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

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

"Інформаційна технологія моніторингу стану атмосферного повітря за даними громадського моніторингу з використанням мобільних пристроїв"

Виконав: студент 2 курсу, групи 2ІСТ-24м
спеціальності 126 – «Інформаційні системи та технології»
_____ Сергій ДЖУРА
Керівник: к.т.н., доц. каф. САІТ
_____ Георгій ГОРЯЧЕВ
«27» 11 _____ 2025 р.
Опонент: доц. каф. КН
_____ Олексій СІЛАГІН
«04» 12 _____ 2025 р.

Допущено до захисту
Завідувач кафедри САІТ
_____ д.т.н., проф. Віталій МОКІН
«28» 11 _____ 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра системного аналізу та інформаційних технологій
Рівень вищої освіти – другий (магістерський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 126 Інформаційні системи та технології
Освітньо-професійна програма – Інформаційні технології аналізу даних та зображень

ЗАТВЕРДЖУЮ
Завідувач кафедри САІТ
_____ д.т.н., проф. Віталій МОКІН
«25» 09 2025 року

ЗАВДАННЯ
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Джурі Сергію Вікторовичу

1. Тема роботи: "Інформаційна технологія моніторингу стану атмосферного повітря за даними громадського моніторингу з використанням мобільних пристроїв",
керівник роботи: Георгій Горячев, к.т.н., доц. каф. САІТ,
затверджені наказом ВНТУ від «24» 09 2025 року №313
2. Строк подання студентом роботи «28» 11 2025 року
3. Вихідні дані до роботи:
Набір даних моніторингу атмосферного повітря (https://api.thingspeak.com/channels/3114388/feed.json?api_key=DDUB4I4THWKNSV6R) .
4. Зміст текстової частини:
   - загальна характеристика об'єкту досліджень;
   - вибір технологічного стеку та засобів розробки інформаційної технології моніторингу повітря з використанням мобільних пристроїв;
   - розроблення інформаційної технології моніторингу якості атмосферного повітря
   - економічна частина.
5. Перелік ілюстративного матеріалу:
   - головний екран застосунку;
   - UML-діаграма прецедентів
   - екран меню аналітики;
   - UML-діаграма класів
   - екран меню стастики;
   - вигляд графіків у веб інтерфейсі ThingSpeak;

6. Консультанти розділів МКР

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
4	Олександр ЛЕСЬКО, к. е. н., проф. каф. ЕПВМ	05.11.2025	15.11.2025

7. Дата видачі завдання «25» 09 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Загальна характеристика об'єкту досліджень	15.09.2025	25.09.2025	виконано
2	Вибір технологічного стеку та засобів розробки інформаційної технології моніторингу повітря з використанням мобільних пристроїв	25.09.2025	05.10.2025	виконано
3	Розробка інформаційної технології моніторингу якості атмосферного повітря	05.10.2025	25.10.2025	виконано
4	Удосконалення інформаційної технології	25.10.2025	05.11.2025	виконано
5	Економічна частина	05.11.2025	15.11.2025	виконано
6	Оформлення матеріалів до захисту МКР	15.11.2025	25.11.2025	виконано

Студент _____ Сергій ДЖУРА

Керівник роботи _____ Георгій ГОРЯЧЕВ

АНОТАЦІЯ

УДК 004.9:628.5

Джура С. В. Інформаційна технологія моніторингу стану атмосферного повітря за даними громадського моніторингу з використанням мобільних пристроїв. Магістерська кваліфікаційна робота зі спеціальності 126 – інформаційні системи та технології, освітньо-професійна програма – інформаційні технології аналізу даних та зображень. Вінниця: ВНТУ, 2025. 116 с.

На укр. мові. Бібліогр.: 24 назв; рис.: 14; табл.: 7.

В магістерській роботі здійснено аналіз та розроблення інформаційної технології моніторингу стану атмосферного повітря на основі даних громадського моніторингу з використанням мобільних пристроїв. Основну увагу приділено обробці, структуризації та візуалізації даних, отриманих зі станції громадського моніторингу, а також реалізації механізмів взаємодії мобільного застосунку з хмарною платформою для збору даних. У роботі запропоновано підхід до підвищення ефективності роботи системи моніторингу за рахунок використання сучасних методів аналізу даних та мобільних технологій для забезпечення доступності та оперативності контролю якості повітря.

Об’єктом дослідження є процес моніторингу якості атмосферного повітря на основі даних громадського моніторингу.

Ілюстративна частина складається з 7 плакатів із результатами аналізу.

У розділі економічної частини розглянуто питання про доцільність розробки та впровадження інформаційної технології визначення місць для нових постів моніторингу атмосферного повітря.

Ключові слова: інформаційна технологія, моніторинг якості атмосферного повітря міста, Android, Android Studio, громадський моніторинг, ThingSpeak.

ABSTRACT

Information Technology for Monitoring Atmospheric Air Quality Based on Community Monitoring Data Using Mobile Devices.. Master's thesis in specialty 126 - information systems and technologies, educational and professional program - information technology data and image analysis. Vinnytsia: VNTU, 2025. 116 p.

In Ukrainian language. Bibliogr .: 24 titles; fig .: 14; table: 7.

This master’s thesis presents the analysis and development of an information technology for monitoring atmospheric air quality based on community monitoring data with the use of mobile devices. The main focus is placed on the processing, structuring, and visualization of data obtained from a community monitoring station, as well as the implementation of mechanisms enabling the interaction of a mobile application with a cloud platform for data collection. The work proposes an approach to improving the efficiency of the monitoring system through the application of modern data analysis methods and mobile technologies to ensure accessibility and prompt assessment of air quality. The object of the study is the process of monitoring atmospheric air quality based on community monitoring data.

The illustrative part consists of 7 posters with the results of analysis.

In the section of the economic part the question of expediency of development and introduction of information technology of definition of places for new posts of monitoring of atmospheric air is considered.

Key words: information technology, urban air quality monitoring, Android, Android Studio, community monitoring, ThingSpeak

ЗМІСТ

ВСТУП

Актуальність теми. Забруднення атмосферного повітря є однією з ключових екологічних проблем сучасності, що суттєво впливає на стан здоров’я населення та якість життя. Попри наявність окремих ініціатив громадського моніторингу, отримані дані часто є фрагментарними, нерівномірно розподіленими у просторі та недостатньо зручними для оперативного аналізу. Це створює потребу в ефективних інструментах, здатних забезпечити безперервний збір, обробку та візуалізацію інформації про стан повітря. Використання мобільних пристроїв у поєднанні з сучасними інформаційними технологіями відкриває можливість підвищення доступності екологічних даних, оперативності прийняття рішень та залучення населення до контролю стану довкілля, що підкреслює актуальність розробки відповідної інформаційної технології.

Мета і завдання роботи Метою даної роботи є зменшення часу прийняття рішень при обробці та візуалізації стану атмосферного повітря на основі даних громадського моніторингу за рахунок використання мобільних пристроїв.

Розроблення інформаційної технології моніторингу стану атмосферного повітря за даними громадського моніторингу з використанням мобільних пристроїв передбачає виконання наступних задач:

– здійснити аналіз даних, отриманих зі станцій громадського моніторингу, та оцінити їхню придатність для мобільної обробки;

– вибрати середовище розробки та оптимальні технології для реалізації мобільного застосунку;

– підготувати та структурувати набір даних для їх коректної обробки і передачі через хмарну платформу;

– розробити архітектуру інформаційної технології та механізм взаємодії мобільного застосунку з хмарним сервісом;

– реалізувати функціонал збору, обробки та візуалізації показників якості атмосферного повітря на мобільному пристрої;

– протестувати роботу системи та оцінити її ефективність для задач громадського моніторингу.

Об’єктом дослідження магістерської кваліфікаційної роботи є процес моніторингу якості атмосферного повітря на основі даних громадського моніторингу з використанням мобільних пристроїв.

Предметом дослідження магістерської кваліфікаційної роботи є інформаційні технології та засоби організації мобільного моніторингу стану атмосферного повітря за даними громадських моніторингових станцій.

Методи дослідження. У роботі застосовано методи аналізу даних, інструменти обробки часових рядів, засоби роботи з геопросторовою інформацією, а також технології мобільних платформ, що забезпечують збір, передачу та візуалізацію даних громадського моніторингу якості повітря.

Новизна одержаних результатів полягає в подальшому розвитку інформаційної технології моніторингу атмосферного повітря шляхом зменшення часу прийняття рішень за рахунок використання мобільних засобів збору та відображення інформації. Запропонований підхід забезпечує підвищення оперативності контролю та аналізу фактичних екологічних показників.

Практичне значення роботи полягає у створені рішень, що сприяють зменшенню часу прийняття рішень при моніторингу якості атмосферного повітря.

Апробація та публікації результатів магістерської кваліфікаційної роботи. Розроблений у межах магістерської роботи мобільний додаток зареєстровано та має свідоцтво про авторське право, що підтверджує його оригінальність та право власності на програмне забезпечення [Додаток Д].

1 ЗАГАЛЬНА ХАРАКТЕРИСТИКА ОБЄ'КТУ ДОСЛІДЖЕНЬ

1.1 Опис об'єкту дослідження

Об'єктом дослідження є процес моніторингу якості атмосферного повітря в міських умовах із використанням даних громадського моніторингу та мобільних технологій. Дослідження охоплює розроблення інформаційної системи, що інтегрує екологічні дані з відкритих джерел, здійснює їх обробку та забезпечує представлення результатів користувачам у зручному форматі за допомогою мобільного застосунку.

Якість атмосферного повітря є одним із ключових чинників, що безпосередньо впливають на стан здоров'я населення, екологічну ситуацію в містах та загальну якість життя. За даними Всесвітньої організації охорони здоров'я (ВООЗ), забруднення повітря є однією з найсерйозніших екологічних загроз для людства, спричиняючи понад 7 мільйонів передчасних смертей щороку [1].

В Україні система державного моніторингу повітря характеризується обмеженою кількістю стаціонарних спостережних постів, що призводить до дефіциту оперативної інформації про стан атмосфери. Це ускладнює своєчасне реагування на локальні епізоди забруднення та знижує ефективність екологічної політики на місцевому рівні.

Основними забруднювачами атмосферного повітря в урбанізованих територіях є тверді частинки різного діаметра ($PM_{2.5}$ та PM_{10}), оксиди азоту (NO_2), озон (O_3), діоксид сірки (SO_2) та чадний газ (CO). Найбільшу небезпеку становлять дрібнодисперсні частинки $PM_{2.5}$, що здатні проникати глибоко в дихальні шляхи та потрапляти в кровоносну систему, створюючи серйозні ризики для здоров'я, особливо для людей із респіраторними та серцево-судинними захворюваннями.

У зв’язку з недостатнім розвитком державних систем моніторингу в Україні дедалі більшого поширення набуває громадський моніторинг, який передбачає збір, обмін і аналіз даних про стан атмосферного повітря силами громадян, громадських організацій та наукових ініціатив. Такий підхід ґрунтується на принципах відкритості даних, доступності сенсорних технологій та колективної участі.

Перевагою громадських систем моніторингу є висока щільність покриття території, можливість встановлення сенсорів безпосередньо в житлових районах і соціально важливих локаціях (біля шкіл, лікарень, дитячих садків тощо), а також оперативність отримання даних у режимі реального часу.

Водночас подібні системи мають низку обмежень, серед яких — різна точність сенсорів, відсутність уніфікації форматів даних, а також перерви у зборі інформації через технічні несправності або нестабільне з’єднання. Тому важливою складовою інформаційної технології моніторингу є валідація, нормалізація та узгодження даних із різних джерел [2].

Інформаційні технології відіграють ключову роль у перетворенні первинних даних моніторингу на корисну, аналітично обґрунтовану інформацію для кінцевих користувачів. Розроблювана інформаційна технологія моніторингу стану атмосферного повітря передбачає реалізацію комплексу взаємопов’язаних етапів — від збору даних до їх інтелектуального аналізу та візуального представлення.

Умовно систему можна представити у вигляді п’ятирівневої архітектури:

Збір та агрегація даних із різних джерел громадського моніторингу через відкриті API та платформи (зокрема Ecocity). Це забезпечує актуальність інформації та можливість створення довгострокових історичних баз для аналізу трендів.

Обробка та валідація даних, що включає фільтрацію аномальних значень, нормалізацію показників і розрахунок агрегованих індексів, таких як індекс якості повітря (AQI).

Візуалізація інформації — подання даних у зрозумілій для користувачів формі: інтерактивні карти забруднення, графіки динаміки показників, кольорове маркування рівнів небезпеки відповідно до міжнародних стандартів.

Інформування користувачів та рекомендації, які враховують географічне розташування, поточні показники та вразливість різних груп населення.

