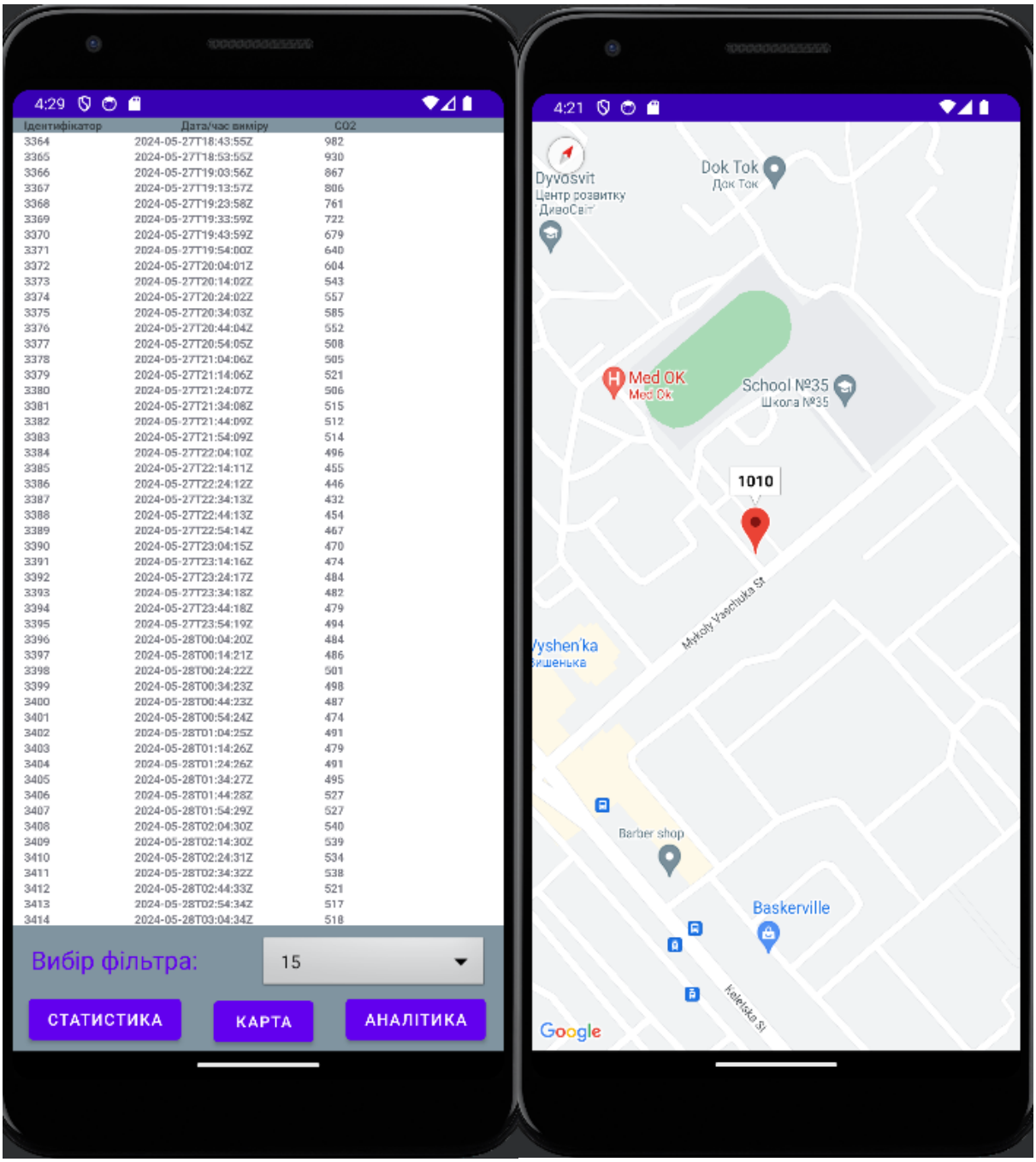

Рисунок 3.11 – Візуалізація даних вимірювань пристроїв IoT на карті

З метою покращення аналізу даних і надання користувачам можливості візуалізації географічних даних у застосунок було додано функціонал для відображення даних на карті рисунок 3.11. Це дозволяє користувачам аналізувати вимірювання в контексті їхнього географічного розташування, що є

особливо важливим для завдань, які вимагають розуміння просторових закономірностей і взаємозв'язків [18].

Просторовий аналіз даних відкриває нові можливості для візуалізації та інтерпретації даних, забезпечуючи користувачів додатковими інструментами . Наприклад, вимірювання рівнів забруднення повітря, температури, вологості тощо можуть бути значно ефективніше проаналізовані, якщо їх представити у вигляді картографічних даних.

Для реалізації цього функціоналу було використано бібліотеку Maps SDK для Android. Ця бібліотека надає широкий спектр інструментів для роботи з картами і дозволяє легко інтегрувати їх у застосунок.

Процес використання цього функціоналу починається з відкриття екрану з картою. Користувач вибирає опцію "Карта" у головному меню застосунку, після чого відкривається новий екран, на якому відображається інтерактивна карта. На цій карті користувач може побачити своє поточне місцезнаходження, що допомагає орієнтуватися та надає відправну точку для подальшого аналізу даних.

Далі, застосунок надсилає запит до сервера для отримання даних вимірювань, які мають просторову прив'язку, тобто координати широти та довготи. Ці дані можуть бути отримані у форматі JSON, який легко обробляється у середовищі Android. Важливо, що запити до API здійснюються асинхронно, що забезпечує плавну роботу застосунку та не блокує основний інтерфейс користувача під час очікування відповіді від сервера.

Після отримання даних застосунок додає на карту маркери, які відповідають кожному вимірюванню. Кожен маркер позиціонується на карті відповідно до координат, наданих у даних вимірювань. Маркери можуть мати різні кольори або іконки, що дозволяє користувачам легко ідентифікувати різні типи даних, такі як рівні забруднення повітря, температури, вологості тощо.

Інтерактивність є важливою частиною цього функціоналу. Користувач може взаємодіяти з картою, змінювати масштаб, переміщуватися по карті, щоб оглянути різні ділянки. При натисканні на маркер користувач може переглядати додаткову інформацію про відповідне вимірювання, таку як точний час

вимірювання, точні показники та інші релевантні дані. Крім того, можливість додавання нових маркерів у режимі реального часу дозволяє користувачам бачити найсвіжіші дані, що надходять від сервера, і таким чином отримувати актуальну інформацію про ситуацію на місцях [19].

Цей функціонал значно розширює можливості користувачів щодо просторового аналізу даних, забезпечуючи більш глибоке розуміння та зручну інтерпретацію інформації у географічному контексті. Репозиторій з проектом розміщено на GitHub [20].

3.4 Висновки

В даному розділі було розроблено мобільний додаток який дозволяє покращити процес аналізу та візуалізації даних отриманих з IoT-систем. Розроблено UML діаграми. Описано алгоритм роботи мобільного додатку та здійснено його програмну реалізацію в Android Studio з акцентом на візуалізацію даних у вигляді графіків, діаграм та відображенням на інтерактивній карті, що надає користувачам зручні інструменти для аналізу даних в географічному контексті, а також забезпечує функції фільтрації, вибору періодів та офлайн-доступ для ефективного моніторингу IoT даних.

ВИСНОВКИ

В ході бакалаврської роботи було визначено основні переваги та недоліки впровадження IoT-системи на Android-пристроях, проаналізовано існуючі методи та інформаційні технології, а також надано опис об’єкта мого дослідження.

Проаналізувавши існуючі методи та інформаційні технології, було виявлено найбільш ефективні підходи до збирання, оброблення та візуалізації даних.

Здійснено опис та обґрунтування вибору оптимальних засобів та технологій для вирішення поставленої задачі. А саме, обґрунтовано вибір необхідного фреймворку для розробки мобільних додатків Android, а також було розглянуто важливість використання хмарних платформ для ефективного зберігання та обробки даних. Зокрема, було обрано Android Studio як інтегроване середовище розробки (IDE) завдяки його потужним інструментам для розробки, тестування та деплою Android-додатків. Щодо хмарних платформ, було здійснено аналіз кількох провідних рішень на ринку, таких як Google Cloud IoT Core, Microsoft Azure IoT Hub, AWS IoT Core та ThingSpeak. Також було розроблено клієнтське API для взаємодії з ThingSpeak.

Розроблено Android-додаток для моніторингу та візуалізації даних з IoT-пристроїв. Архітектуру додатку представлено за допомогою UML-діаграм, які візуалізують функціональні вимоги та структуру модулів. Описано алгоритм роботи додатку та здійснено програмну реалізацію в Android Studio, з акцентом на візуалізацію даних у вигляді графіків, діаграм та інтерактивної карти. Додаток надає користувачам зручні інструменти для аналізу даних у географічному контексті, а також забезпечує функції фільтрації, вибору періодів та офлайн-доступ для ефективного моніторингу IoT-даних.

. Результати роботи були представлені на Міжнародній науково-методичній інтернет-конференції «Проблеми вищої математичної освіти: виклики сучасності (Вінниця, 2024р)», де були опубліковані відповідні тези.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Горячев Г.В., Джура С.В. Аналіз та візуалізація даних моніторингу IoT-систем на базі Android-пристроїв. *Міжнародна науково-методична інтернет-конференція «Проблеми вищої математичної освіти: виклики сучасності (Вінниця, 2024 р)». URL*: https://conferences.vntu.edu.ua/index.php/pmovc/pmovc24/paper/view/21809

2. Афанасьєв Ю. В. Аналіз розвитку IoT систем в складних організаційно-технічних системах. *Інформаційно-комунікаційні технології та кібербезпека (ІКТК-2023)* : матеріали дев'ятої Міжнародної науково-технічної конференції, м. Харків, 7 грудня 2023 р. Харків : ХНУРЕ, 2023. С. 93–95.

3. Tripathy B., Anuradha J. Internet of Things (IoT): TeChnologies, AppliCations, Challenges and Solutions. Florida : CRC Press, 2022. 334 p.

4. Developer Documentation SmartThing [Електронний ресурс].URL: https://developer.smartthings.com/docs/

5. Documentation - Home Assistant [Електронний ресурс]. URL: https://www.home-assistant.io/docs/

6. MathWorks Read Data from ThingSpeak Channel. [Електронний ресурс] URL:https://www.mathworks.com/help/thingspeak/readdata.html

7. Advantages and Disadvantages of IoT - GeeksforGeeks [Електронний ресурс]. URL:https://www.geeksforgeeks.org/advantages-and-disadvantages-of-iot/

8. Ткаченко В.В. Кросплатформна розробка мобільних додатків: огляд фреймворків та інструментів Вісник Національного університету "Львівська політехніка". Серія: Інформаційні системи та мережі, 2023. № 954. С. 124-132.

9. Марченко О. В. Розробка мобільних додатків для платформ iOS та Android: порівняльний аналіз. *Наукові записки Тернопільського національного педагогічного університету імені Володимира Гнатюка. Серія: Комп'ютерні технології*. 2020. № 1(4). С. 56–63.

10. AWS IoT Core Documentation [Електронний ресурс]. URL: https://docs.aws.amazon.com/iot/index.html

11. Cloud IoT Core Google Cloud [Електронний ресурс]. URL: https://cloud.google.com/iot-core

12. Azure IoT Hub Documentation Microsoft Learn [Електронний ресурс]. URL: https://learn.microsoft.com/en-us/azure/iot-hub/

13. IoT Analytics - ThingSpeak Internet of Things [Електронний ресурс]. URL: https://thingspeak.com/

14. WeeklyCoding. MPAndroidChart Documentation [Електронний ресурс].URL:MPAndroidChart Documentation - Weeklycoding

15. Безкоштовний онлайн-інструмент для створення діаграм та схем [Електронний ресурс].URL: https://app.diagrams.net/

16. Smyth N. Android Studio Arctic Fox Essentials - Java Edition: Developing Android Apps Using Android Studio 2020.31 and Java: textbook. eBookFrenzy, 2021. 778 p.

17. Горячев Г.В., Джура С.В., Караваєв В.О., Литвинюк О.С., Тарасовський Т.С. Android додаток для візуалізації та аналізу даних отриманих з IoT пристроїв. *Міжнародна науково-практична інтернет-конференція «Молодь в науці: дослідження, проблеми, перспективи (Вінниця, 2023-2024 рр)».* URL: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2024/paper/view/20989

18. Domínguez-Bolaño, T., Campos, O., Barral, V., Escudero, C. J.García-Naya, J. A. An overview of IOT architectures, technologies, and existing open-source projects. *Internet of Things*, 2022, v. 20, p. 100626–100648: DOI: https://doi.org/10.1016/j.iot.2022.100626

19. Paolone, G., Iachetti, D., Paesani, R., Pilotti, F., Marinelli, M.Di Felice, P. A holistic overview of the internet of things ecosystem. IoT. DOI:https://doi.org/10.3390/iot3040022

20. GitHub. (2023). GitHub Docs. [Електроний ресурс]URL: https://docs.github.com/

Додаток А

(обов’язковий)

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації

ЗАТВЕРДЖУЮ
Завідувач кафедри САІТ
_____ д.т.н., проф. Віталій МОКІН
“____”_____ 2024 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на бакалаврську дипломну роботу

АНАЛІЗ ТА ВІЗУАЛІАЦІЯ ДАНИХ МОНІТОРИНГОВИХ ІОТ–СИСТЕМ НА БАЗІ МОБІЛЬНИХ ANDROID–ПРИСТРОЇВ

08-34.БДР.004.00.000 ТЗ

Керівник: к.т.н., доц.
_____ Олексій КОЗАЧКО
«__» _____ 2024 р.
Розробив студент гр. СА-20б
_____ Сергій ДЖУРА
«__» _____ 2024 р.

Вінниця 2024

1. Підстава для проведення робіт.

Підставою для виконання роботи є наказ №__ по ВНТУ від «__» _____2024р., та індивідуальне завдання на БДР, затверджене протоколом №__ засідання кафедри САІТ від «__» _____ 2024р.

2. Джерела розробки:

1) Smyth N. Android Studio Arctic Fox Essentials - Java Edition: Developing Android Apps Using Android Studio 2020.31 and Java: textbook. eBookFrenzy, 2021. 778 p.

3. Мета і призначення роботи.

Метою дослідження є покращення мобільних засобів аналізу та візуалізації даних вимірювань отриманих із пристроїв IoT.

4. Вихідні дані для проведення робіт:

Дані вимірювань CO_2

https://api.thingspeak.com/channels/2418631/feeds.json?api_key=VCDRU86G0QQWDHRR

5. Методи дослідження:

Об'єктно-орієнтоване програмування, функціональне програмування. Використання платформи Thing Speak та Github.

6. Етапи роботи і терміни їх виконання:

a) Аналіз предметної області _____ – _____
b) Загальна характеристика об'єкту досліджень _____ – _____
c) Вибір оптимальних інформаційних технологій _____ – _____
d) Результати дослідження та аналіз і візуалізація даних _____ – _____
e) Оформлення матеріалів до захисту БДР _____ – _____

7. Очікувані результати та порядок реалізації

Виконано покращення мобільних засобів аналізу та візуалізації даних вимірювань отриманих із пристроїв IoT.

8. Вимоги до розробленої документації

Текстова та ілюстративна частини роботи оформлені у відповідності до вимог «Методичних вказівок до виконання бакалаврських кваліфікаційних робіт для студентів спеціальностей: 124 «Системний аналіз», 126 «Інформаційні системи та технології» (освітня програма «Прикладні інформаційні технології»)».

9. Порядок приймання роботи

Публічний захист «___» _____ 2024 р.
Початок розробки «___» _____ 2024 р.
Граничні терміни виконання БДР «___» _____ 2024 р.

Розробив студент групи СА-20б _____ Сергій ДЖУРА

Додаток Б

(обов'язковий)

Протокол перевірки бакалаврської дипломної роботи на наявність текстових запозичень

Назва роботи: «Аналіз та візуалізація даних моніторингових IoT – систем на базі мобільних Android – пристроїв»
Тип роботи: бакалаврська дипломна робота
Підрозділ: кафедра САІТ

Показники звіту подібності Unicheck

Оригінальність 97,07% Схожість 2,93%

Аналіз звіту подібності (відмітити потрібне)

- Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.
- Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і самостійності її автора. Роботу направити на розгляд експертної комісії кафедри.
- Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ (підпис) Сергій ЖУКОВ

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ (підпис) Сергій ДЖУРА

Керівник роботи _____ (підпис) Олексій КОЗАЧКО

Додаток В

(довідниковий)

Фрагмент лістингу програми

Код класу MainActiviti

```
package ua.edu.vntu

import android.content.Intent
import android.os.Bundle
import android.util.Log
import android.view.View
import android.widget.AdapterView
import android.widget.ArrayAdapter
import android.widget.Button
import android.widget.Spinner
import android.widget.TextView
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import retrofit2.Call
import retrofit2.Callback
import retrofit2.Response
import ua.edu.vntu.R.array
import ua.edu.vntu.R.id
import ua.edu.vntu.R.layout
import ua.edu.vntu.model.FeedAPI
import ua.edu.vntu.model.JsonResponse
import ua.edu.vntu.view.CustomAdapter

class MainActivity : AppCompatActivity(), Callback<JsonResponse> {
    lateinit var rvMain: RecyclerView
    lateinit var sp: Spinner
    lateinit var myAdapter: CustomAdapter
    val dbHelper = DatabaseHelper(this)
    var positionSelected: Int = 0;
    var restApi: RestApi = RestApi()
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        val activityMain = this
//        dbHelper.addRow(Feed(1,"20.12.2023", "23.4"))


        val graphButton = findViewById<Button>(R.id.btnGraph)
        graphButton.setOnClickListener {
            val intent = Intent(this, GraphActivity::class.java)
           startActivity(intent)
        }

        val btnStatistic = findViewById<Button>(R.id.btnStatistic)
        btnStatistic.setOnClickListener {
```

```
        val intent = Intent(this, GrafstatisticActivity::class.java)
        intent.putExtra("position", activityMain.positionSelected)
        startActivity(intent)
        //val intent = Intent(this, StatisticActivity::class.java)
        //          intent.putExtra("position", activityMain.positionSelected)
        //          startActivity(intent)
    }
    val btnMap = findViewById<Button>(R.id.mapActivityBtn)
    btnMap.setOnClickListener {
        val intent = Intent(this, MapActivity::class.java)
        startActivity(intent)
    }

    rvMain = findViewById(R.id.recycler_view)

    rvMain.layoutManager = LinearLayoutManager(this)
    /////
    val retrofit = restApi.retrofit()
    val productAPI = retrofit.create(FeedAPI::class.java)
    //.create(ProductAPI::class.java)
    val call = productAPI.getAllFeeds()
    call.enqueue(this)


    val filters = resources.getStringArray(array.filters)
    sp = findViewById(id.spinner1)
    val adapter = ArrayAdapter(
        this,
        layout.drop_down_item, filters
    )
    sp.adapter = adapter
    sp.onItemSelectedListener = object :
        AdapterView.OnItemSelectedListener {
        override fun onItemSelected(
            parent: AdapterView<*>,
            view: View, position: Int, id: Long
        ) {
            activityMain.positionSelected = position
//                Toast.makeText(this@MainActivity,
//                    "вибраний фільтр:" + " " +
//                        "" + filters[position], Toast.LENGTH_SHORT).show()
            if (position == 0) {
                val call = productAPI.getDayFeeds()
                call.enqueue(this@MainActivity)
            } else if (position == 1) {
                val call = productAPI.getMonthFeeds()
                call.enqueue(this@MainActivity)
            } else if (position == 2) {
                val call = productAPI.getQuarterFeeds()
                call.enqueue(this@MainActivity)
            } else if (position == 3) {
                val call = productAPI.getAllFeeds()
                call.enqueue(this@MainActivity)
            }

        }

        override fun onNothingSelected(parent: AdapterView<*>) {
            // write code to perform some action
        }
```

```
        }

    }



    override fun onResume() {
        super.onResume()
        Log.d("OnResume", "")

    }

    override fun onResponse(call: Call<JsonResponse>, response: Response<JsonResponse>) {
        CoroutineScope(Dispatchers.IO).launch {
            runOnUiThread {
                val body = response.body()

                if (body == null || response.body()!!.feeds.isEmpty()) {
                    Toast.makeText(
                        this@MainActivity,
                        "Даних немає ", Toast.LENGTH_SHORT
                    ).show()
                } else {
                    var data = response.body()!!
                    // записати дані із Інтернета у БД
                    //dbHelper.insert(data.feeds)
                    // add data to DB
                    val f1 = findViewById<TextView>(id.field1_title)
                    f1.text = data.channel.field1
                    val f2 = findViewById<TextView>(id.field2_title)
                    f2.text = data.channel.field2
                    val f3 = findViewById<TextView>(id.field3_title)
                    f3.text = data.channel.field3
                    myAdapter = CustomAdapter(baseContext, data.feeds)
                    rvMain.adapter = myAdapter
                }
            }
        }
    }

    override fun onFailure(call: Call<JsonResponse>, t: Throwable) {
        val mes = t.message
        Log.i("HELP_ME", mes.toString())
    }

}
```