Мобільна платформа, яка забезпечує доступність інформації на смартфонах, використання геолокації для визначення найближчих точок моніторингу, адаптивний інтерфейс та можливість автономної роботи завдяки кешуванню даних.

Таким чином, об’єктом дослідження виступає комплексний процес моніторингу атмосферного повітря, що включає збір, обробку, валідацію, збереження, візуалізацію та мобільне представлення даних громадського моніторингу з використанням сучасних інформаційних технологій.

1.2 Аналіз аналогів

Для формування комплексних вимог до розроблюваної інформаційної технології було проведено детальний аналіз існуючих рішень у сфері моніторингу якості атмосферного повітря. Огляд охоплює як глобальні міжнародні платформи з багаторічною історією розвитку, так і локальні українські ініціативи, що з'явилися у відповідь на специфічні потреби вітчизняних користувачів. Аналіз проводився за критеріями функціональності, зручності користування, технологічної реалізації, доступності даних та адаптованості до українського ринку.

Міжнародні платформи моніторингу. Глобальні платформи моніторингу якості повітря накопичили значний досвід у збиранні, обробці та візуалізації

екологічних даних. Ці системи характеризуються високим рівнем технологічної зрілості, розвиненою інфраструктурою та великою базою користувачів.

IQAir (раніше відомий як AirVisual) є однією з найбільших та найавторитетніших глобальних платформ моніторингу якості повітря. Компанія, заснована у Швейцарії, об'єднує дані з понад 100 країн світу, створюючи найповнішу картину стану атмосфери на планеті. Платформа IQAir агрегує інформацію як з офіційних державних станцій моніторингу, так і з численних датчиків громадського моніторингу, що забезпечує високу щільність покриття та актуальність даних [3].

Основу системи IQAir становить глобальна карта якості повітря, що працює у режимі реального часу та відображає поточні показники забруднення у тисячах міст світу. Кожна точка на карті кольорово кодується відповідно до рівня забруднення за шкалою AQI (Air Quality Index) американського Агентства з охорони навколишнього середовища (EPA). Мобільні додатки для iOS та Android забезпечують зручний доступ до інформації в будь-який час та в будь-якому місці. Додатки оптимізовані для швидкої роботи, мають інтуїтивний інтерфейс та надають персоналізовані рекомендації щодо активності на відкритому повітрі [3].

Унікальною особливістю IQAir є система прогнозування якості повітря, що базується на складних метеорологічних моделях, історичних даних та машинному навчанні. Прогнози надаються на період до 7 днів вперед, що дозволяє користувачам планувати свою діяльність з урахуванням очікуваного стану атмосфери. Платформа підтримує можливість порівняння якості повітря між різними містами світу, що особливо корисно для людей, які планують подорожі або релокацію.

До переваг IQAir слід віднести величезну базу даних з понад 30,000 точок моніторингу у всьому світі, що робить платформу найбільшою в своєму роді. Професійна візуалізація даних включає не тільки карти та графіки, але й детальну інформацію про кожен тип забруднювача з поясненнями його впливу

на здоров'я. Система надає диференційовані рекомендації для різних груп населення - дітей, літніх людей, вагітних жінок, астматиків та людей з серцево-судинними захворюваннями. Інтеграція з розумними будинками через Amazon Alexa, Google Home та Apple HomeKit дозволяє керувати системами вентиляції та очищення повітря на основі актуальних даних.

Проте платформа має і певні недоліки. Покриття України є обмеженим - дані доступні лише для кількох найбільших міст, причому щільність точок моніторингу значно нижча порівняно з країнами Західної Європи чи США. Частина розширених функцій, зокрема детальні історичні дані, експорт інформації та відсутність реклами, доступна лише в платній преміум-версії. Фокус на великих містах означає, що малі населені пункти та сільські райони залишаються практично не охопленими моніторингом. Відсутність локалізації українською мовою ускладнює використання платформи для україномовних користувачів, хоча інтерфейс доступний англійською та кількома іншими мовами.

AirNow є офіційною платформою моніторингу якості повітря Сполучених Штатів Америки, розробленою та підтримуваною Агентством з охорони навколишнього середовища США (EPA) у співпраці з NOAA (National Oceanic and Atmospheric Administration), NASA та іншими державними агенціями. Платформа об'єднує дані з понад 2000 офіційних станцій моніторингу, розташованих по всій території США [4].

Основою AirNow є інтерактивна карта з кольоровим кодуванням, де різні кольори відповідають шести категоріям якості повітря - від зеленого (хороша) до темно-червоного (небезпечна). Мобільний додаток AirNow Mobile надає доступ до актуальної інформації, дозволяє зберігати улюблені локації та отримувати push-сповіщення про погіршення якості повітря. Особливу увагу платформа приділяє інформуванню про лісові пожежі та димові викиди, що є актуальною проблемою для західних штатів США.

Перевагами AirNow є надзвичайно високий рівень точності та валідації даних, оскільки всі вимірювання проводяться сертифікованим обладнанням

згідно з суворими стандартами EPA. Офіційний статус платформи забезпечує високу довіру з боку користувачів та можливість використання даних для юридичних та регуляторних цілей. Детальні поради щодо здоров'я розроблені медичними експертами та регулярно оновлюються на основі нових наукових досліджень. Важливо відзначити, що EPA надає безкоштовний API для розробників, що стимулює створення сторонніх додатків та інтеграцій.

Головним недоліком AirNow для української аудиторії є те, що платформа обмежена виключно територією Сполучених Штатів, хоча в окремих випадках доступна інформація про прикордонні райони Канади та Мексики. Система не застосовна для моніторингу якості повітря в Україні та інших країнах. Також, незважаючи на велику кількість станцій, їх щільність покриття все одно нижча порівняно з деякими системами громадського моніторингу.

Breezometer є комерційною платформою ізраїльської компанії, що спеціалізується на наданні високоточних даних про якість повітря через програмний інтерфейс (API). На відміну від попередніх платформ, орієнтованих на кінцевих користувачів, Breezometer позиціонується переважно як B2B рішення для інтеграції в сторонні додатки та сервіси [5].

Унікальною особливістю Breezometer є надзвичайно високе просторове розрізнення даних. У деяких містах платформа надає інформацію з точністю до 5 метрів, використовуючи складні алгоритми інтерполяції та машинного навчання. Система інтегрує дані з численних джерел - офіційних станцій моніторингу, супутникових спостережень, метеорологічних станцій, даних про дорожній трафік та промислові викиди. Прогнози якості повітря доступні на період до 96 годин (4 доби) вперед з погодинною деталізацією.

До переваг платформи належать передові алгоритми машинного навчання, що дозволяють не просто інтерполювати наявні дані, а реально прогнозувати якість повітря з урахуванням численних факторів. Надзвичайно детальна просторова інформація робить Breezometer незамінним інструментом для додатків, що потребують гіперлокальних даних. Надійність

API та висока доступність сервісу (понад 99.9% uptime) забезпечують безперебійну роботу інтегрованих рішень. Не дивно, що послугами Breezometer користуються такі гіганти як Google (для Google Maps), Apple (для Weather app), Microsoft та інші великі компанії.

Проте використання Breezometer має суттєві обмеження. Доступ до API є платним, причому вартість залежить від кількості запитів та географічного покриття. Безкоштовний рівень обслуговування дозволяє лише дуже обмежену кількість запитів на день, недостатню для повноцінного функціонування додатку з реальною базою користувачів. Покриття України є недостатнім, особливо поза межами найбільших міст, що обмежує застосовність платформи для локального ринку.

Європейські ініціативи громадського та державного моніторингу. Європейський континент характеризується розвиненою культурою екологічного моніторингу як на рівні офіційних державних структур, так і на рівні громадських ініціатив. Європейські підходи часто поєднують строгі стандарти якості даних з принципами відкритості та громадської участі.

European Environment Agency (EEA) представляє офіційну систему моніторингу якості повітря Європейського Союзу. Агенція координує збір та аналіз екологічних даних з усіх країн-членів ЄС, а також деяких країн-партнерів. Платформа об'єднує інформацію з понад 4000 офіційних станцій моніторингу, розташованих по всій Європі [6].

Центральним елементом системи є Європейський індекс якості повітря (EAQI - European Air Quality Index), що використовує уніфіковану шкалу від 0 до 100+ для оцінки стану атмосфери. На відміну від американської системи AQI, європейський індекс використовує дещо іншу методологію розрахунку та порогові значення. Інтерактивна карта європейського континенту дозволяє переглядати актуальні дані та прогнози для будь-якої країни або регіону. Платформа надає доступ до детальних історичних даних за багато років, що є безцінним ресурсом для наукових досліджень, аналізу довгострокових трендів та оцінки ефективності екологічної політики.

Перевагами системи EEA є офіційний статус та висока достовірність даних, що забезпечується суворими європейськими стандартами вимірювань та регулярною сертифікацією обладнання. Єдиний стандарт для всіх країн ЄС дозволяє коректно порівнювати екологічну ситуацію між різними державами та регіонами. Всі дані є повністю відкритими для дослідників, розробників та громадськості, що сприяє прозорості та розвитку інновацій. Інтеграція з національними системами моніторингу забезпечує потік інформації від локального до європейського рівня.

До недоліків слід віднести те, що Україна, не будучи членом Європейського Союзу, не входить до централізованої системи моніторингу EEA, хоча співпраця в рамках програми Європейського сусідства розвивається. У деяких регіонах, особливо в сільській місцевості, щільність станцій моніторингу є недостатньою для детального картування якості повітря. Офіційний веб-інтерфейс EEA, розроблений переважно для професійної аудиторії (науковців, політиків, регуляторів), може здаватися складним для пересічних користувачів. Також іноді спостерігаються затримки в оновленні даних через складну багаторівневу систему збору та валідації інформації.

Sensor.Community (відомий раніше як Luftdaten.info) представляє собою одну з найбільших та найуспішніших міжнародних мереж громадського моніторингу якості повітря. Проект зародився в 2015 році в німецькому місті Штутгарт за ініціативи місцевих активістів та швидко поширився по всьому світу. Філософія проекту базується на принципах відкритості, доступності та громадської участі - будь-хто може створити власний датчик за детальними інструкціями та приєднатися до глобальної мережі [7].

Платформа Sensor.Community є повністю відкритою - від апаратного забезпечення (open hardware) до програмного коду (open source) та даних (open data). Детальні інструкції дозволяють навіть людям без спеціальної технічної підготовки самостійно зібрати датчик якості повітря з доступних компонентів вартістю близько 30-40 євро. Глобальна карта платформи відображає понад

15,000 активних датчиків у різних куточках світу, від мегаполісів до невеликих сіл. Відкритий API надає доступ до всіх даних без будь-яких обмежень, що зробило Sensor.Community популярною платформою для розробників, дослідників та освітніх проектів.

Серед переваг проекту варто відзначити його повну відкритість та безкоштовність - відсутні будь-які приховані платежі, підписки чи обмеження функціоналу. Велика міжнародна спільнота розробників, активістів та ентузіастів постійно вдосконалює платформу, додає нові можливості та підтримує новачків. DIY (Do It Yourself) підхід не тільки знижує вартість участі, але й сприяє освіті та розумінню принципів моніторингу якості повітря. Важливо, що в Україні вже функціонує кілька сотень датчиків Sensor.Community, особливо в таких містах як Київ, Львів, Дніпро, Одеса. Детальна документація проекту доступна кількома мовами та охоплює всі аспекти від збирання датчика до інтерпретації даних.

Недоліками платформи є неоднорідна якість даних, що виникає через використання різних типів датчиків, різні умови їх розміщення та експлуатації. Хоча існують рекомендації щодо калібрування, відсутність централізованої валідації означає, що деякі дані можуть бути неточними. Базовий веб-інтерфейс платформи є функціональним, але відстає за зручністю та естетикою від комерційних рішень. Складність для непідготовлених користувачів полягає в тому, що для повноцінного використання платформи бажано мати певні технічні знання.

Українські платформи та локальні ініціативи. Розвиток систем моніторингу якості повітря в Україні набув особливої активності протягом останніх п'яти років у відповідь на зростання екологічної свідомості населення та обмеженість офіційної інфраструктури спостережень. Українські ініціативи відрізняються адаптованістю до локальних умов, використанням української мови та фокусом на актуальних для вітчизняних користувачів проблемах.

SaveEcoBot є одним з найбільш комплексних українських проектів в галузі екологічного моніторингу. Платформа включає не тільки моніторинг

якості повітря, але й контроль якості води, інформацію про викиди промислових підприємств, радіаційний фон та інші екологічні параметри. Проект реалізований як телеграм-бот у поєднанні з веб-платформою, що забезпечує максимальну доступність для українських користувачів [8].