Код класу GraphActivity

```
import android.annotation.SuppressLint
import android.content.Intent
import android.graphics.Color
import android.graphics.DashPathEffect
import android.os.Bundle
import android.util.Log
import android.view.View
import android.widget.AdapterView
import android.widget.Spinner
import android.widget.Toast
```

```
import android.widget.Button
import androidx.appcompat.app.AppCompatActivity
import androidx.core.content.ContextCompat
import com.github.mikephil.charting.charts.BarChart
import com.github.mikephil.charting.charts.LineChart
import com.github.mikephil.charting.components.XAxis
import com.github.mikephil.charting.components.YAxis
import com.github.mikephil.charting.data.BarData
import com.github.mikephil.charting.data.BarEntry
import com.github.mikephil.charting.data.BarDataSet
import com.github.mikephil.charting.data.Entry
import com.github.mikephil.charting.data.LineData
import com.github.mikephil.charting.data.LineDataSet
import com.github.mikephil.charting.formatter.IndexAxisValueFormatter
import com.github.mikephil.charting.formatter.ValueFormatter
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import retrofit2.Call
import retrofit2.Callback
import retrofit2.Response
import ua.edu.vntu.model.FeedAPI
import ua.edu.vntu.model.JsonResponse
import java.text.DecimalFormat
import java.text.SimpleDateFormat
import java.util.Date
import java.util.LinkedList
import java.util.Locale
import java.util.concurrent.TimeUnit

class GraphActivity : AppCompatActivity(), Callback<JsonResponse> {
    private lateinit var barChart: BarChart
    private lateinit var lineChart1: LineChart
    private lateinit var lineChart2: LineChart
    private lateinit var lineChart3: LineChart
    private lateinit var graphTypeSpinner: Spinner
    private var restApi: RestApi = RestApi()

    override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContentView(R.layout.activity_graph)
      val graphStatus = findViewById<Button>(R.id.btnStatus)
      graphStatus.setOnClickListener {
        val intent = Intent(this, StatusActivity::class.java)
        startActivity(intent)
      }
      barChart = findViewById(R.id.barChart)
      lineChart1 = findViewById(R.id.chart)
      lineChart2 = findViewById(R.id.chart2)
      lineChart3 = findViewById(R.id.chart3)
      graphTypeSpinner = findViewById(R.id.graph_type_spinner)
      graphTypeSpinner.onItemSelectedListener = object : AdapterView.OnItemSelectedListener {
        override fun onItemSelected(parent: AdapterView<*>, view: View?, position: Int, id: Long) {
          when (position) {
            0 -> showLineCharts()
            1 -> showBarChart()
```

```
        }
      }

      override fun onNothingSelected(parent: AdapterView<*>) {}
    }

    val retrofit = restApi.retrofit()
    val productAPI = retrofit.create(FeedAPI::class.java)
    val call = productAPI.getAllFeeds()
    call.enqueue(this)

    val btnStatus = findViewById<Button>(R.id.btnStatus)
}
private fun showLineCharts() {
    barChart.visibility = View.INVISIBLE
    lineChart1.visibility = View.VISIBLE
    lineChart2.visibility = View.VISIBLE
    lineChart3.visibility = View.VISIBLE
}

private fun showBarChart() {
    barChart.visibility = View.VISIBLE
    lineChart1.visibility = View.INVISIBLE
    lineChart2.visibility = View.INVISIBLE
    lineChart3.visibility = View.INVISIBLE
}

private fun buildBarChart(dataObject: JsonResponse) {
    val barEntries = ArrayList<BarEntry>()
    for ((index, feed) in dataObject.feeds.withIndex()) {
      if (feed.field1 != null) {
        barEntries.add(BarEntry(index.toFloat(), feed.field1.toFloat()))
      }
    }
    val barDataSet = BarDataSet(barEntries, dataObject.channel.field1)
    barDataSet.color = ContextCompat.getColor(this, R.color.colorPrimary)
    barDataSet.valueTextSize = 5f
    val barData = BarData(barDataSet)
    barChart.data = barData
    setupBarChart(barChart)
    barChart.invalidate()
}

private fun setupBarChart(barChart: BarChart) {
    barChart.description.isEnabled = false
    barChart.setNoDataText("No data")
    barChart.legend.isEnabled = true

    val xAxis = barChart.xAxis
    xAxis.setDrawGridLines(false)
    xAxis.position = XAxis.XAxisPosition.BOTTOM
    xAxis.valueFormatter = IndexAxisValueFormatter()

    val leftAxis = barChart.axisLeft
    leftAxis.setDrawGridLines(true)
    leftAxis.valueFormatter = LargeValueFormatter()
```

```
    val rightAxis = barChart.axisRight
    rightAxis.setDrawGridLines(false)
    rightAxis.valueFormatter = LargeValueFormatter()

    // Збільшення відстані між стовпчиками
    barChart.barData.barWidth = 0.3f
}

private fun buildLineCharts(dataObject: JsonResponse) {
    val f1 = LinkedList<Pair<Float, Float>>()
    val f2 = LinkedList<Pair<Float, Float>>()
    val f3 = LinkedList<Pair<Float, Float>>()

    for (feed in dataObject.feeds) {
        if (feed.field1 != null)
            f1.add(Pair(convertStringDateToFloat(feed.created_at), feed.field1.toFloat()))
        if (feed.field2 != null)
            f2.add(Pair(convertStringDateToFloat(feed.created_at), feed.field2.toFloat()))
        if (feed.field3 != null)
            f3.add(Pair(convertStringDateToFloat(feed.created_at), feed.field3.toFloat()))
    }

    val dataSet1 = lineDataSet(f1, dataObject.channel.field1)
    val dataSet2 = lineDataSet(f2, dataObject.channel.field2)
    val dataSet3 = lineDataSet(f3, dataObject.channel.field3)

    val data1 = LineData(dataSet1)
    val data2 = LineData(dataSet2)
    val data3 = LineData(dataSet3)

    lineChart1.data = data1
    lineChart2.data = data2
    lineChart3.data = data3

    setupLineChart(lineChart1)
    setupLineChart(lineChart2)
    setupLineChart(lineChart3)

    lineChart1.invalidate()
    lineChart2.invalidate()
    lineChart3.invalidate()
}

private fun setupLineChart(lineChart: LineChart) {
    lineChart.description.isEnabled = false
    lineChart.setNoDataText("Немає даних")
    lineChart.legend.isEnabled = true

    val xAxis = lineChart.xAxis
    xAxis.setDrawGridLines(false)
    xAxis.valueFormatter = DateAxisValueFormatter()

    val leftAxis = lineChart.axisLeft
    leftAxis.setDrawGridLines(true)
    leftAxis.valueFormatter = LargeValueFormatter()
```

```
    val rightAxis = lineChart.axisRight
    rightAxis.setDrawGridLines(false)
    rightAxis.valueFormatter = LargeValueFormatter()
}

private fun lineDataSet(listData: List<Pair<Float, Float>>, label: String): LineDataSet {
    val entryList: List<Entry> = convertToEntries(listData)

    val dataSet = LineDataSet(entryList, label)
    dataSet.setAxisDependency(YAxis.AxisDependency.LEFT)
    dataSet.setDrawIcons(false)
    dataSet.enableDashedLine(10f, 5f, 0f)
    dataSet.enableDashedHighlightLine(10f, 5f, 0f)
    dataSet.color = Color.BLACK
    dataSet.setCircleColor(Color.BLACK)
    dataSet.lineWidth = 1f
    dataSet.circleRadius = 3f
    dataSet.setDrawCircleHole(false)
    dataSet.valueTextSize = 9f
    dataSet.setDrawFilled(true)
    dataSet.formLineWidth = 1f
    dataSet.formLineDashEffect = DashPathEffect(floatArrayOf(10f, 5f), 0f)
    dataSet.formSize = 15f
    dataSet.setDrawValues(false)
    return dataSet
}

private fun convertToEntries(listData: List<Pair<Float, Float>>): List<Entry> {
    return listData
        .filter { it.second != null }
        .map { Entry(it.first, it.second) }
}

@SuppressLint("SimpleDateFormat")
private fun convertStringDateToFloat(createdAt: String): Float {
    val simpleDateFormat = SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss'Z'")
    val date = simpleDateFormat.parse(createdAt)
    val timeInMinutes = TimeUnit.MILLISECONDS.toMinutes(date.getTime()).toFloat()
    Log.d("convertStringDateToFloat", timeInMinutes.toString())
    return timeInMinutes
}

private fun getMeasuredValue(sValue: String): Float {
    val floatPart = sValue.substring(2)
    val firstPart = sValue.substring(0, 2)
    val fromHex = floatPart.toInt(16)
    val first = firstPart.toInt(16)
    return "$first.$fromHex".toFloat()
}

override fun onResponse(call: Call<JsonResponse>, response: Response<JsonResponse>) {
    CoroutineScope(Dispatchers.IO).launch {
        runOnUiThread {
            val body = response.body()
```

```
            if (body == null || response.body()!!.feeds.isEmpty()) {
                Toast.makeText(
                    this@GraphActivity,
                    "Даних немає ",
                    Toast.LENGTH_SHORT
                ).show()
            } else {
                val data = response.body()!!
                buildBarChart(data)
                buildLineCharts(data)
            }
        }
    }

    override fun onFailure(call: Call<JsonResponse>, t: Throwable) {
        TODO("Not yet implemented")
    }
}

class IndexAxisValueFormatter : ValueFormatter() {
    override fun getFormattedValue(value: Float): String {
        return (value + 1).toInt().toString()
    }
}

class LargeValueFormatter : ValueFormatter() {
    private val mFormat = DecimalFormat("###,###,##0")

    override fun getFormattedValue(value: Float): String {
        return mFormat.format(value)
    }
}

class DateAxisValueFormatter : ValueFormatter() {
    private val mFormat = SimpleDateFormat("yyyy-MM-dd", Locale.getDefault())

    override fun getFormattedValue(value: Float): String {
        val date = Date(value.toLong() * 60 * 1000)
        return mFormat.format(date)
    }

}
```

Код класу GrafstatisticActivity

```
import android.icu.text.SimpleDateFormat
import android.os.Bundle
import android.view.View
import android.widget.AdapterView
import android.widget.ArrayAdapter
import android.widget.CalendarView
import android.widget.Spinner
import androidx.appcompat.app.AppCompatActivity
import com.github.mikephil.charting.charts.BarChart
import com.github.mikephil.charting.data.BarData
import com.github.mikephil.charting.data.BarDataSet
```

```
import com.github.mikephil.charting.data.BarEntry
import retrofit2.Call
import retrofit2.Callback
import retrofit2.Response
import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory
import ua.edu.vntu.model.Feed
import ua.edu.vntu.model.JsonResponse
import ua.edu.vntu.model.Feed2
import ua.edu.vntu.model.FeedAPI
import java.util.*
import kotlin.properties.Delegates

class GrafstatisticActivity : AppCompatActivity(), AdapterView.OnItemSelectedListener {

    private lateinit var calendarView: CalendarView
    private lateinit var calendarTypeSpinner: Spinner
    private lateinit var timeRangeSpinner: Spinner
    private lateinit var averageDataChart: BarChart
    private lateinit var sumChart: BarChart
    private lateinit var medianChart: BarChart
    private lateinit var productAPI: FeedAPI
    private var startDate: String = "2024-01-01"
    private var endDate: String = "2025-01-01"

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.graf_statistic)

        initializeViews()
        setupCalendar()
        setupSpinners()
        setupRetrofit()
    }

    private fun initializeViews() {
        calendarView = findViewById(R.id.calendar_view)
        calendarTypeSpinner = findViewById(R.id.calendar_type_spinner)
        timeRangeSpinner = findViewById(R.id.time_range_spinner)
        averageDataChart = findViewById(R.id.average_data_chart)
        sumChart = findViewById(R.id.sum_chart)
        medianChart = findViewById(R.id.median_chart)
    }

    private fun setupCalendar() {
        calendarView.setOnDateChangeListener { _, year, month, dayOfMonth ->
            val selectedDate = formatDate(year, month, dayOfMonth)
            val selectedCalendarType = calendarTypeSpinner.selectedItem.toString()

            when (selectedCalendarType) {
                getString(R.string.calendar_type_start) -> startDate = selectedDate
                getString(R.string.calendar_type_end) -> endDate = selectedDate
            }

            fetchDataForSelectedDates(startDate, endDate, getMinutes())
        }
```

```
}

private fun formatDate(year: Int, month: Int, dayOfMonth: Int): String {
    val calendar = Calendar.getInstance()
    calendar.set(year, month, dayOfMonth)
    val dateFormat = SimpleDateFormat("yyyy-MM-dd", Locale.getDefault())
    return dateFormat.format(calendar.time)
}

private fun setupSpinners() {
    val calendarTypeItems = resources.getStringArray(R.array.calendar_type_items)
    calendarTypeSpinner.adapter = ArrayAdapter(this, android.R.layout.simple_spinner_item, calendarTypeItems)
    calendarTypeSpinner.onItemSelectedListener = this

    val timeRangeItems = resources.getStringArray(R.array.time_range_items)
    timeRangeSpinner.adapter = ArrayAdapter(this, android.R.layout.simple_spinner_item, timeRangeItems)
    timeRangeSpinner.onItemSelectedListener = this
}

private fun setupRetrofit() {
    val retrofit = Retrofit.Builder()
        .baseUrl("https://api.thingspeak.com/")
        .addConverterFactory(GsonConverterFactory.create())
        .build()

    productAPI = retrofit.create(FeedAPI::class.java)
}

override fun onItemSelected(parent: AdapterView<*>?, view: View?, position: Int, id: Long) {
    when (parent?.id) {
        R.id.calendar_type_spinner -> handleCalendarTypeSelection()
        R.id.time_range_spinner -> handleTimeRangeSelection(position)
    }
}

private fun handleCalendarTypeSelection() {
    val selectedCalendarType = calendarTypeSpinner.selectedItem.toString()

    when (selectedCalendarType) {
        getString(R.string.calendar_type_start) -> {
            val calendar = Calendar.getInstance()
            calendar.time = SimpleDateFormat("yyyy-MM-dd", Locale.getDefault()).parse(startDate)
            calendarView.date = calendar.timeInMillis
        }
        getString(R.string.calendar_type_end) -> {
            val calendar = Calendar.getInstance()
            calendar.time = SimpleDateFormat("yyyy-MM-dd", Locale.getDefault()).parse(endDate)
            calendarView.date = calendar.timeInMillis
        }
    }
}

private fun handleTimeRangeSelection(position: Int) {
    fetchDataForSelectedDates(startDate, endDate, getMinutes())
}
```

```
private fun getMinutes(): Int {
    val selectedTimeRange = timeRangeSpinner.selectedItem.toString()
    val minutes = when (selectedTimeRange) {
        getString(R.string.time_range_10_minutes) -> 10
        getString(R.string.time_range_15_minutes) -> 15
        getString(R.string.time_range_20_minutes) -> 20
        getString(R.string.time_range_30_minutes) -> 30
        getString(R.string.time_range_60_minutes) -> 60
        else -> 0
    }
    return minutes
}

private fun fetchDataForSelectedDates(startDate: String, endDate: String, minutes: Int) {
    productAPI.getStartEndFeedsAverage(startDate, endDate, minutes).enqueue(object : Callback<JsonResponse> {
        override fun onResponse(call: Call<JsonResponse>, response: Response<JsonResponse>) {
            if (response.isSuccessful) {
                val data = response.body()
                updateAverageDataChart(data?.feeds)
            }
        }

        override fun onFailure(call: Call<JsonResponse>, t: Throwable) {
            // Handle failure
        }
    })
    productAPI.getFeedsSum(minutes).enqueue(object : Callback<JsonResponse> {
        override fun onResponse(call: Call<JsonResponse>, response: Response<JsonResponse>) {
            if (response.isSuccessful) {
                val data = response.body()
                updateSumChart(data?.feeds)
            }
        }

        override fun onFailure(call: Call<JsonResponse>, t: Throwable) {
            // Handle failure
        }
    })
    productAPI.getFeedsMedian(minutes).enqueue(object : Callback<JsonResponse> {
        override fun onResponse(call: Call<JsonResponse>, response: Response<JsonResponse>) {
            if (response.isSuccessful) {
                val data = response.body()
                updateMedianChart(data?.feeds)
            }
        }

        override fun onFailure(call: Call<JsonResponse>, t: Throwable) {
            // Handle failure
        }
    })
}

private fun updateAverageDataChart(feeds: List<Feed>?) {
    val entries = mutableListOf<BarEntry>()
    feeds?.forEachIndexed { index, feed ->
        feed.field1?.let {
```

```
                entries.add(BarEntry(index.toFloat(), it.toFloat()))
            }
        }
        val dataSet = BarDataSet(entries, "Average Data")
        val barData = BarData(dataSet)
        averageDataChart.data = barData
        averageDataChart.invalidate()
    }

    private fun updateSumChart(feeds: List<Feed>?) {
        val entries = mutableListOf<BarEntry>()
        feeds?.forEachIndexed { index, feed ->
            feed.field1?.let {
                entries.add(BarEntry(index.toFloat(), it.toFloat()))
            }
        }

        val dataSet = BarDataSet(entries, "Sum")
        val barData = BarData(dataSet)
        sumChart.data = barData
        sumChart.invalidate()
    }

    private fun updateMedianChart(feeds: List<Feed>?) {
        val entries = mutableListOf<BarEntry>()
        feeds?.forEachIndexed { index, feed ->
            feed.field1?.let {
                entries.add(BarEntry(index.toFloat(), it.toFloat()))
            }
        }

        val dataSet = BarDataSet(entries, "Median")
        val barData = BarData(dataSet)
        medianChart.data = barData
        medianChart.invalidate()
    }

    override fun onNothingSelected(parent: AdapterView<*>?) {
        // Handle when nothing is selected
    }
}
```

Додаток Г

(обов’язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

АНАЛІЗ ТА ВІЗУАЛІАЦІЯ ДАНИХ МОНІТОРИНГОВИХ IOT–СИСТЕМ НА БАЗІ МОБІЛЬНИХ ANDROID–ПРИСТРОЇВ

Нормоконтроль: к.т.н., доцент

_____ Сергій ЖУКОВ

«___» _____ 2024 р.