Інтеграція з українськими станціями моніторингу охоплює як державні пости спостереження Центральної геофізичної обсерваторії та регіональних центрів гідрометеорології, так і громадські мережі датчиків. Телеграм-бот SaveEcoBot дозволяє швидко отримати інформацію про якість повітря в конкретному місті, просто надіславши назву населеного пункту або геолокацію. Веб-карта відображає точки моніторингу по всій Україні з актуальними показниками та історичними даними. Унікальною особливістю платформи є можливість подання скарг на екологічні порушення безпосередньо через бот, що сприяє громадському контролю за дотриманням природоохоронного законодавства.

Переваги SaveEcoBot включають чіткий фокус на українські міста та актуальні для країни екологічні проблеми. Інтеграція як з державними, так і з громадськими джерелами даних забезпечує більш повне покриття території. Зручний телеграм-інтерфейс звичний для мільйонів українців та не вимагає встановлення окремого додатку. Платформа підтримується активною громадою користувачів та має безкоштовний доступ до всіх функцій. Інформація про викиди промислових підприємств та можливість скарг додають громадянської значущості проекту.

Серед недоліків слід відзначити обмежену кількість точок моніторингу якості повітря, особливо в малих містах та сільській місцевості. Актуальність даних іноді викликає питання - не всі джерела оновлюються в режимі реального часу. Відсутність повноцінного нативного мобільного додатку обмежує можливості візуалізації та інтерактивності. Функціонал візуалізації даних є досить базовим порівняно з міжнародними аналогами - відсутні детальні графіки, прогнози, порівняльний аналіз.

LUN Air (доступний за адресою air.kiev.ua) є проектом популярного українського порталу нерухомості LUN.ua, присвяченим моніторингу якості повітря в столиці України. Платформа з'явилася у 2019 році та швидко стала одним з основних джерел інформації про екологічну ситуацію в Києві для мешканців міста [9].

Платформа створила власну мережу станцій моніторингу, розташованих у різних районах Києва, що забезпечує детальне покриття території столиці. Веб-карта відображає показники якості повітря в режимі реального часу з кольоровим кодуванням залежно від рівня забруднення. Історичні графіки дозволяють переглянути зміни якості повітря за різні періоди - день, тиждень, місяць, рік. Унікальною особливістю є рейтинг районів Києва за чистотою повітря, що допомагає потенційним покупцям нерухомості враховувати екологічний фактор при виборі місця проживання. Платформа також аналізує кореляцію між якістю повітря та вартістю нерухомості в різних районах.

До переваг LUN Air належить дуже детальне покриття столиці з високою щільністю станцій моніторингу, що забезпечує точну локалізацію зон забруднення. Якісна візуалізація даних включає як карти, так і різноманітні графіки та діаграми. Регулярне оновлення інформації з інтервалом 15-30 хвилин забезпечує актуальність даних. Зрозумілий інтерфейс повністю українською мовою робить платформу доступною для широкої аудиторії. Додаткова аналітика по районах, включаючи порівняння показників, тренди та сезонні особливості, надає глибше розуміння екологічної ситуації.

Недоліками платформи є обмеження виключно містом Києвом мешканці інших українських міст не можуть скористатися сервісом. Відсутність мобільного додатку означає, що доступ можливий тільки через веб-браузер, що менш зручно для щоденного використання. Комерційна мета проекту, пов'язана з продажем нерухомості, може створювати потенційний конфлікт інтересів щодо об'єктивності інформації. Відсутність публічного API не дозволяє стороннім розробникам створювати інтеграції чи використовувати дані для власних проектів.

Air Lviv представляє львівську ініціативу громадського моніторингу якості повітря, що з'явилася в 2016 році як відповідь на проблему забруднення повітря в місті, особливо в опалювальний сезон. Проект об'єднує зусилля громадських активістів, IT-спеціалістів та екологічно свідомих мешканців міста [10].

Мережа громадських датчиків у Львові налічує кілька десятків точок, розташованих як у центральній частині міста, так і в спальних районах. Веб-платформа надає доступ до актуальних даних з картографічною візуалізацією та базовою аналітикою. Проект активно співпрацює з міською владою Львова, надаючи дані для прийняття управлінських рішень в екологічній сфері. Важливою складовою є освітні заходи - лекції, воркшопи, шкільні програми, спрямовані на підвищення екологічної свідомості мешканців. Всі зібрані дані є відкритими та доступними для аналізу науковцями, студентами та всіма зацікавленими сторонами.

Переваги проекту включають активну громадську спільноту, що забезпечує не тільки технічну підтримку платформи, але й адвокацію екологічних змін на міському рівні. Висока щільність покриття міста дозволяє ідентифікувати локальні джерела забруднення та відстежувати ефективність заходів щодо їх усунення. Прозорість даних та відкритість проекту сприяють довірі з боку користувачів. Локальна підтримка та розвиток забезпечують адаптованість до специфічних потреб Львова.

Недоліками є обмеження одним містом, що не дозволяє мешканцям інших регіонів України скористатися сервісом. Веб-інтерфейс є функціональним, але базовим з точки зору дизайну та user experience. Відсутність мобільних додатків обмежує зручність щоденного використання. Залежність від волонтерів означає, що розвиток проекту може уповільнюватися через обмеженість ресурсів та часу учасників.

1.3 Висновки

У першому розділі роботи проведено аналіз об’єкта дослідження — процесу моніторингу якості атмосферного повітря в міських умовах із використанням даних громадського моніторингу та мобільних інформаційних технологій. Актуальність теми зумовлена обмеженою кількістю офіційних станцій спостереження та потребою в оперативному інформуванні населення про рівень забруднення повітря. Громадські сенсорні мережі забезпечують ширше покриття території, відкритість даних і можливість інтеграції з мобільними застосунками, що робить їх важливим доповненням до державних систем.

Проведений аналіз існуючих міжнародних (IQAir, AirNow, BreezoMeter), європейських (EEA, Sensor.Community) та українських (SaveEcoBot, LUN Air, Air Lviv) платформ показав відсутність комплексного рішення, орієнтованого на українську аудиторію та інтегрованого з локальними джерелами громадського моніторингу. Міжнародні сервіси не мають належного покриття України й україномовної локалізації, тоді як вітчизняні ініціативи обмежені окремими містами та здебільшого не пропонують мобільних застосунків.

Виявлені недоліки аналогів обґрунтовують необхідність створення нової інформаційної технології, яка забезпечить збір, обробку, валідацію та візуалізацію екологічних даних через зручний мобільний інтерфейс. Розроблюваний застосунок має на меті об’єднати переваги відкритих даних, інтеграцію з API та сучасні засоби візуалізації, створивши доступний інструмент для українських користувачів з метою підвищення екологічної поінформованості та захисту здоров’я населення

2 ВИБІР ТЕХНОЛОГІЧНОГО СТЕКУ ТА ЗАСОБІВ РОЗРОБКИ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ МОНІТОРИНГУ ПОВІТРЯ З ВИКОРИСТАННЯМ МОБІЛЬНИХ ПРИСТРОЇВ

2.1 Вибір середовища розробки

При виборі платформи для розробки інформаційної технології було проведено аналіз поточного стану ринку мобільних операційних систем та специфіки цільової аудиторії проєкту. Згідно зі статистичними даними міжнародних аналітичних агентств за 2024 рік, операційна система Android займає домінуюче становище на глобальному ринку мобільних пристроїв з часткою близько сімдесяти одного відсотка. На території України цей показник є ще більш значущим, становлячи від сімдесяти п'яти до вісімдесяти відсотків загальної кількості активних мобільних пристроїв [11].

Вибір платформи Android для реалізації інформаційної технології моніторингу якості атмосферного повітря обґрунтовується сукупністю ключових факторів. По-перше, висока частка ринку дозволяє забезпечити максимальне охоплення цільової аудиторії, що є критично важливим для проєкту громадського моніторингу, оскільки ефективність системи безпосередньо залежить від кількості користувачів, які мають доступ до інформації. По-друге, відкритість платформи Android надає розробникам ширші можливості для інтеграції з зовнішніми сервісами та програмними інтерфейсами, що є необхідним для взаємодії з платформами збору даних Інтернету речей. По-третє, різноманітність пристроїв від різних виробників у широкому ціновому діапазоні дозволяє користувачам з різним рівнем доходу долучитися до системи моніторингу, що відповідає соціальній спрямованості проєкту. По-четверте, наявність потужних інструментів розробки, офіційно підтримуваних компанією Google, забезпечує ефективний процес створення якісного програмного забезпечення. По-п'яте, велика міжнародна спільнота

розробників гарантує доступність навчальних матеріалів, бібліотек та готових рішень типових проблем, що виникають у процесі розробки.

При прийнятті рішення також було розглянуто можливість розробки кросплатформного додатку з використанням таких фреймворків як React Native або Flutter, що дозволило б створити єдину кодову базу для платформ Android та iOS. Проте для даного проєкту було прийнято рішення використовувати нативну розробку з наступних міркувань. Нативні додатки демонструють значно вищу продуктивність, що є критично важливим при роботі з інтерактивними картами та обробкою великих обсягів даних вимірювань. Нативна розробка надає повний доступ до всіх можливостей та програмних інтерфейсів платформи Android без будь-яких обмежень або проміжних шарів абстракції. Крім того, нативні додатки забезпечують кращу інтеграцію з системою дизайну Material Design та оптимальне споживання ресурсів пристрою, зменшуючи навантаження на батарею та оперативну пам'ять [12].

2.2 Вибір мови програмування

Для розробки додатку на платформі Android було прийнято рішення використовувати дві офіційно підтримувані мови програмування - Kotlin та Java, що є стандартною практикою в сучасній Android-розробці. Основний обсяг нового коду написано мовою Kotlin, тоді як Java використовується для окремих компонентів, що забезпечує оптимальне поєднання сучасних можливостей та сумісності. Такий підхід дозволяє використовувати переваги обох мов та забезпечує гнучкість при інтеграції існуючих бібліотек і компонентів.

Kotlin представляє собою сучасну статично типізовану мову програмування, розроблену компанією JetBrains та офіційно рекомендовану компанією Google для розробки на платформі Android починаючи з 2017 року. З того часу Kotlin активно набуває популярності серед розробників - за даними

Google, понад 70% топових Android-додатків використовують Kotlin у своєму коді. Ключовою перевагою мови Kotlin є система типів з підтримкою nullable-безпеки, що на рівні компілятора запобігає помилкам через нульові посилання, які є однією з найчастіших причин аварійного завершення додатків. Це досягається через розділення типів на nullable (з можливістю містити null) та non-nullable (які не можуть бути null), що позначається оператором "?" після типу даних.

Лаконічність синтаксису Kotlin дозволяє писати значно менше коду для досягнення того ж результату порівняно з Java, що полегшує читання та підтримку програмного забезпечення. Наприклад, створення data-класів у Kotlin вимагає лише одного рядка коду, тоді як еквівалентна реалізація в Java потребує написання конструкторів, геттерів, сеттерів, методів equals(), hashCode() та toString(). У процесі розробки даного проєкту це дозволило скоротити обсяг коду приблизно на 30-35% порівняно з гіпотетичною реалізацією повністю на Java.

Мова підтримує механізм розширень функцій, що дозволяє додавати нові методи до існуючих класів без їх модифікації або успадкування. Ця особливість виявилася особливо корисною при роботі з класами Android SDK, оскільки дозволила створювати зручні утилітарні функції для часто використовуваних операцій без необхідності створення додаткових класів-обгорток. Наприклад, було створено розширення для класу Context, що спрощують показ Toast-повідомлень та перевірку доступності мережевого з'єднання.

Вбудований механізм корутин забезпечує легкий та ефективний підхід до асинхронного програмування, що є критично важливим для мобільних додатків, де блокування основного потоку призводить до зависання інтерфейсу. На відміну від традиційних потоків (threads), корутини є значно легшими з точки зору споживання ресурсів - на одному пристрої можна створити тисячі корутин без суттєвого впливу на продуктивність. У розробленому додатку корутини використовуються для виконання мережевих

запитів до Firebase, операцій читання та запису в локальну базу даних SQLite, а також для обробки та компресії зображень перед їх завантаженням на сервер.

Підтримка функцій вищого порядку та лямбда-виразів дозволяє використовувати функціональний стиль програмування, що робить код більш виразним та компактним. Kotlin надає багатий набір функцій для роботи з колекціями (map, filter, reduce, groupBy тощо), що дозволяє обробляти дані декларативним способом замість імперативного підходу з використанням циклів. Додатково, мова підтримує smart casts - автоматичне приведення типів після перевірки, що усуває необхідність явного приведення типів та робить код більш читабельним і безпечним [13].

Мова Java залишається в проєкті для забезпечення сумісності з деякими компонентами та існуючими рішеннями. Java є зрілою мовою програмування з великою екосистемою бібліотек, що накопичувалася протягом багатьох років існування платформи Android. Незважаючи на активне просування Kotlin, багато корпоративних проєктів та legacy-систем все ще написані на Java, тому розуміння цієї мови залишається важливою компетенцією для Android-розробників. Крім того, деякі спеціалізовані бібліотеки та SDK все ще надають кращу документацію та приклади коду саме для Java.