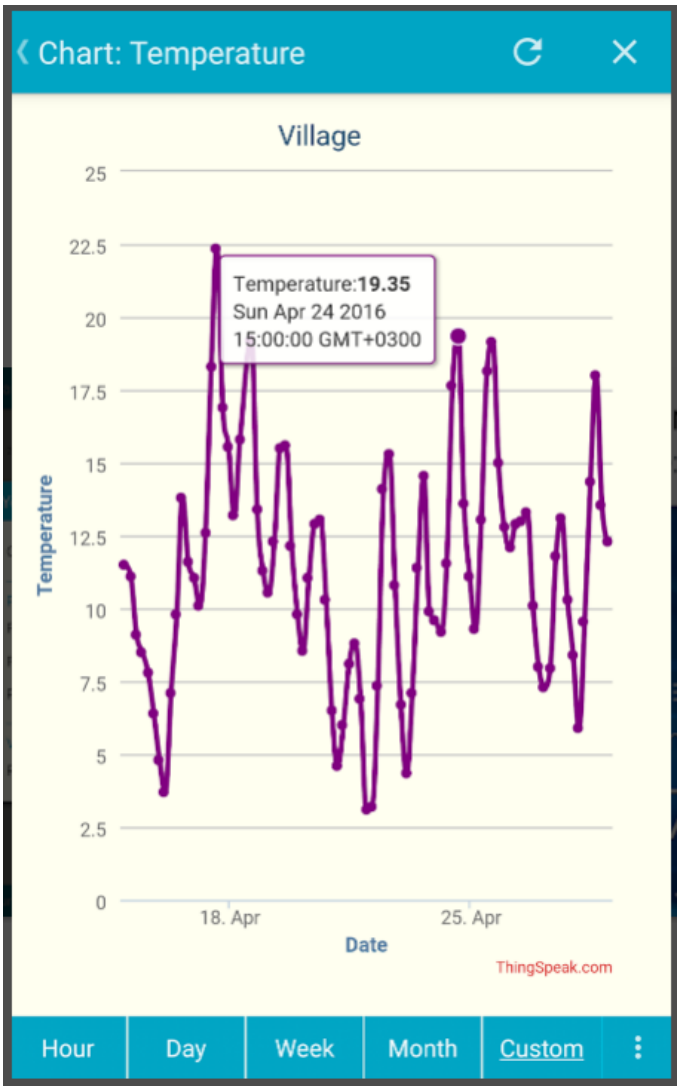

Рисунок Г.1 – Інтерфейс додатку IoT ThingSpeak Monitor Widget

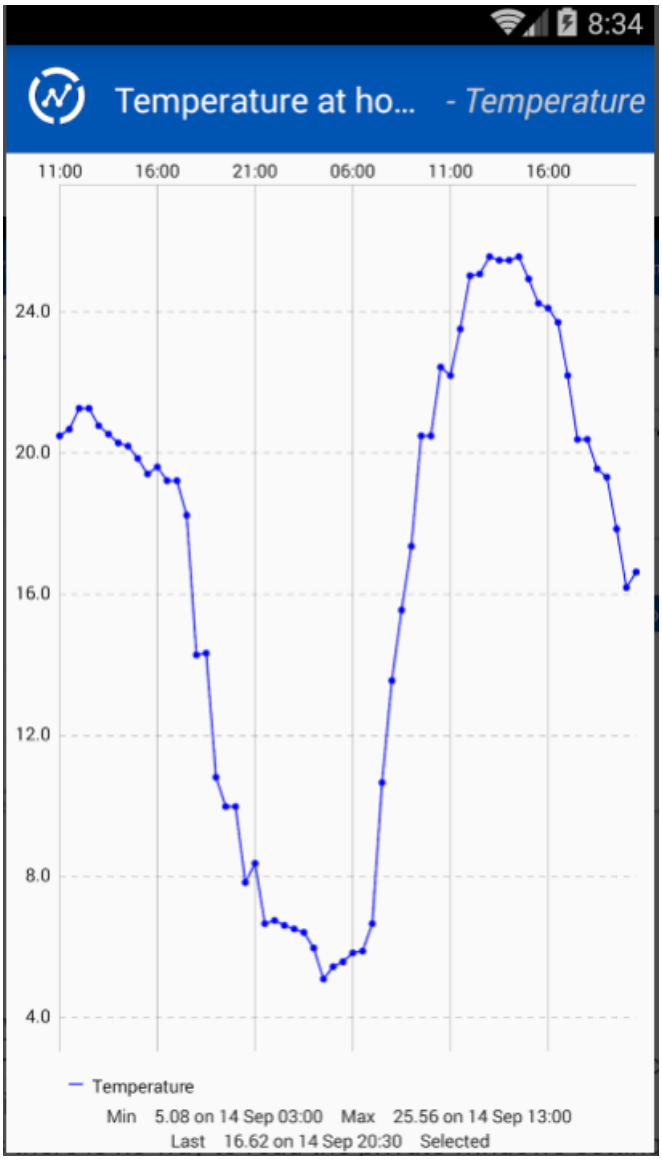

Рисунок Г.2 – Інтерфейс додатку ThingView - ThingSpeak viewer

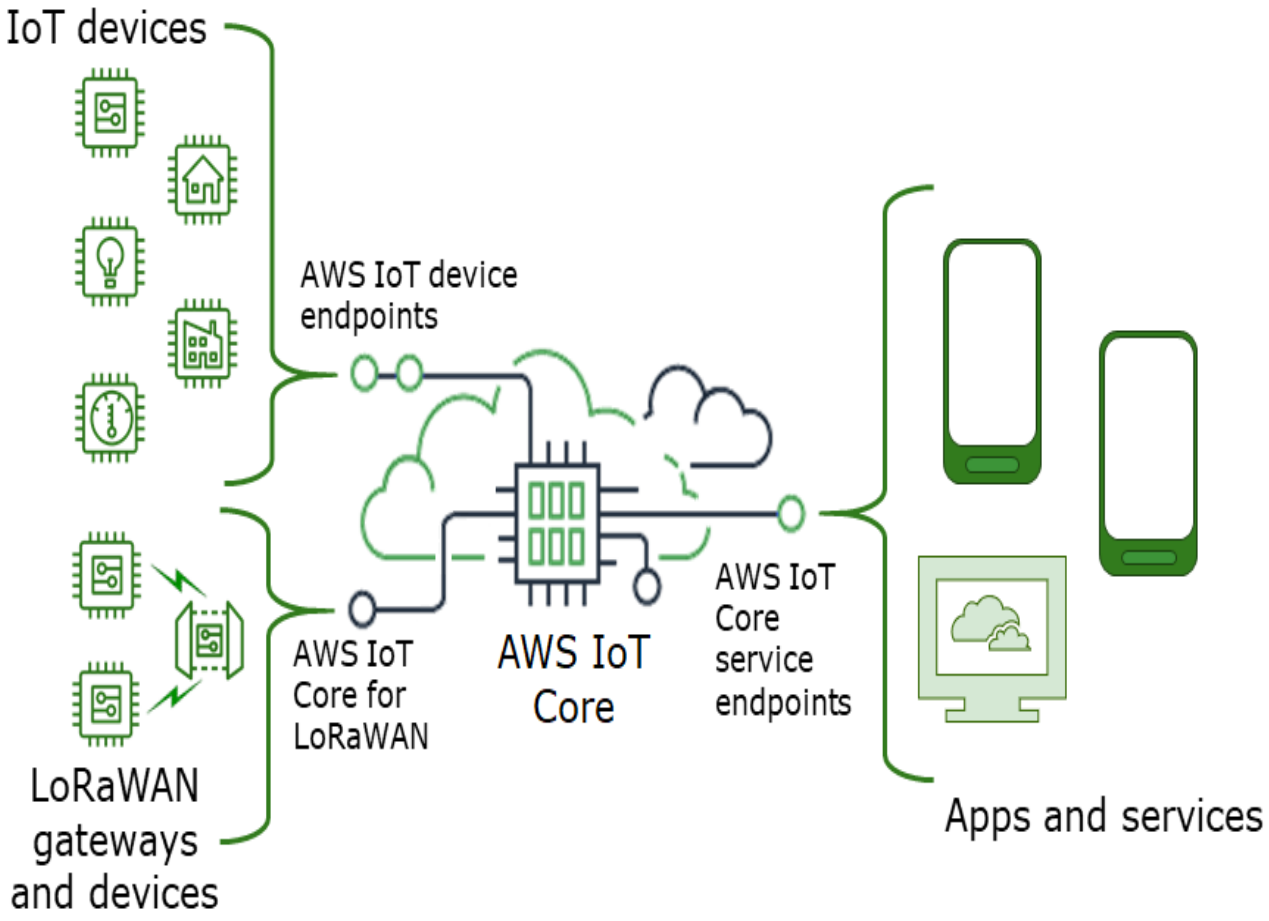

Рисунок Г.3 – Схема роботи хмарного сховища AWS IoT Core

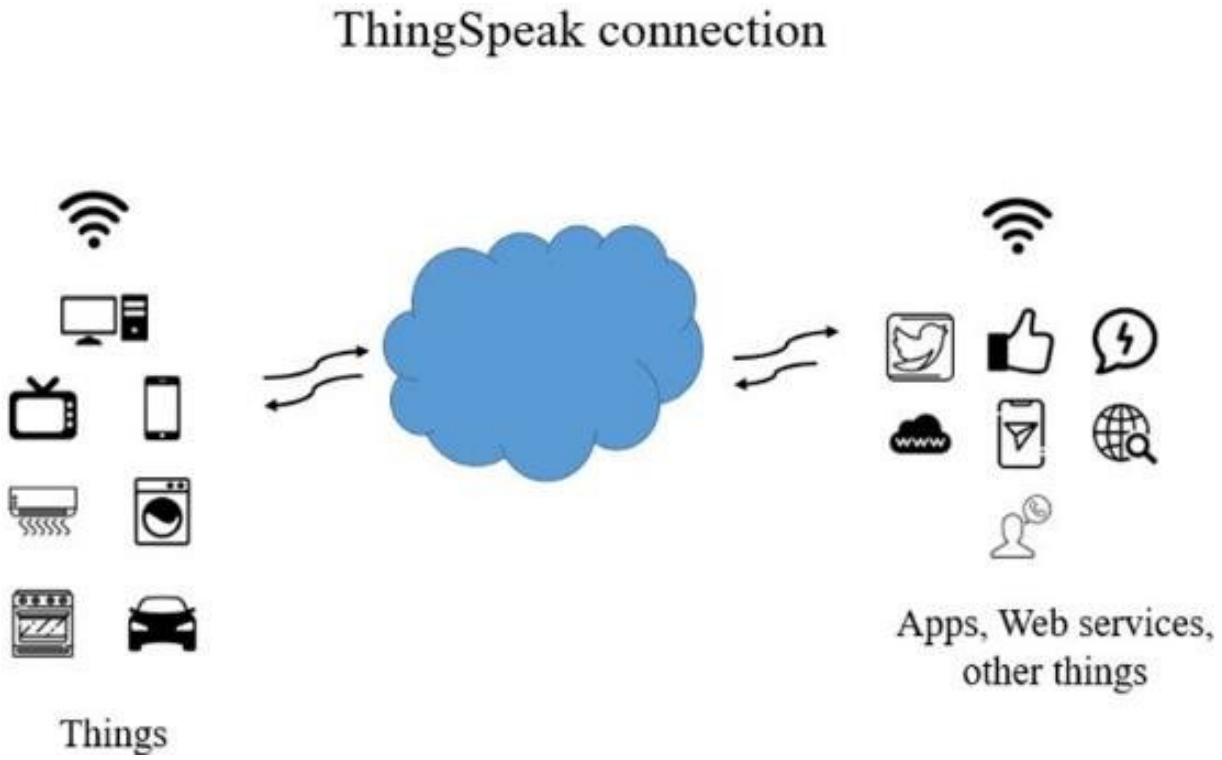

Рисунок Г.4 – Схема роботи хмарного сховища ThingSpeak

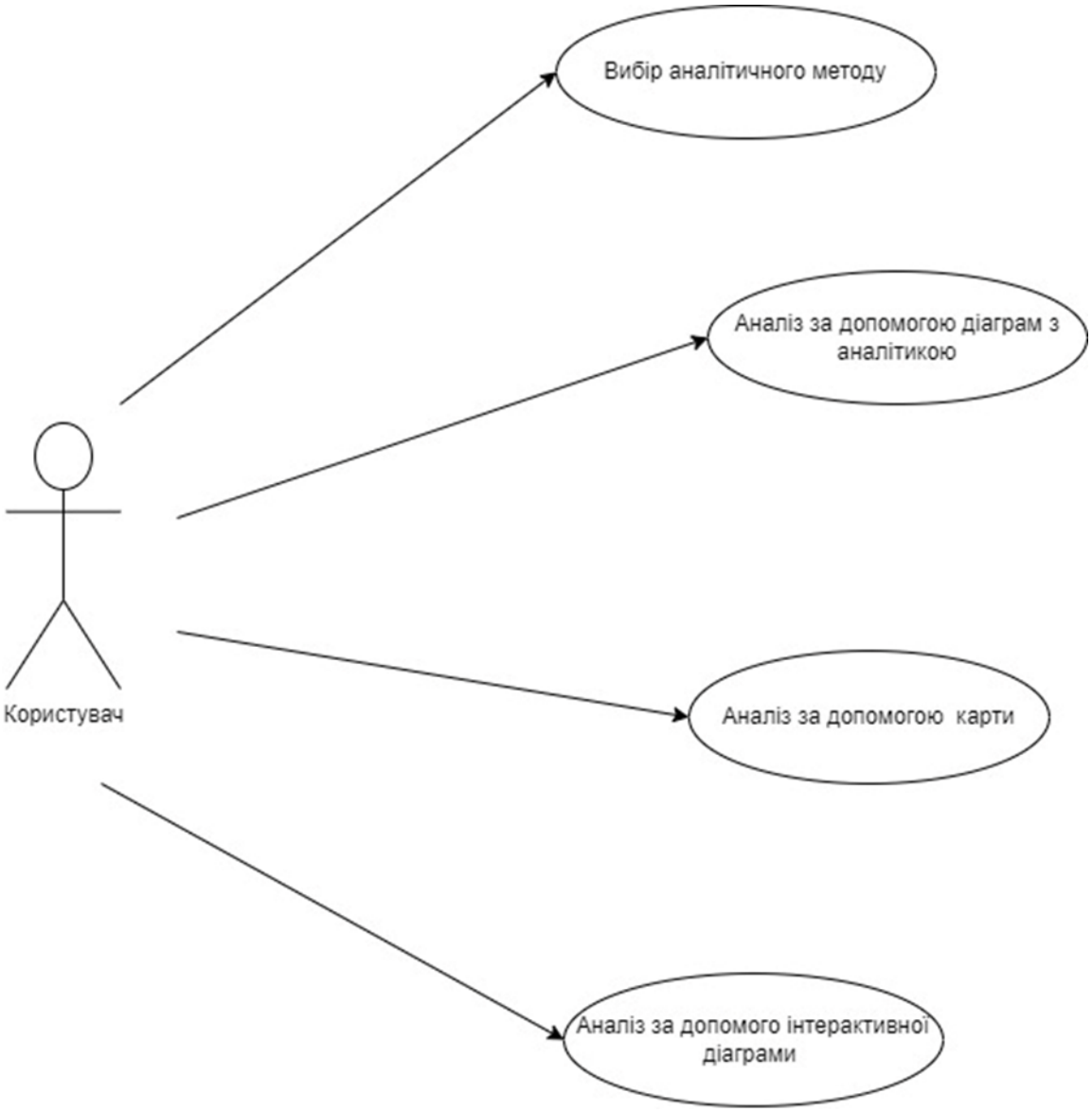

Рисунок Г.5 – Діаграма USE CASE андроїд застосунку

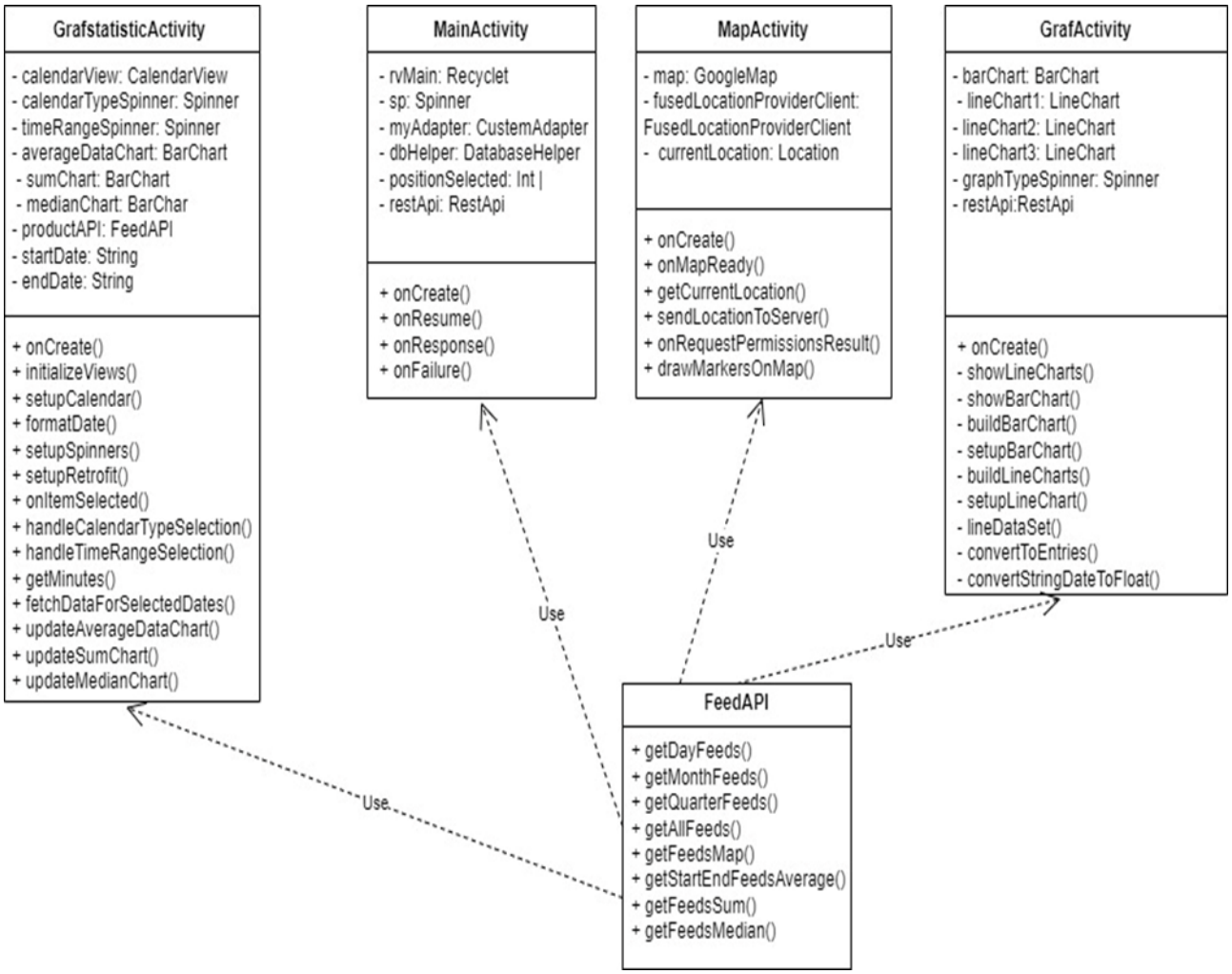