В рамках даного проєкту Java використовується для реалізації класу DatabaseHelper, що забезпечує роботу з локальною базою даних SQLite. Цей клас успадковує SQLiteOpenHelper - базовий клас Android SDK для управління версіями бази даних та її життєвим циклом. Рішення використати Java для цього компонента було прийнято з огляду на те, що більшість офіційної документації Android щодо SQLite містить приклади саме на Java, а також через наявність великої кількості перевірених шаблонів реалізації. DatabaseHelper відповідає за створення таблиць бази даних, керування міграціями при оновленні схеми та надає методи для базових CRUD-операцій (Create, Read, Update, Delete).

Важливо відзначити, що Kotlin та Java мають повну взаємну сумісність на рівні байт-коду JVM, що дозволяє вільно комбінувати код на обох мовах в

одному проєкті, викликати Java-код з Kotlin та навпаки без будь-яких додаткових перетворень або проміжних шарів. Обидві мови компілюються у байт-код, який виконується на Android Runtime (ART) - віртуальній машині, що є еволюцією класичної Dalvik VM. Це означає, що з точки зору продуктивності виконання немає практично жодної різниці між еквівалентним кодом, написаним на Kotlin чи Java. Під час розробки проєкту така взаємна сумісність дозволила поступово впроваджувати нові компоненти на Kotlin, зберігаючи при цьому робочі Java-модулі без необхідності їх негайного рефакторингу [14].

2.3 Огляд інтегрованого середовища Android Studio

В якості інтегрованого середовища розробки для створення мобільного додатку було обрано Android Studio, що є офіційним середовищем розробки для платформи Android, створеним компанією Google на базі популярної IDE IntelliJ IDEA від компанії JetBrains. Android Studio надає комплексний набір інструментів, необхідних для всіх етапів розробки мобільного додатку - від проектування інтерфейсу до налагодження та профілювання продуктивності. Важливою перевагою цього середовища є його безкоштовність та відкритий вихідний код, що робить його доступним для розробників будь-якого рівня. На момент розробки проєкту використовувалася версія Android Studio Hedgehog (2023.1.1), яка включає всі необхідні оновлення та підтримку останніх версій Android SDK [15].

Середовище включає інтелектуальний редактор коду з підтримкою автоматичного доповнення з урахуванням контексту, що значно прискорює процес написання коду та зменшує кількість помилок. Система автодоповнення аналізує не лише синтаксис мови, але й семантику коду, пропонуючи релевантні варіанти на основі типів даних, доступних методів та поточного контексту виконання. Під час розробки проєкту ця функція виявилася особливо корисною при роботі з Firebase API, оскільки автоматично

пропонувала доступні методи та їх параметри, що зменшило необхідність постійного звернення до документації.

Вбудовані можливості рефакторингу дозволяють безпечно змінювати структуру коду, автоматично оновлюючи всі посилання на модифіковані елементи. До таких операцій належать перейменування класів, методів та змінних, витягування методів з дублікованого коду, переміщення класів між пакетами, зміна сигнатур методів та інлайнінг змінних. Під час розробки функціональності роботи з рецептами було використано рефакторинг для виділення загальної логіки валідації даних в окремі утилітарні методи, що покращило структуру коду та його повторне використання.

Система статичного аналізу коду виявляє потенційні помилки, неоптимальні конструкції та порушення стандартів кодування ще на етапі написання, що підвищує якість програмного забезпечення. Android Studio використовує інструмент Android Lint, який перевіряє код на понад 400 різних типів проблем, включаючи витоки пам'яті, неправильне використання API, проблеми з продуктивністю, доступністю інтерфейсу та інтернаціоналізацією. Наприклад, Lint автоматично попереджає про використання застарілих методів API та пропонує сучасні альтернативи, що допомогло підтримувати код у відповідності з актуальними практиками Android-розробки.

Швидка навігація по проєкту через систему індексації дозволяє миттєво переходити до визначень класів, методів та змінних навіть у великих кодових базах. Функції пошуку включають глобальний пошук по всьому проєкту, пошук використань конкретного елемента коду, навігацію до суперкласів та реалізацій інтерфейсів. Особливо корисною виявилася можливість швидкого переходу між Activity, Fragment та їх відповідними layout-файлами через комбінацію клавіш, що прискорило процес розробки інтерфейсу.

Візуальний редактор інтерфейсів надає можливість створювати макети додатку в графічному режимі з попереднім переглядом на екранах різних розмірів та роздільних здатностей. Редактор працює у двох режимах: Design (візуальне редагування) та Split (одночасний перегляд графічного інтерфейсу

та XML-коду), що дозволяє обирати найзручніший спосіб роботи. Під час розробки інтерфейсу додатку використовувався переважно режим Split, оскільки він забезпечував повний контроль над XML-розміткою при збереженні візуального зворотного зв'язку.

Редактор підтримує технологію ConstraintLayout, що дозволяє створювати адаптивні інтерфейси, які коректно відображаються на пристроях з різними характеристиками екранів. ConstraintLayout замінив застарілі підходи з використанням вкладених LinearLayout та RelativeLayout, забезпечуючи кращу продуктивність рендерингу завдяки плоскій ієрархії елементів. У розробленому додатку всі основні екрани використовують ConstraintLayout для забезпечення адаптивності інтерфейсу до різних розмірів екранів - від компактних смартфонів до великих планшетів.

Інтеграція з компонентами Material Design забезпечує доступ до готових UI-елементів, що відповідають сучасним стандартам дизайну. Android Studio надає бібліотеку Material Components, яка включає такі елементи як FloatingActionButton, BottomNavigationView, CardView, TextInputLayout та інші компоненти з підтримкою анімацій та ефектів згідно з принципами Material Design. Використання цих компонентів дозволило створити сучасний та інтуїтивно зрозумілий інтерфейс додатку без необхідності розробки власних UI-компонентів з нуля.

Вбудований емулятор Android забезпечує швидкий запуск та роботу віртуальних пристроїв для тестування додатку без необхідності використання фізичних пристроїв на етапі розробки. Сучасні версії емулятора використовують апаратне прискорення через Intel HAXM або AMD Hypervisor, що забезпечує продуктивність, близьку до реальних пристроїв. Емулятор підтримує емуляцію різних моделей пристроїв з різними версіями операційної системи, конфігураціями екранів та апаратними характеристиками. У процесі розробки використовувалися емульовані пристрої з Android 8.0 (API 26) та Android 13 (API 33) для тестування сумісності додатку з різними версіями операційної системи.

Можливість симуляції роботи сенсорів, таких як GPS, акселерометр та гіроскоп, дозволяє тестувати функціональність, що залежить від датчиків пристрою. Емулятор також підтримує симуляцію різних мережевих умов (швидкість з'єднання, затримки, втрата пакетів), різних рівнів заряду батареї та зміни орієнтації екрану. Додатково можна тестувати роботу додатку при різних розмірах шрифту та налаштуваннях доступності, що важливо для забезпечення зручності використання різними категоріями користувачів.

Система збірки проєкту базується на інструменті Gradle, що автоматизує процес компіляції коду, управління залежностями та створення фінальних APK-файлів. Gradle є гнучкою системою автоматизації збірки, яка використовує декларативний підхід до опису конфігурації проєкту. Gradle підтримує різні варіанти збірки для різних конфігурацій, що дозволяє створювати окремі версії додатку для різних середовищ. У даному проєкті налаштовано два build варіанти: debug (для розробки та тестування з увімкненим логуванням) та release (оптимізована версія для публікації з вимкненим логуванням та увімкненою обфускацією коду через ProGuard).

Можливість гнучкого налаштування процесу збірки через скрипти на мові Kotlin DSL забезпечує повний контроль над всіма аспектами компіляції та пакування додатку. Gradle дозволяє автоматично підтягувати зовнішні бібліотеки з репозиторіїв Maven Central та Google Maven Repository, керувати версіями залежностей та вирішувати конфлікти між різними версіями бібліотек. У проєкті через Gradle підключено такі залежності як Firebase SDK, бібліотеки для роботи з зображеннями (Glide), компоненти Material Design та інші необхідні модулі.

Інструменти відладки включають потужний дебагер з підтримкою точок зупинки, покрокового виконання та перегляду стану змінних у режимі реального часу. Дебагер дозволяє встановлювати умовні точки зупинки, які спрацьовують лише при виконанні певних умов, що особливо корисно при налагодженні циклів та обробці великих масивів даних. Також підтримується функція "hot swap", яка дозволяє вносити зміни в код під час сесії

налагодження без повного перезапуску додатку, що значно прискорює процес виправлення помилок.

Система перегляду логів дозволяє фільтрувати та аналізувати повідомлення, що виводяться додатком під час роботи. Logcat надає можливість фільтрації повідомлень за рівнем пріоритету (Verbose, Debug, Info, Warning, Error), за тегами та за текстом повідомлення. Під час розробки активно використовувалося логування для відстеження виконання асинхронних операцій з Firebase та SQLite, що допомогло виявити та усунути кілька критичних помилок у логіці обробки даних.

Профайлер надає детальну інформацію про використання процесора, оперативної пам'яті, мережі та енергії батареї, що дозволяє виявляти та усувати проблеми з продуктивністю. CPU Profiler показує, які методи споживають найбільше процесорного часу, Memory Profiler допомагає виявляти витоки пам'яті та аналізувати розподіл об'єктів, Network Profiler відображає всі мережеві запити додатку з детальною інформацією про розмір даних та час виконання. За допомогою Memory Profiler було виявлено та усунуто витік пам'яті, пов'язаний з неправильним використанням Context у статичних змінних.

Інспектор макетів забезпечує візуальний аналіз ієрархії елементів інтерфейсу та їх властивостей. Цей інструмент дозволяє у реальному часі переглядати структуру UI на працюючому додатку або емуляторі, перевіряти розміри та позиціювання елементів, аналізувати складність ієрархії view та виявляти надмірно вкладені елементи, які можуть погіршувати продуктивність рендерингу.

Інспектор бази даних дозволяє переглядати та редагувати вміст локальних таблиць SQLite безпосередньо з середовища розробки. Це значно спростило процес налагодження роботи з локальною базою даних, оскільки з'явилася можливість перевіряти коректність збережених даних, виконувати SQL-запити для тестування та вручну модифікувати записи для перевірки різних сценаріїв роботи додатку. Інспектор також відображає схему бази

даних з типами полів та зв'язками між таблицями, що полегшує розуміння структури даних.

2.4 Вибір та обґрунтування бібліотек для реалізації функціоналу системи

Для реалізації функціональних вимог системи моніторингу якості атмосферного повітря було обрано технологічний стек, що включає перевірені та широко використовувані бібліотеки з активною підтримкою спільноти розробників. Робота з картографією та геолокацією реалізована з використанням комплексного набору взаємопов'язаних бібліотек. Основу картографічного функціоналу складає офіційний Google Maps SDK для платформи Android версії. Цей програмний інструментарій розроблений компанією Google та надає потужні можливості відображення інтерактивної географічної карти з підтримкою різних режимів візуалізації, включаючи звичайний режим з відображенням вулиць та назв об'єктів, супутниковий режим з фотографічними знімками місцевості та гібридний режим, що поєднує супутникові знімки з позначеннями доріг та назвами. Бібліотека забезпечує додавання маркерів з індивідуальними іконками для позначення розташування датчиків моніторингу на карті, що дозволяє користувачам візуально ідентифікувати станції спостереження. Підтримка інтуїтивних жестів для навігації картою включає масштабування через pinch-to-zoom, переміщення шляхом перетягування, обертання карти двома пальцями та зміну кута нахилу для тривимірного відображення місцевості [16].

Візуалізація історичних даних якості атмосферного повітря у вигляді інтерактивних графіків та діаграм реалізована з використанням бібліотеки MPAndroidChart. Вибір цієї бібліотеки було здійснено після детального аналізу доступних на ринку рішень для візуалізації даних на платформі Android з урахуванням функціональних можливостей, продуктивності, якості документації та активності підтримки спільнотою розробників.

MPAndroidChart являє собою потужну та гнучку бібліотеку з відкритим вихідним кодом під ліцензією Apache 2.0, розроблену спеціально для створення різноманітних типів графіків та діаграм в мобільних додатках Android. Бібліотека користується високою популярністю серед розробників, що підтверджується понад тридцятьма тисячами зірок на платформі GitHub та активною спільнотою користувачів [17].

Функціональні можливості MPAndroidChart включають підтримку широкого спектру типів візуалізацій для різних сценаріїв відображення даних. Лінійні графіки є основним інструментом для відображення часових рядів вимірювань якості повітря та дозволяють користувачам наочно спостерігати динаміку зміни концентрації забруднюючих речовин, температури, вологості та інших параметрів довкілля протягом різних часових періодів від години до місяця. Стовпчикові діаграми забезпечують ефективне порівняння значень параметрів забруднення з різних датчиків моніторингу або в різні періоди часу, дозволяючи користувачам швидко ідентифікувати найбільш та найменш забруднені локації. Кругові діаграми надають можливість відображення пропорційного розподілу різних типів забруднюючих речовин у складі атмосферного повітря або часток забруднення від різних джерел емісії. Точкові діаграми можуть використовуватися для аналізу кореляцій між різними параметрами довкілля, наприклад, для виявлення залежності між температурою повітря та концентрацією твердих частинок.

Бібліотека забезпечує високий рівень інтерактивності взаємодії користувача з графіками, що значно покращує користувацький досвід та робить процес аналізу даних більш зручним та ефективним. Підтримка мультидотикових жестів дозволяє масштабувати графіки за допомогою pinch-to-zoom для детального вивчення окремих ділянок часового ряду та виявлення короткочасних піків забруднення. Функція плавного прокручування забезпечує зручний перегляд великих обсягів історичних даних без перевантаження оперативної пам'яті пристрою через використання ефективних алгоритмів відрисовки лише видимої частини графіку. Виділення

точок даних при торканні екрану надає користувачам можливість отримати точні числові значення вимірювань для будь-якої точки на графіку без необхідності наближення або інших маніпуляцій. Спливаючі підказки відображають детальну контекстну інформацію про вибране вимірювання, включаючи дату та час вимірювання, значення всіх релевантних параметрів та розрахований індекс якості повітря. Плавні анімації при завантаженні та оновленні даних роблять інтерфейс більш живим та привабливим, покращуючи загальне сприйняття додатку користувачами.

MPAndroidChart спеціально оптимізована для ефективної роботи з великими обсягами даних на мобільних пристроях з обмеженими обчислювальними ресурсами порівняно з настільними комп'ютерами. Бібліотека використовує апаратне прискорення за допомогою технології OpenGL ES для забезпечення плавної роботи навіть при відображенні декількох тисяч точок даних одночасно на одному графіку. Ефективне управління оперативною пам'яттю через використання об'єктних пулів та раціональне кешування запобігає витокам пам'яті та зменшує навантаження на систему збирання сміття. Оптимізовані алгоритми відрисовки мінімізують споживання процесорного часу та енергії батареї, що є критично важливим фактором для мобільних застосунків, оскільки інтенсивні обчислення призводять до швидкого розряджання батареї пристрою.

Мережева взаємодія між мобільним додатком та серверною частиною системи реалізована з використанням бібліотеки RetrofitMe, що є типобезпечним HTTP клієнтом для платформ Android та Java, розробленим компанією Square. Retrofit представляє собою високорівневу абстракцію над HTTP-протоколом, що перетворює програмний інтерфейс REST у чітко типізовані інтерфейси мови програмування через використання анотацій та механізмів рефлексії. Ключові переваги використання Retrofit включають автоматичну серіалізацію об'єктів мови програмування в JSON-формат при формуванні тіла запиту та десеріалізацію JSON-відповідей від сервера в строго типізовані об'єкти при отриманні даних, що повністю усуває необхідність

ручної роботи з JSON та значно зменшує кількість допоміжного коду. Бібліотека забезпечує повну підтримку всіх стандартних HTTP-методів, що визначені специфікацією протоколу, включаючи GET для отримання ресурсів з сервера, POST для створення нових записів, PUT для оновлення існуючих ресурсів, DELETE для видалення даних та PATCH для часткової модифікації ресурсів. Легка інтеграція з механізмом корутин Kotlin дозволяє писати асинхронний мережевий код у синхронному стилі без вкладених зворотних викликів, що значно покращує читабельність та підтримуваність програмного забезпечення. Можливість додавання перехоплювачів запитів надає гнучкість для централізованої реалізації наскрізної функціональності, такої як детальне логування всієї мережевої активності для цілей налагодження, додавання загальних заголовків автентифікації до всіх запитів, обробка типових помилок в єдиному місці та модифікація запитів або відповідей перед їх обробкою основною логікою додатку [18].

Для конвертації даних між JSON-форматом, що використовується для обміну інформацією з сервером через мережу, та об'єктами предметної області мови програмування Kotlin інтегровано бібліотеку Gson Converter. Цей конвертер є офіційним адаптером між бібліотекою Retrofit та бібліотекою Gson від компанії Google, забезпечуючи ефективну та надійну обробку JSON-даних з підтримкою складних структур. Gson автоматично виконує маппінг полів JSON-об'єктів на властивості класів Kotlin на основі відповідності імен або спеціальних анотацій, підтримує складні ієрархічні структури даних з необмеженим рівнем вкладеності, включаючи вкладені об'єкти, масиви та колекції, коректно обробляє nullable поля відповідно до системи типів Kotlin з можливістю явного контролю обов'язковості полів, а також надає розширені можливості для кастомізації процесу серіалізації та десеріалізації через механізм анотацій та адаптерів типів для обробки нестандартних форматів даних [19].

2.5 Інструменти інтерполяції

Локальне зберігання даних на мобільному пристрої реалізовано з використанням вбудованої системи управління базами даних SQLite, що є стандартним рішенням для Android-додатків та не вимагає встановлення додаткових компонентів або залежностей. SQLite представляє собою легковісну реляційну базу даних, що зберігається у вигляді одного файлу на файловій системі пристрою та працює безпосередньо в процесі додатку без необхідності окремого серверного процесу. Основними перевагами SQLite є мінімальне споживання ресурсів (розмір бібліотеки близько 600 КБ), відсутність конфігурації та адміністрування, а також повна підтримка стандарту SQL92 з деякими розширеннями. SQLite підтримує транзакції з гарантіями ACID (Atomicity, Consistency, Isolation, Durability), що забезпечує цілісність даних навіть у випадку несподіваного завершення роботи додатку або вимкнення пристрою.

Для роботи з базою даних було розроблено власний клас DatabaseHelper на мові Java, що розширює стандартний клас SQLiteOpenHelper та реалізує всю логіку створення, оновлення та міграції структури бази даних між різними версіями додатку. SQLiteOpenHelper надає два ключові методи: onCreate(), який викликається при першому створенні бази даних і містить SQL-команди для створення всіх таблиць та індексів, та onUpgrade(), який викликається при збільшенні версії бази даних і відповідає за міграцію існуючих даних до нової структури. Поточна версія бази даних у проєкті встановлена як 2, що відображає одну виконану міграцію, під час якої було додано нове поле для зберігання типу сенсорів. Використання DatabaseHelper як синглтону забезпечує наявність лише одного екземпляра з'єднання з базою даних у межах додатку, що запобігає проблемам з одночасним доступом та конфліктам блокувань [20].

Структура локальної бази даних включає кілька взаємопов'язаних таблиць для зберігання різних типів інформації. Таблиця датчиків (sensors)

зберігає повну інформацію про станції моніторингу, включаючи унікальний ідентифікатор датчика (sensor_id) типу INTEGER з атрибутом PRIMARY KEY, назву станції (station_name) типу TEXT для зберігання довільного текстового опису, географічні координати розташування у форматі широти та довготи (latitude, longitude) типу REAL з точністю до шести знаків після коми для забезпечення точності позиціювання в межах кількох метрів, додаткову інформацію про локацію у текстовому форматі (location_info) типу TEXT, поточний статус активності датчика (is_active) типу INTEGER як булеве значення (0 або 1), час останнього оновлення даних (last_updated) типу INTEGER для зберігання Unix timestamp та тип використовуваних сенсорів (sensor_type) типу TEXT. Всі текстові поля мають обмеження NOT NULL для гарантування наявності даних у критичних полях.

Таблиця вимірювань (measurements) містить історичні дані про якість повітря, отримані з датчиків, включаючи унікальний ідентифікатор вимірювання (measurement_id) типу INTEGER з атрибутом PRIMARY KEY AUTOINCREMENT, зовнішній ключ для зв'язку з таблицею датчиків (sensor_id) типу INTEGER з обмеженням FOREIGN KEY REFERENCES sensors(sensor_id) для забезпечення референційної цілісності, часову мітку вимірювання (timestamp) типу INTEGER у форматі Unix timestamp для ефективного зберігання та порівняння часових даних без необхідності парсингу рядків, значення концентрації твердих частинок PM2.5 (pm25_value) та PM10 (pm10_value) типу REAL у мікрограмах на кубічний метр, температуру повітря (temperature) типу REAL у градусах Цельсія, відносну вологість (humidity) типу REAL у відсотках, атмосферний тиск (pressure) типу REAL у гектопаскалях та розрахований індекс якості повітря (aqi) типу INTEGER за стандартною шкалою від 0 до 500.

Для забезпечення швидкого пошуку та вибірки вимірювань створено складений індекс за ідентифікатором датчика та часовою міткою (CREATE INDEX idx_sensor_timestamp ON measurements(sensor_id, timestamp DESC)), що значно прискорює запити для отримання історії вимірювань конкретного

датчика за визначений період часу. Використання індексу зменшує час виконання типових запитів з O(n) до O(log n), що особливо помітно при роботі з великими обсягами історичних даних. Додатково створено індекс за полем timestamp для швидкого отримання найновіших вимірювань незалежно від датчика. В середньому база даних містить близько 5000-7000 записів вимірювань після тижня активного використання додатку, що підкреслює важливість оптимізації запитів через індексування.

Всі операції запису в базу даних виконуються в межах транзакцій для забезпечення атомарності операцій. Наприклад, при синхронізації даних з сервера спочатку викликається метод beginTransaction(), потім виконується серія операцій INSERT або UPDATE, після чого транзакція підтверджується через setTransactionSuccessful() та закривається через endTransaction(). Якщо під час виконання операцій виникає помилка, транзакція автоматично відкочується, і база даних залишається в консистентному стані. Такий підхід також підвищує продуктивність, оскільки групове виконання операцій в одній транзакції працює значно швидше, ніж окремі автокомітні операції.

DatabaseHelper надає набір методів для виконання CRUD-операцій (Create, Read, Update, Delete): insertSensor() для додавання нового датчика, getSensorById() для отримання інформації про конкретний датчик, getAllSensors() для отримання списку всіх датчиків, updateSensor() для оновлення інформації про датчик, deleteSensor() для видалення датчика, insertMeasurement() для додавання нового вимірювання, getMeasurementsBySensor() для отримання історії вимірювань конкретного датчика за певний період, getLatestMeasurement() для отримання найновішого вимірювання та clearOldMeasurements() для очищення застарілих даних старших за 30 днів для економії місця на пристрої.

Локальна база даних забезпечує реалізацію підходу offline-first, коли додаток розроблений з пріоритетом можливості повноцінної роботи без постійного підключення до Інтернету. Цей архітектурний підхід передбачає, що локальна база даних є первинним джерелом даних для інтерфейсу

користувача, а віддалений сервер використовується для синхронізації та резервного копіювання. Усі дані, отримані з сервера, автоматично кешуються в локальній базі даних через механізм синхронізації, який працює у фоновому режимі при наявності з'єднання з Інтернетом. Процес синхронізації реалізовано таким чином, що спочатку відбувається завантаження нових даних з Firebase, потім порівняння з локальними даними за часовими мітками, і нарешті оновлення локальної бази даних лише новими або змінними записами, що мінімізує трафік та час синхронізації.

Це дозволяє користувачам переглядати попередньо завантажену інформацію про датчики, історичні дані вимірювань та статистику навіть при відсутності мережевого з'єднання. Користувачі можуть аналізувати тренди якості повітря, переглядати графіки та порівнювати показники різних датчиків без очікування завантаження даних з серверу. Це є критично важливим для користувачів з нестабільним інтернет-з'єднанням, обмеженими тарифними планами або тих, хто переглядає додаток у місцях з поганим покриттям мобільної мережі, та забезпечує безперервність роботи додатку в будь-яких умовах експлуатації.

Додатково було реалізовано механізм перевірки актуальності кешованих даних: при наявності інтернет-з'єднання додаток порівнює час останнього оновлення локальних даних з часом останніх змін на сервері, і якщо різниця перевищує 15 хвилин, автоматично ініціюється фонова синхронізація. Користувач може також вручну ініціювати оновлення даних через жест "потягнути вниз" (pull-to-refresh) на головному екрані. Під час синхронізації відображається індикатор прогресу, а після завершення користувач отримує повідомлення про кількість оновлених записів або про те, що дані вже є актуальними.

Для забезпечення продуктивності при роботі з базою даних всі операції читання та запису виконуються в окремих корутинах з використанням Dispatchers.IO, що запобігає блокуванню головного UI-потоку. Результати запитів повертаються до UI-шару через LiveData або StateFlow, що забезпечує

реактивне оновлення інтерфейсу при зміні даних у базі. Такий підхід відповідає рекомендованій архітектурі Android-додатків та забезпечує плавність роботи інтерфейсу навіть при виконанні складних запитів до бази даних.