Рисунок Г.6 – Діаграма класів андроїд застосунку

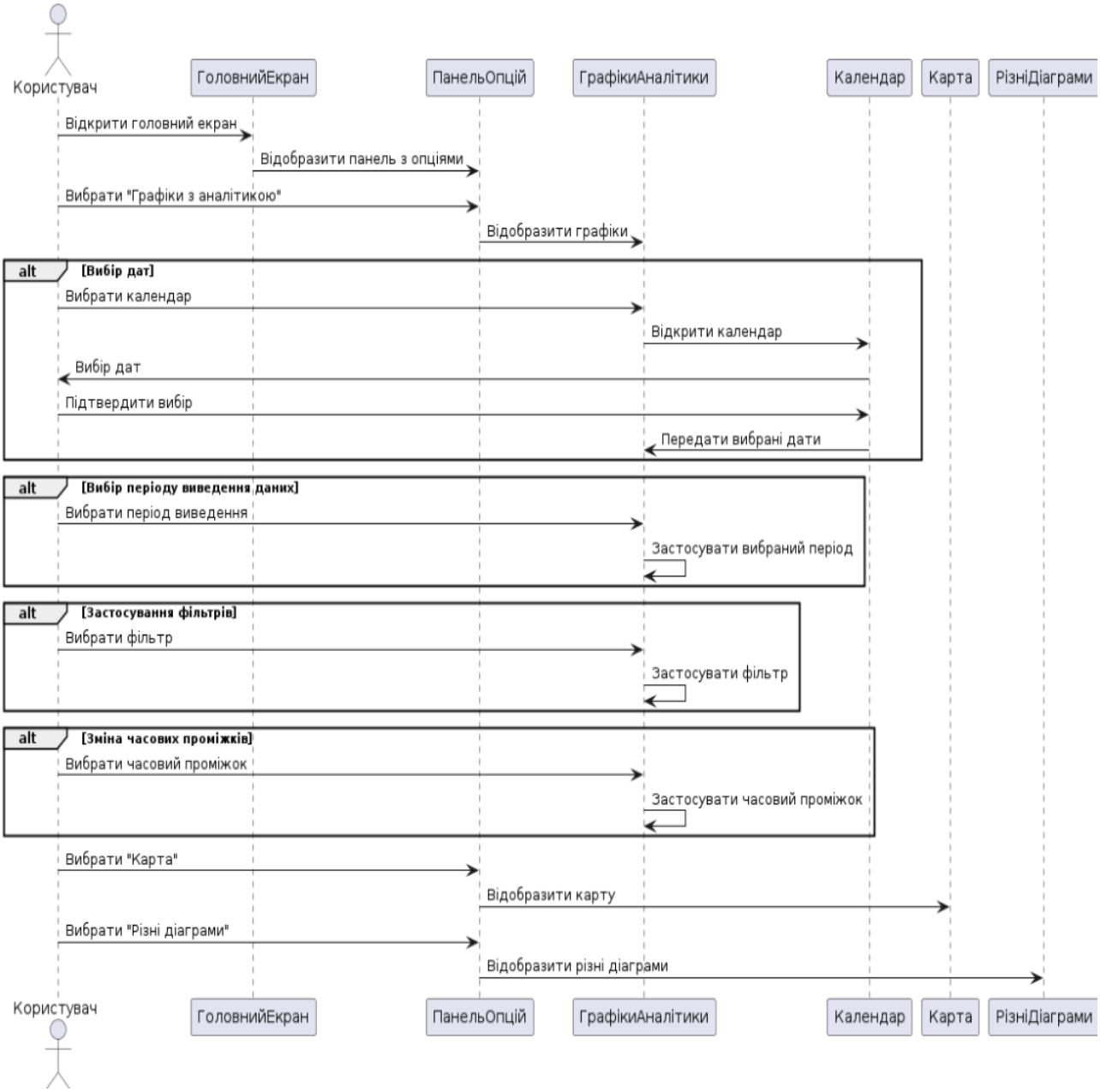

Рисунок Г.7 – Діаграма взаємодії андроїд застосунку

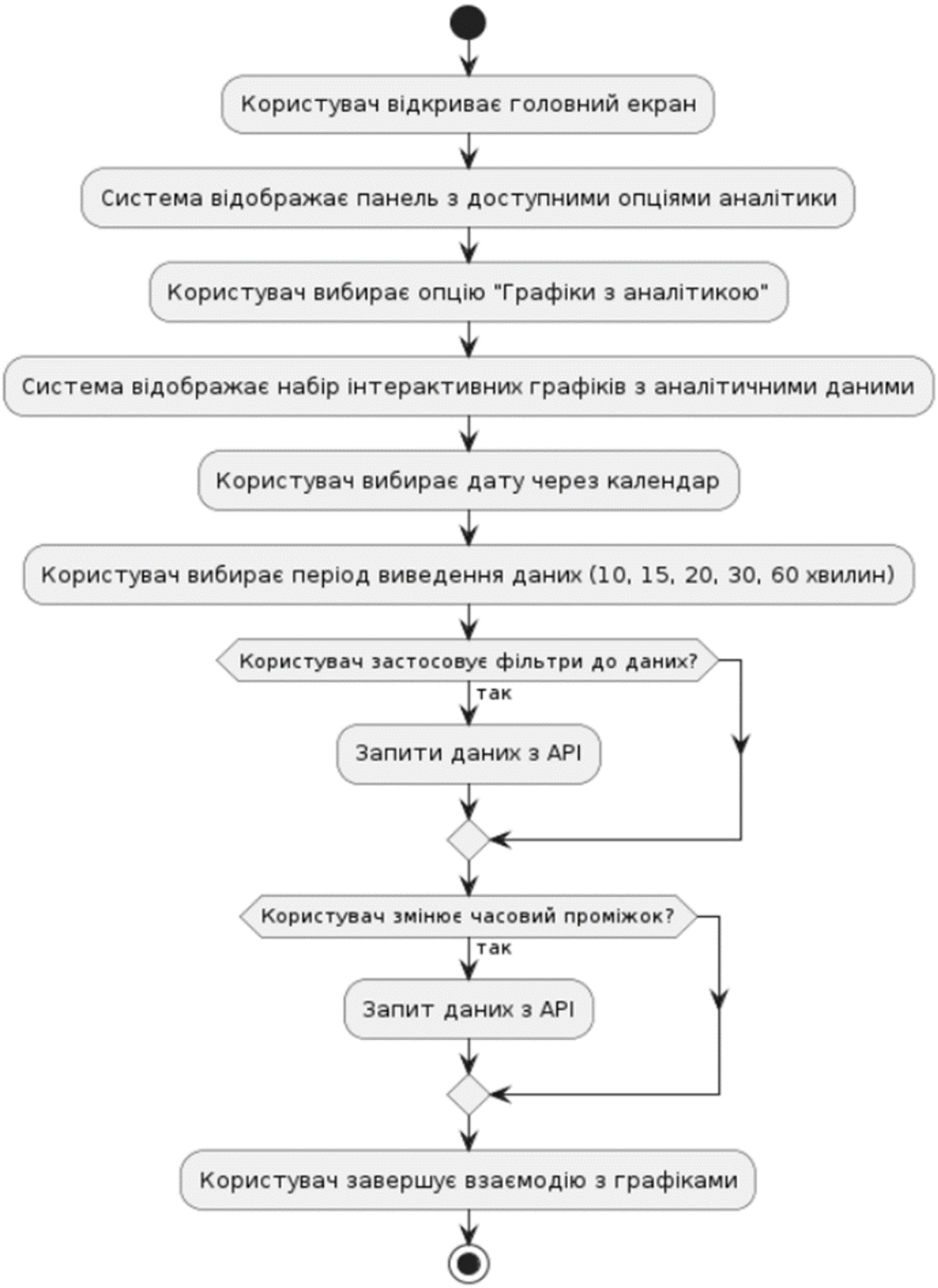

Рисунок Г.4 – Діаграма діяльності андроїд застосунку

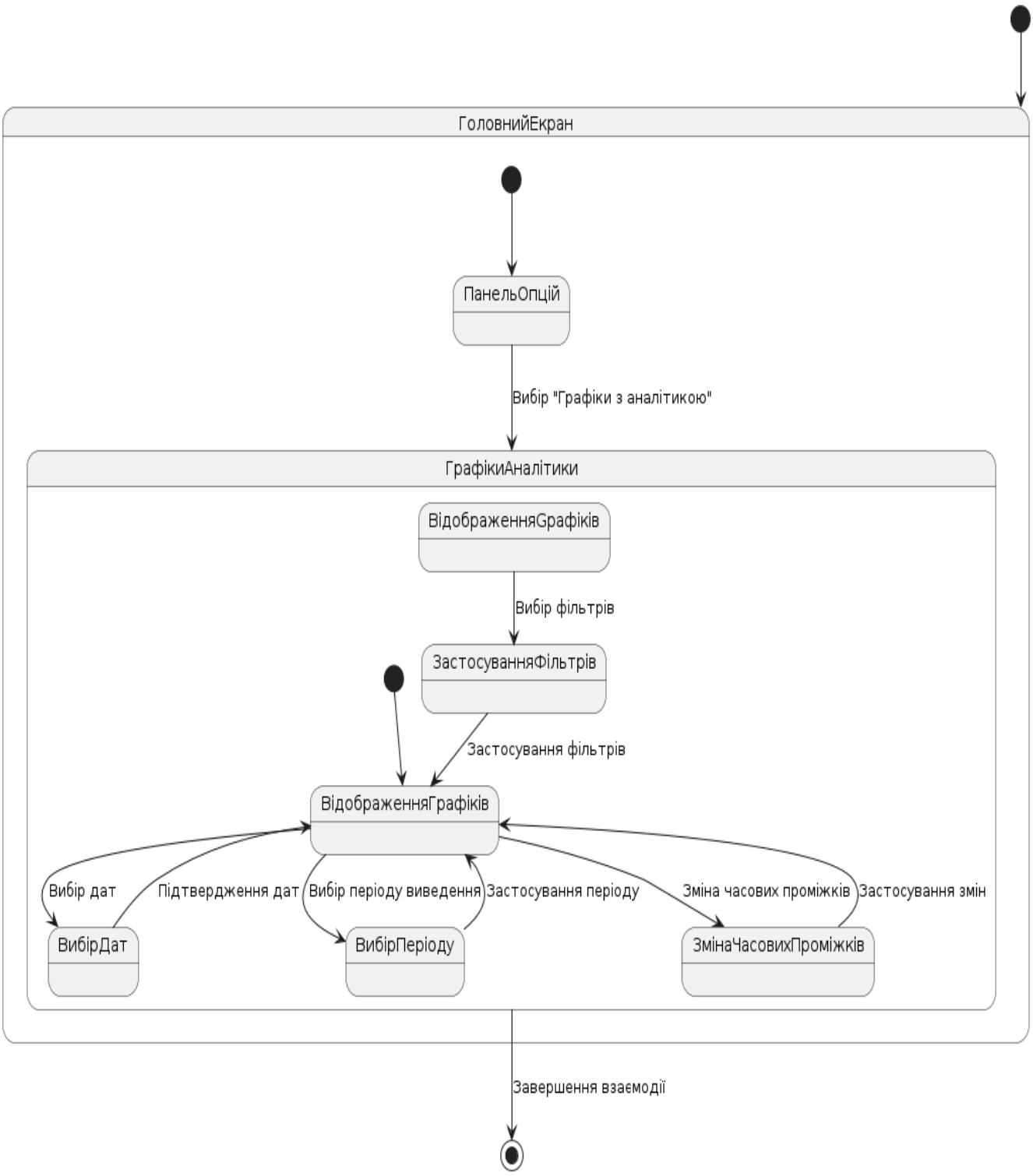