2.6 Використання хмарної платформи ThingSpeak

В якості джерела даних для системи моніторингу було обрано платформу ThingSpeak, що являє собою відкриту хмарну платформу Інтернету речей, розроблену компанією MathWorks. ThingSpeak надає комплексне рішення для збору, зберігання, аналізу та візуалізації даних з розподілених сенсорних пристроїв через Інтернет. Платформа широко використовується в проєктах громадського моніторингу довкілля завдяки простоті інтеграції, надійності роботи та наявності безкоштовного тарифного плану для некомерційних цілей [21].

ThingSpeak підтримує збір даних з пристроїв Інтернету речей через різні протоколи передачі даних, включаючи HTTP та MQTT, що забезпечує гнучкість підключення датчиків з різними можливостями та обмеженнями. Платформа здійснює автоматичне зберігання часових рядів вимірювань з організацією даних у вигляді каналів, де кожен канал може містити до восьми полів для різних параметрів. Дані зберігаються з часовими мітками та автоматично індексуються для швидкого доступу при запитах за певний період часу.

Ключовою перевагою ThingSpeak для даного проєкту є наявність зручного програмного інтерфейсу REST для читання та запису даних без необхідності автентифікації для публічних каналів. Це дозволяє мобільному додатку безперешкодно отримувати актуальні вимірювання якості повітря з датчиків громадського моніторингу, що публікують свої дані у відкритому доступі. Програмний інтерфейс підтримує гнучкі параметри запитів для

фільтрації даних за часовим діапазоном, обмеження кількості записів у відповіді та вибірки конкретних полів каналу.

Платформа забезпечує високу надійність та доступність сервісу через використання хмарної інфраструктури з географічно розподіленими серверами та автоматичним резервним копіюванням даних. Безкоштовний тарифний план ThingSpeak дозволяє отримувати до трьох мільйонів повідомлень на рік та створювати до чотирьох каналів, чого є цілком достатнім для освітніх та дослідницьких проєктів. Обмеження на частоту оновлення даних, що становить не частіше ніж раз на п'ятнадцять секунд для безкоштовних акаунтів, є прийнятним для моніторингу якості повітря, оскільки концентрація забруднюючих речовин змінюється відносно повільно і не потребує частішого відстеження.

Інтеграція з ThingSpeak реалізована через бібліотеку Retrofit, що забезпечує типобезпечне виконання HTTP-запитів до програмного інтерфейсу платформи. Для кожного публічного каналу датчика моніторингу додаток виконує періодичні запити для отримання останніх вимірювань, які потім зберігаються в локальній базі даних SQLite для офлайн-доступу та відображаються користувачу на карті та у вигляді графіків. Така архітектура забезпечує оптимальний баланс між актуальністю даних та ефективним використанням мережевого трафіку й енергії батареї мобільного пристрою.

Обраний технологічний стек повністю відповідає сучасним стандартам розробки програмного забезпечення для платформи Android та офіційним рекомендаціям компанії Google щодо найкращих практик мобільної розробки. Комбінація офіційних бібліотек від Google з перевіреними open-source рішеннями та надійними хмарними платформами забезпечує високу якість програмного забезпечення, його стабільність в різних умовах експлуатації та довгострокову підтримуваність з можливістю оновлення окремих компонентів без повного переписування системи. Використання стандартизованих компонентів Material Design гарантує створення інтуїтивно зрозумілого інтерфейсу, що відповідає очікуванням користувачів Android-

платформи щодо поведінки та зовнішнього вигляду елементів управління і не вимагає додаткового навчання для ефективного використання системи моніторингу якості атмосферного повітря.

2.7 Висновки

У другому розділі було здійснено постановку задачі розробки інформаційної технології моніторингу якості атмосферного повітря та обґрунтовано вибір оптимальних програмних засобів для її реалізації.

Проведений аналіз сучасного стану ринку мобільних платформ підтвердив доцільність вибору Android як цільової платформи для розробки, що обумовлено домінуючою позицією системи на ринку (71% глобально, 75-80% в Україні), відкритістю платформи та широкими можливостями інтеграції з зовнішніми сервісами. Прийняте рішення про використання нативної розробки замість кросплатформних фреймворків забезпечує максимальну продуктивність при роботі з інтерактивними картами та великими обсягами даних, що є критичним для системи моніторингу.

Вибір технологічного стеку, що включає мови програмування Kotlin та Java, інтегроване середовище Android Studio, систему збірки Gradle, демонструє орієнтацію на сучасні стандарти розробки та офіційні рекомендації Google. Використання Kotlin як основної мови програмування забезпечує високу надійність коду завдяки системі типів з null-safety, лаконічність синтаксису та ефективну підтримку асинхронного програмування через механізм корутин.

Обрані бібліотеки та компоненти повністю покривають функціональні вимоги системи. Картографічний функціонал реалізовано через Google Maps SDK версії та Google Maps Android Utility Library версії, що забезпечують професійне відображення даних на інтерактивній карті з підтримкою теплових карт та кластеризації маркерів. Альтернативна інтеграція OSMDroid версії надає можливість офлайн-роботи та незалежність від комерційних

провайдерів. Візуалізація історичних даних реалізована з використанням бібліотеки MPAndroidChart версії, що забезпечує створення інтерактивних графіків з високою продуктивністю. Мережева взаємодія організована через Retrofit версії з Gson Converter версії для типобезпечної роботи з REST API. Компоненти інтерфейсу базуються на Material Components for Android версії та бібліотеках AndroidX, що гарантує відповідність сучасним стандартам дизайну та зручність використання.

Вибір платформи ThingSpeak як джерела даних IoT обґрунтовано простотою інтеграції, надійністю роботи та наявністю зручного REST API для публічних каналів моніторингу.

Обраний технологічний стек повністю відповідає поставленим задачам, забезпечує високу якість програмного забезпечення, його стабільність та довгострокову підтримуваність, створюючи надійну основу для реалізації системи моніторингу якості атмосферного повітря.

3 РОЗРОБЛЕННЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ МОНІТОРИНГУ ЯКОСТІ АТМОСФЕРНОГО ПОВІТРЯ

3.1 Архітектура системи моніторингу

Архітектурна організація мобільного додатку для моніторингу атмосферного повітря побудована на принципах модульності, розширюваності та чіткого розмеження відповідальності між функціональними компонентами системи. Структурування програмного рішення реалізовано згідно з концепцією багатошарової архітектури, що забезпечує незалежність бізнес-логіки від деталей реалізації користувацького інтерфейсу та механізмів отримання даних з зовнішніх джерел.

Суттєвою характеристикою розробленої системи є комплексний підхід до аналізу екологічних параметрів, що охоплює не лише вимірювання концентрації твердих частинок у повітрі, а й моніторинг супутніх метеорологічних характеристик, які безпосередньо впливають на розповсюдження забруднюючих речовин в атмосфері. Інтеграція різнопланових показників забезпечує формування цілісної картини стану довкілля та виявлення кореляційних залежностей між метеорологічними умовами та рівнем забруднення повітря.

Структурна організація додатку реалізована у вигляді сукупності екранних компонентів, кожен з яких відповідає за виконання чітко визначеного набору функціональних обов'язків. Головний екран додатку, реалізований через клас MainActivity, виконує функцію центрального навігаційного вузла, забезпечуючи користувачам швидкий доступ до всіх підсистем моніторингу через інтуїтивний інтерфейс навігації. Цей компонент інтегрує елементи управління для переходу між різними режимами візуалізації

даних та налаштування параметрів відображення інформації відповідно до потреб конкретного користувача.

Підсистема графічної візуалізації часових рядів екологічних вимірювань представлена компонентом GraphActivity, що забезпечує відображення динаміки зміни п'яти незалежних параметрів довкілля протягом обраного часового інтервалу. На відміну від традиційних рішень, що обмежуються одним або двома типами візуалізації, розроблена система надає користувачам можливість вибору між лінійним представленням даних для детального аналізу трендів та стовпчиковою діаграмою для порівняльного аналізу дискретних значень вимірювань. Така гнучкість візуального представлення інформації дозволяє адаптувати інтерфейс під специфічні завдання аналізу даних, підвищуючи інформативність та зручність роботи з системою.

Важливою складовою архітектури є модуль статистичного аналізу агрегованих показників, реалізований у компоненті GrafstatisticActivity. Даний функціональний блок забезпечує можливість інтерактивного вибору конкретного параметра довкілля з множини доступних показників та автоматичного обчислення трьох основних статистичних характеристик: середнього арифметичного значення, сумарного показника та медіанного значення для обраного часового періоду. Практична значущість такого підходу полягає у можливості оперативного отримання статистично обґрунтованих оцінок стану атмосферного повітря без необхідності залучення зовнішніх інструментів аналітичної обробки даних. Користувач отримує інструментарій для самостійного проведення базового статистичного аналізу безпосередньо в мобільному додатку.

Компонент StatisticActivity забезпечує функціональність швидкого статистичного огляду даних через обчислення та відображення граничних значень вимірювань разом з середнім показником за обраний часовий діапазон. Гнучка система вибору тимчасового вікна аналізу дозволяє користувачам фокусуватися на даних за останню добу, місяць, квартал або весь доступний період спостережень, що надає можливість як оперативного

моніторингу поточного стану, так і ретроспективного аналізу динаміки екологічної ситуації.

Шар мережевої взаємодії архітектурно відокремлено в спеціалізований клас RestApi, що інкапсулює конфігурацію HTTP-клієнта на базі бібліотеки Retrofit для взаємодії з програмним інтерфейсом платформи збору даних Інтернету речей. Використання патерну централізованої конфігурації мережевого клієнта забезпечує уніфікацію параметрів з'єднання та обробки відповідей сервера в єдиній точці системи, що спрощує супровід та модифікацію мережевого рівня додатку.

Інтерфейс програмного доступу до віддаленого сховища даних формалізовано через декларативний опис, що містить специфікацію двадцяти різноманітних методів отримання екологічних вимірювань з гнучкими можливостями параметризації запитів. Розширена функціональність інтерфейсу програмування додатків дозволяє реалізувати складні сценарії вибірки даних з фільтрацією за часовими діапазонами, обмеженням кількості результатів, агрегацією значень через обчислення середніх, сумарних та медіанних показників безпосередньо на серверній стороні, що мінімізує обсяг переданих даних та навантаження на мобільний пристрій.

Важливою складовою системи є модуль проактивного інформування користувачів про небезпечні рівні забруднення повітря, реалізований через клас NotificationHelper. Даний компонент здійснює моніторинг отримуваних вимірювальних даних та автоматичну перевірку їх відповідності санітарно-гігієнічним нормативам для твердих частинок різних фракцій. При виявленні перевищення граничнодопустимих концентрацій система ініціює комплексне сповіщення користувача, що поєднує генерацію системного повідомлення з детальною інформацією про параметр, що перевищив норму, його фактичне значення та часову мітку фіксації, а також активацію тактильного зворотного зв'язку через вібраційний механізм пристрою для привернення уваги користувача навіть при вимкненому звуковому супроводі.

Модель предметної області структуровано в окремий пакет model, що містить класи даних для представлення інформації про канали моніторингу, окремі вимірювання, структуровані відповіді програмного інтерфейсу. Така організація забезпечує чітке розмежування між об'єктами предметної області та компонентами користувацького інтерфейсу, дотримуючись принципу єдиної відповідальності кожного класу системи.

Рівень візуального представлення даних винесено в спеціалізований пакет, що містить адаптери для ефективного відображення списків вимірювань, власні візуальні компоненти для специфічних елементів інтерфейсу та адаптери інформаційних вікон для картографічних маркерів. Використання патерну повторного використання елементів в адаптерах списків забезпечує оптимальну продуктивність при відображенні великих обсягів даних через повторне використання елементів інтерфейсу замість створення нових об'єктів для кожного елемента списку.

Технологічну основу системи складає комбінація мов програмування Kotlin та Java, що дозволяє поєднувати переваги сучасного функціонального підходу Kotlin з надійністю перевірених Java-компонентів. Основний обсяг нового функціоналу реалізовано мовою Kotlin з використанням її можливостей для лаконічного та типобезпечного коду, тоді як компоненти роботи з локальним сховищем даних реалізовано на Java для забезпечення сумісності з існуючими програмними рішеннями.

Управління залежностями та процесом збірки додатку делеговано системі автоматизації Gradle з використанням Kotlin DSL для декларативного опису конфігурації проєкту. Мінімальна підтримувана версія операційної системи встановлена на рівні Android 8.1 Oreo (API Level 27), що забезпечує доступність додатку для значної частини активних пристроїв при збереженні можливості використання сучасних можливостей платформи. Цільова версія SDK встановлена на рівні Android 14 (API Level 34), що забезпечує коректну роботу на сучасних пристроях та відповідність актуальним вимогам безпеки платформи Google Play.

Система дозволів додатку ретельно налаштована для забезпечення необхідного функціоналу при мінімізації вимог до приватності користувачів. Дозвіл на доступ до мережі Інтернет є фундаментальною вимогою для отримання актуальних даних моніторингу з хмарної платформи. Доступ до інформації про стан мережевого з'єднання дозволяє системі адаптувати поведінку залежно від доступності та типу з'єднання. Дозволи на визначення геолокації використовуються картографічним модулем для центрування карти на поточному розташуванні користувача та знаходження найближчих датчиків моніторингу. Право на надсилання сповіщень необхідне для реалізації функціоналу проактивного інформування про небезпечні рівні забруднення. Доступ до вібраційного механізму використовується для посилення привертання уваги користувача при критичних ситуаціях з якістю повітря.

Архітектурне рішення передбачає можливість подальшого розширення функціональності системи через додавання нових модулів аналізу та візуалізації без необхідності модифікації існуючих компонентів. Слабка зв'язаність між шарами додатку та використання інтерфейсів для взаємодії між компонентами забезпечують високу гнучкість системи та спрощують процес тестування окремих модулів незалежно від решти додатку.

На рисунку 3.1 представлено головний екран додатку з табличним представленням даних моніторингу.

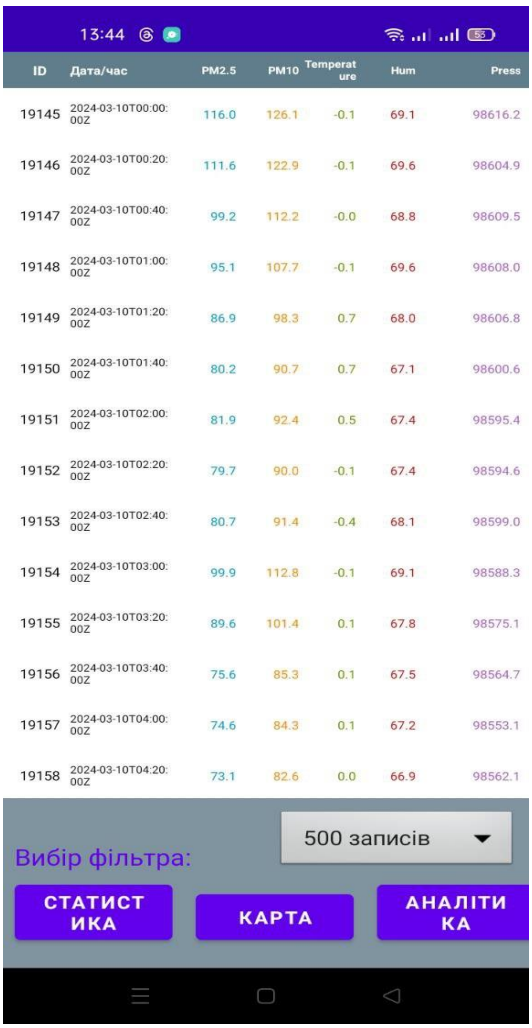

Рисунок 3.1 – Головний екран додатку з табличним представленням даних моніторингу

На рис. 3.2 наведено діаграму прецедентів системи моніторингу атмосферного повітря. Діаграма відображає основних учасників взаємодії (акторів) та функціональні можливості системи.

Користувач взаємодіє із застосунком для отримання даних моніторингу, перегляду агрегованої статистики, візуалізації багатопараметричних графіків та отримання сповіщень про небезпечні рівні забруднення.

Для кожного прецеденту на діаграмі визначено межі відповідальності системи, а також залежності між функціями, що дозволяє сформувати цілісне уявлення про вимоги до системи та її функціональну структуру.

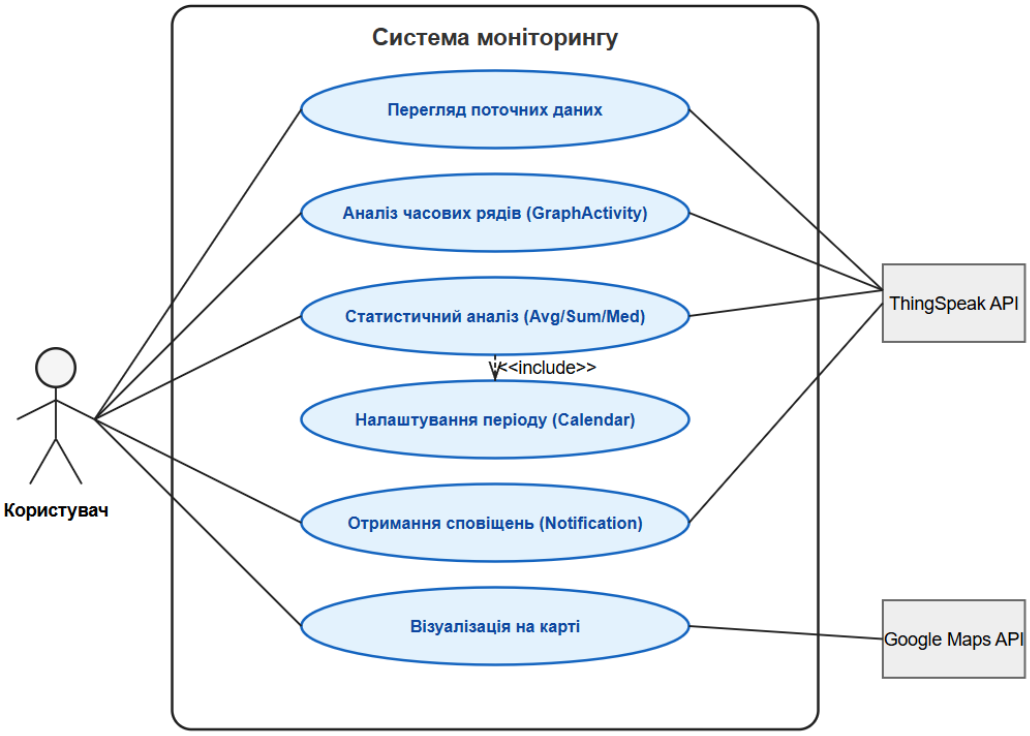

Рисунок 3.2 – UML-діаграма прецедентів системи моніторингу

3.2 Інтеграція з розширеним програмним інтерфейсом платформи моніторингу

Фундаментальною основою функціонування розробленої інформаційної технології є організація ефективної взаємодії з хмарною платформою збору та зберігання даних моніторингу якості атмосферного повітря. На відміну від попередніх реалізацій, що обмежувалися базовою функціональністю отримання необроблених вимірювань, поточне рішення інтегрує розширені можливості серверної аналітичної обробки даних безпосередньо на етапі формування запитів до програмного інтерфейсу.

Центральним елементом мережевого рівня системи виступає інтерфейс FeedAPI, що формалізує специфікацію взаємодії з REST API платформи ThingSpeak через декларативний опис методів доступу до ресурсів. Базовий URL програмного інтерфейсу вказує на кореневий ендпоінт api.thingspeak.com, що забезпечує глобальну доступність сервісу через

розподілену мережу серверів з автоматичним балансуванням навантаження. Ідентифікація цільового каналу даних здійснюється через унікальний числовий ідентифікатор 3123423, що однозначно визначає джерело екологічних вимірювань в екосистемі платформи. Автентифікація запитів реалізована через механізм ключів програмного доступу, що гарантує контрольований доступ до інформації та можливість відстеження використання програмного інтерфейсу.

Важливою особливістю інтегрованого програмного інтерфейсу є підтримка серверних обчислень агрегованих статистичних показників безпосередньо в процесі формування вибірки даних. Метод getStartEndFeedsAverage дозволяє отримувати автоматично розраховані середні значення вимірювань за визначений часовий інтервал з можливістю параметризації тимчасового вікна усереднення. Практична значущість такого підходу полягає у суттєвому скороченні обсягу переданих через мережу даних та зменшенні обчислювального навантаження на мобільний пристрій, оскільки агрегація виконується на серверних ресурсах платформи до передачі результатів клієнтській частині додатку.

Паралельно з обчисленням середніх значень система підтримує отримання сумарних показників через метод getFeedsSum та медіанних значень через getFeedsMedian, що забезпечує комплексний статистичний аналіз розподілу вимірювань. Медіанне значення представляє особливу аналітичну цінність для екологічних даних, оскільки є стійким до викидів та аномальних вимірювань, що часто зустрічаються в реальних умовах експлуатації датчиків через тимчасові збої обладнання або короткочасні локальні джерела забруднення.

Система темпоральної фільтрації даних реалізована через набір методів з різними способами визначення часового діапазону вибірки. Метод getDaysFeeds дозволяє отримувати вимірювання за останні N днів, що доцільно для аналізу короткострокових трендів та оперативного моніторингу поточної ситуації. Функціонал getMinutesFeeds забезпечує доступ до

актуальних даних з хвилинною точністю, що є важливим для відстеження динамічних змін концентрації забруднюючих речовин під час інтенсивних викидів або швидких метеорологічних змін.

Для детального ретроспективного аналізу екологічної ситуації система надає методи з явним визначенням початкової та кінцевої дат періоду аналізу. Метод getStartEndFeedsAverage приймає параметри start та end у форматі ISO 8601, що є міжнародним стандартом представлення дат та часу, забезпечуючи однозначність темпоральної інтерпретації даних між клієнтською та серверною частинами системи. Додатковий параметр average визначає інтервал усереднення в хвилинах, дозволяючи здійснювати баланс між деталізацією отриманих даних та обсягом результуючого набору вимірювань.

Система підтримки обмеження розміру результуючої вибірки реалізована через параметр results у методі getResultsFeeds, що дозволяє отримувати фіксовану кількість найсвіжіших вимірювань незалежно від загального обсягу даних в каналі моніторингу. Такий підхід оптимізує споживання мережевого трафіку та пам'яті мобільного пристрою при необхідності швидкого огляду поточного стану без завантаження повної історії вимірювань, що може налічувати тисячі записів.

Для спеціалізованих аналітичних сценаріїв програмний інтерфейс надає методи з додатковими можливостями фільтрації та збагачення даних. Параметр location активує включення геопросторової інформації про координати розташування датчиків у відповідь сервера, що використовується картографічним модулем для позиціонування маркерів на географічній карті.

Структура даних, що повертається програмним інтерфейсом, формалізована через модель JsonResponse, яка інкапсулює інформацію про канал моніторингу та колекцію окремих вимірювань. Об'єкт каналу містить метадані про джерело даних, включаючи описові назви для п'яти полів вимірювань: концентрації дрібнодисперсних частинок PM2.5, грубодисперсних частинок PM10, температури атмосферного повітря, відносної вологості та атмосферного тиску. Така структуризація дозволяє

системі автоматично формувати інформативні підписи для графіків та елементів інтерфейсу без необхідності жорсткого кодування назв параметрів у клієнтській частині додатку. На рисунках 3.3–3.7 представлено візуалізацію даних моніторингу у веб-інтерфейсі платформи ThingSpeak для кожного з п'яти параметрів.