Рисунок Г.5 Діаграма станів та переходів андроїд застосунку

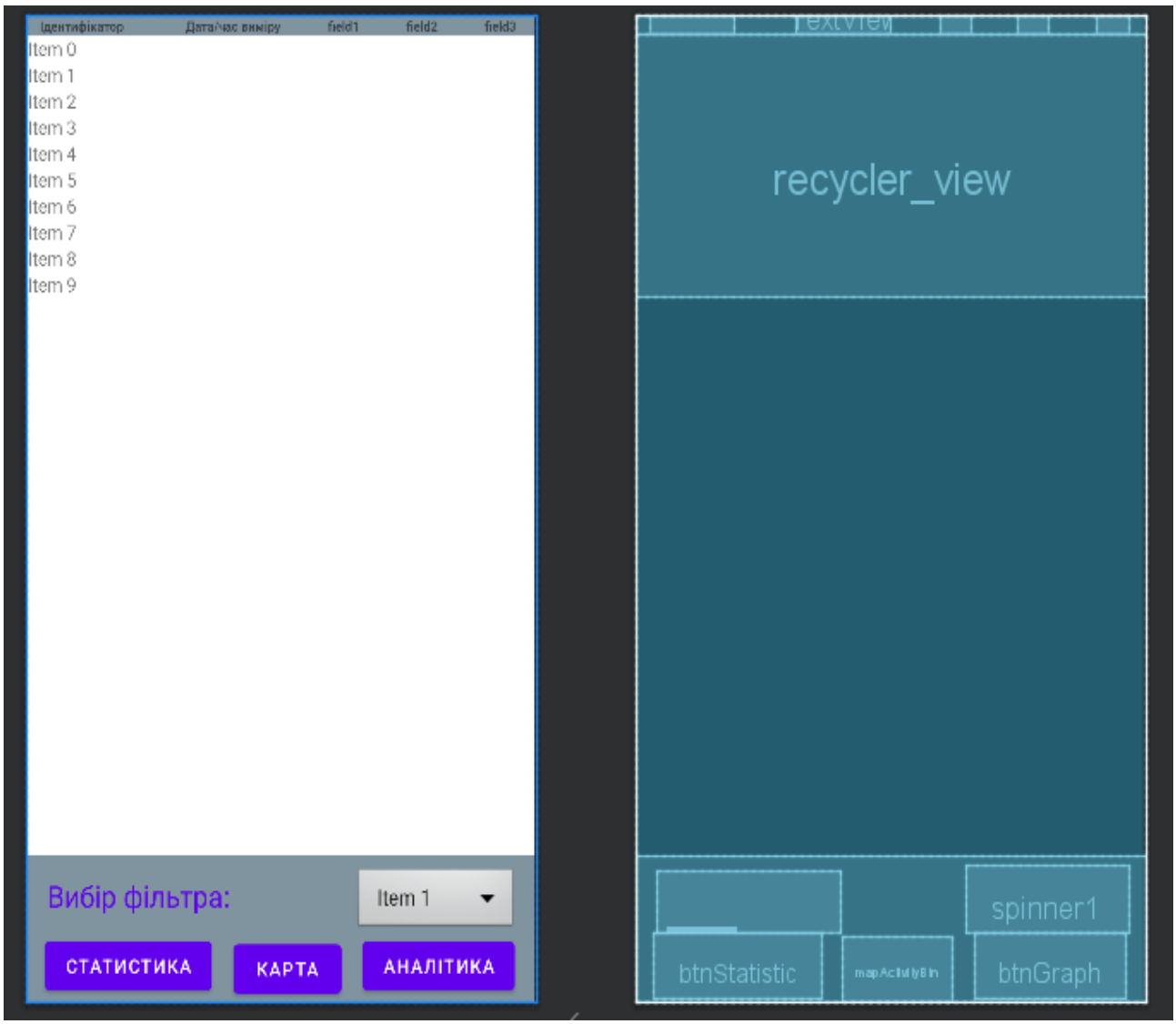

Рисунок Г.6 – Макет інтерфейсу користувача

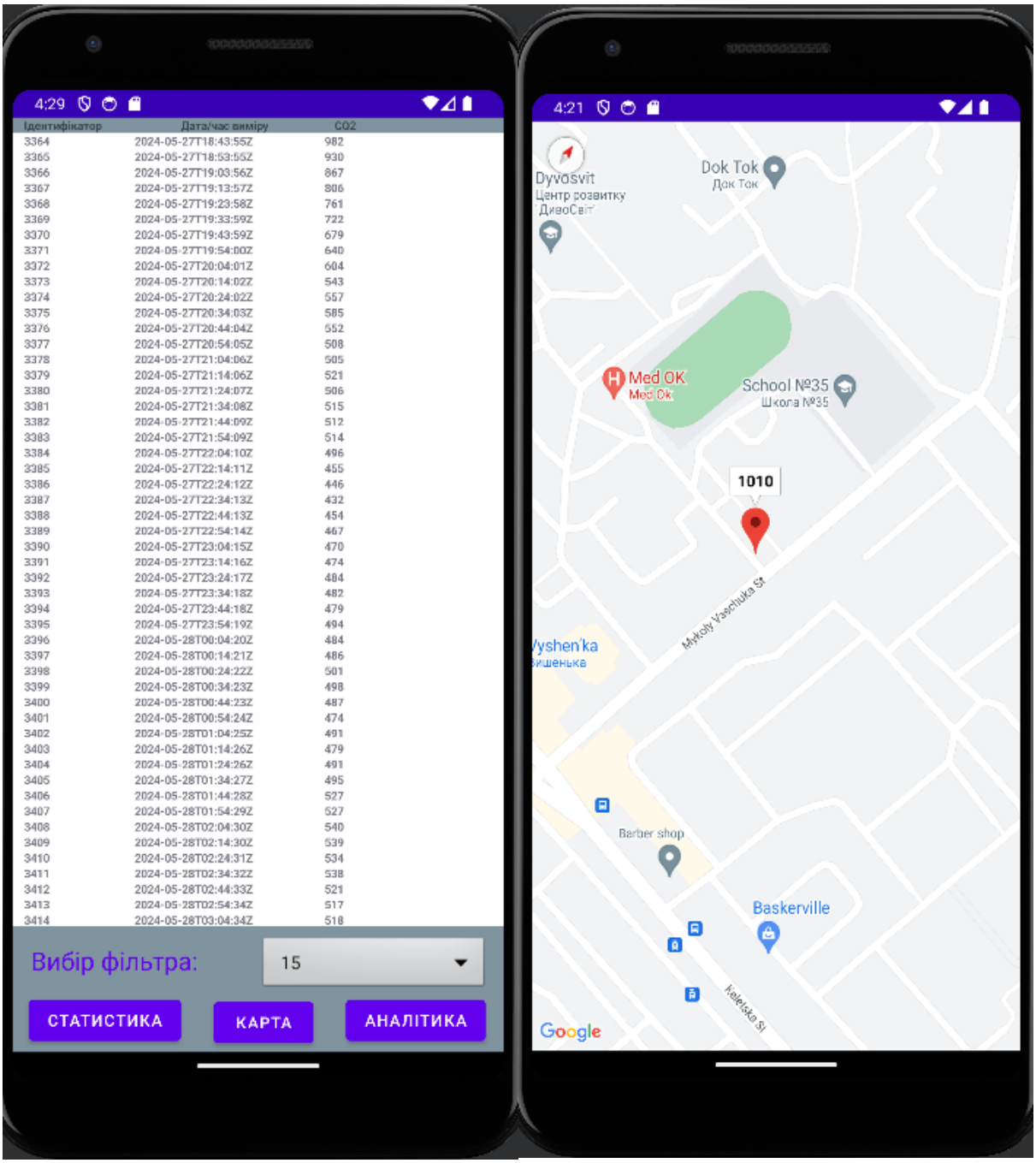

Рисунок Г.7 – Візуалізація даних вимірювань пристроїв IoT на карті