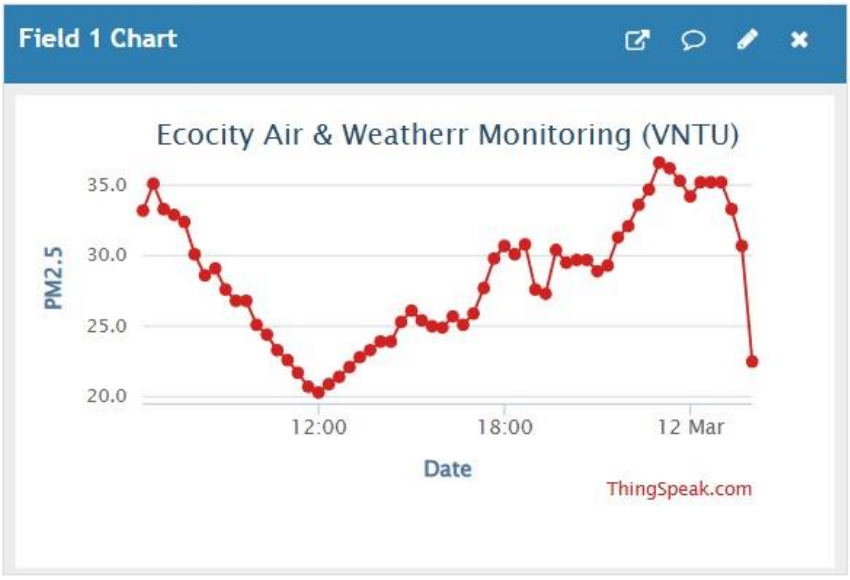

Рисунок 3.3 – Візуалізація концентрації дрібнодисперсних частинок PM2.5 у веб-інтерфейсі ThingSpeak

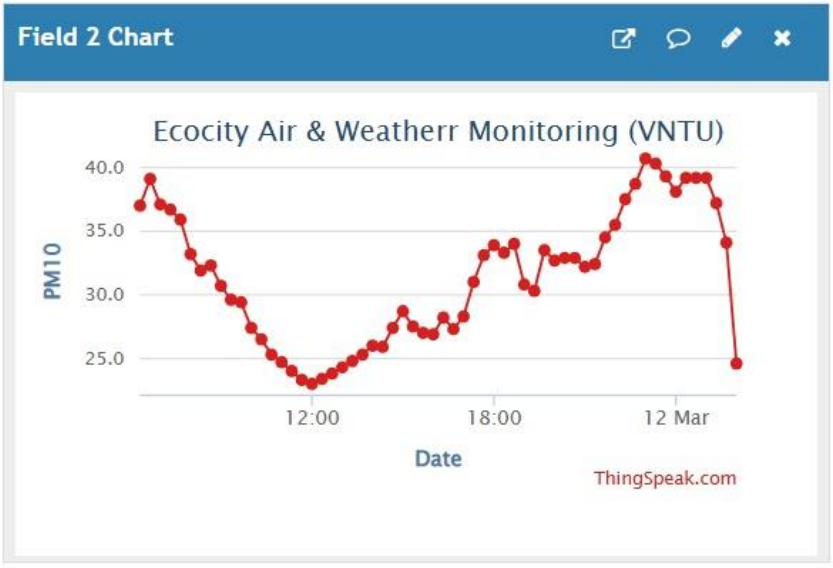

Рисунок 3.4 – Візуалізація концентрації грубодисперсних частинок PM10 у веб-інтерфейсі ThingSpeak

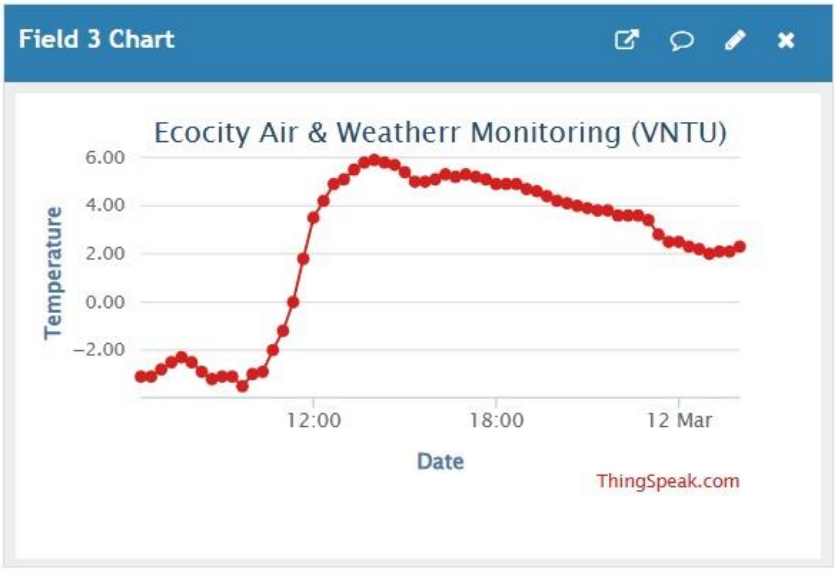

Рисунок 3.5 – Візуалізація температури атмосферного повітря у веб-інтерфейсі ThingSpeak

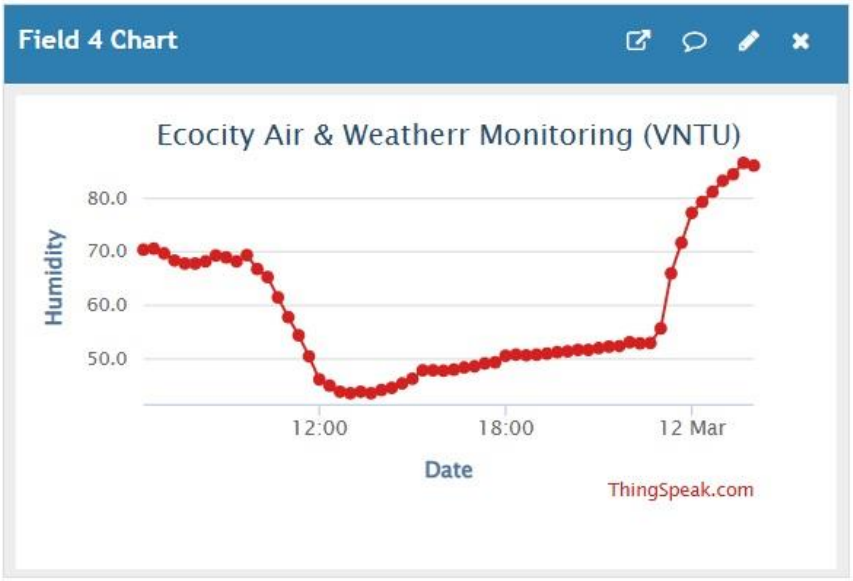

Рисунок 3.6 – Візуалізація відносної вологості повітря у веб-інтерфейсі ThingSpeak

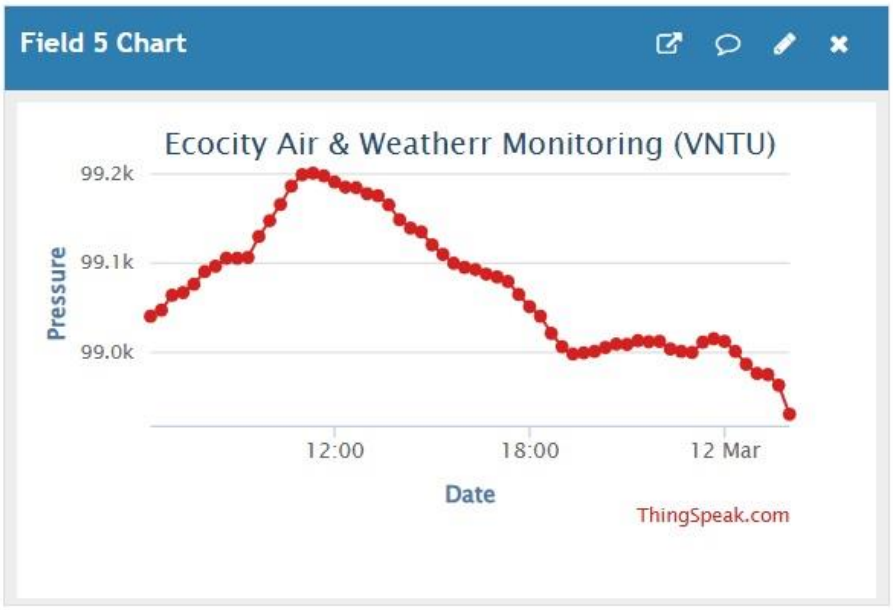

Рисунок 3.7 – Візуалізація атмосферного тиску у веб-інтерфейсі ThingSpeak

Кожне окреме вимірювання представлене об'єктом Feed, що містить унікальний ідентифікатор запису, часову мітку створення у форматі UTC, значення п'яти полів даних та опціональні атрибути геолокації і статусної інформації. Гнучка система nullable-типів мови Kotlin дозволяє коректно обробляти відсутні або некоректні значення окремих полів без ризику аварійного завершення додатку, що критично важливо для роботи з реальними даними сенсорних мереж, де пропуски та аномалії є звичайним явищем через обмеження обладнання та умов експлуатації.

Механізм серіалізації та десеріалізації JSON-структур делеговано бібліотеці Gson через конвертер GsonConverterFactory, інтегрований у конфігурацію Retrofit-клієнта. Автоматичний маппінг між JSON-об'єктами відповідей сервера та строго типізованими класами Kotlin виконується через аналіз назв полів та використання рефлексії, що усуває необхідність ручного розбору JSON-структур та скорочує обсяг допоміжного коду для обробки мережевих відповідей.

Обробка асинхронних мережевих операцій організована через механізм зворотних викликів, що дозволяє виконувати HTTP-запити у фонових потоках виконання без блокування головного потоку інтерфейсу користувача. Кожен

метод програмного інтерфейсу повертає об'єкт Call, що надає методи enqueue для асинхронного виконання запиту з обробкою результату через інтерфейс Callback, або execute для синхронного виклику, який блокує поточний потік до отримання відповіді. В розробленій системі використовується виключно асинхронний підхід для збереження відгуку інтерфейсу навіть при повільному або нестабільному мережевому з'єднанні.

Інтерфейс Callback визначає два обов'язкових методи обробки результату мережевої операції: onResponse викликається при успішному отриманні відповіді від сервера незалежно від HTTP-статусу, що дозволяє обробляти як успішні відповіді з кодом 200, так і помилки клієнта або сервера з кодами 4xx та 5xx; onFailure активується при неможливості встановлення з'єднання з сервером або отримання відповіді через мережеві проблеми, що вимагає реалізації логіки повторних спроб або інформування користувача про тимчасову недоступність даних.

Комплексний підхід до обробки помилкових ситуацій включає аналіз HTTP-статусу відповіді через властивість response.code(), перевірку наявності тіла відповіді через response.body(), обробку виключень при десеріалізації JSON-структур та реагування на мережеві таймаути. Така багаторівнева система виявлення та обробки помилок забезпечує стабільність роботи додатку навіть в умовах проблем з мережевим з'єднанням або тимчасової недоступності серверних ресурсів.

Конфігурація HTTP-клієнта інкапсульована в класі RestApi, що забезпечує централізоване управління параметрами мережевої взаємодії. Базовий URL програмного інтерфейсу встановлюється через метод baseUrl() при побудові екземпляра Retrofit, гарантуючи, що всі запити автоматично направляються на правильний ендпоінт без необхідності дублювання адреси в кожному методі інтерфейсу. Фабрика конвертерів додається через метод addConverterFactory(), забезпечуючи автоматичне перетворення між JSON-представленням та об'єктами предметної області.

Розширені можливості програмного інтерфейсу включають фільтрацію результатів за числовими діапазонами через методи getMinFeeds та getMaxFeeds, що дозволяють отримувати лише вимірювання, що перевищують або не досягають певного порогового значення. Такий функціонал критично важливий для швидкої ідентифікації періодів з небезпечними рівнями забруднення без необхідності завантаження та локальної фільтрації великих обсягів даних на мобільному пристрої.

На рисунку 3.8 наведено UML-діаграму класів, що відображає архітектурну структуру застосунку для моніторингу параметрів атмосферного повітря. Діаграма демонструє взаємозв’язки між основними компонентами системи, які організовано у три логічні рівні: UI Layer, Network Layer та Model Layer.

У UI-рівні зосереджені активності, що відповідають за взаємодію з користувачем та відображення даних.

Клас MainActivity виконує роль стартового компонента застосунку та містить метод setupNavigation(), який забезпечує ініціалізацію навігації між екранами.

Клас GrafStatisticActivity відповідає за формування статистичних графіків та містить методи updateAverageChart(), updateSumChart() та updateMedianChart(). Він також взаємодіє зі спадаючим списком параметрів (spinner: Spinner).

Клас GraphActivity відповідає за безпосереднє побудування лінійних і стовпчикових графіків за допомогою компонента MPAndroidChart.

Допоміжний клас NotificationHelper забезпечує функціональність надсилання сповіщень про небезпечні рівні забруднення. Він містить методи checkDustLevels() та sendDustAlert() і використовує системний сервіс Vibrator.

Network Layer реалізує взаємодію із зовнішнім API за допомогою бібліотеки Retrofit.

Клас RestApi містить базову URL-адресу сервера та метод getService(), який повертає інтерфейс FeedAPI.

Інтерфейс FeedAPI описує набір HTTP-запитів, необхідних для отримання агрегованих даних: середніх значень (getStartEndFeedsAverage()), сум (getFeedsSum()), медіан (getFeedsMedian()), добових значень (getDaysFeeds()) та загальних результатів (getResultsFeeds()).

Model Layer включає класи, що відповідають за десеріалізацію JSON-даних, отриманих від API.

Клас JsonResponse містить інформацію про канал та список об’єктів типу Feed. Клас Feed описує окремий запис моніторингу та включає ідентифікатор (entry_id: Long) і набір параметрів (field1…5).

Зв’язок між JsonResponse і Feed реалізовано у вигляді композиції, що означає, що об’єкти Feed існують лише в межах однієї відповіді API.

Таким чином, наведена діаграма відображає цілісну структуру програмної системи, демонструє принципи розподілу відповідальності між компонентами та забезпечує зрозумілу модель взаємодії між UI-логікою, мережею та моделями даних.

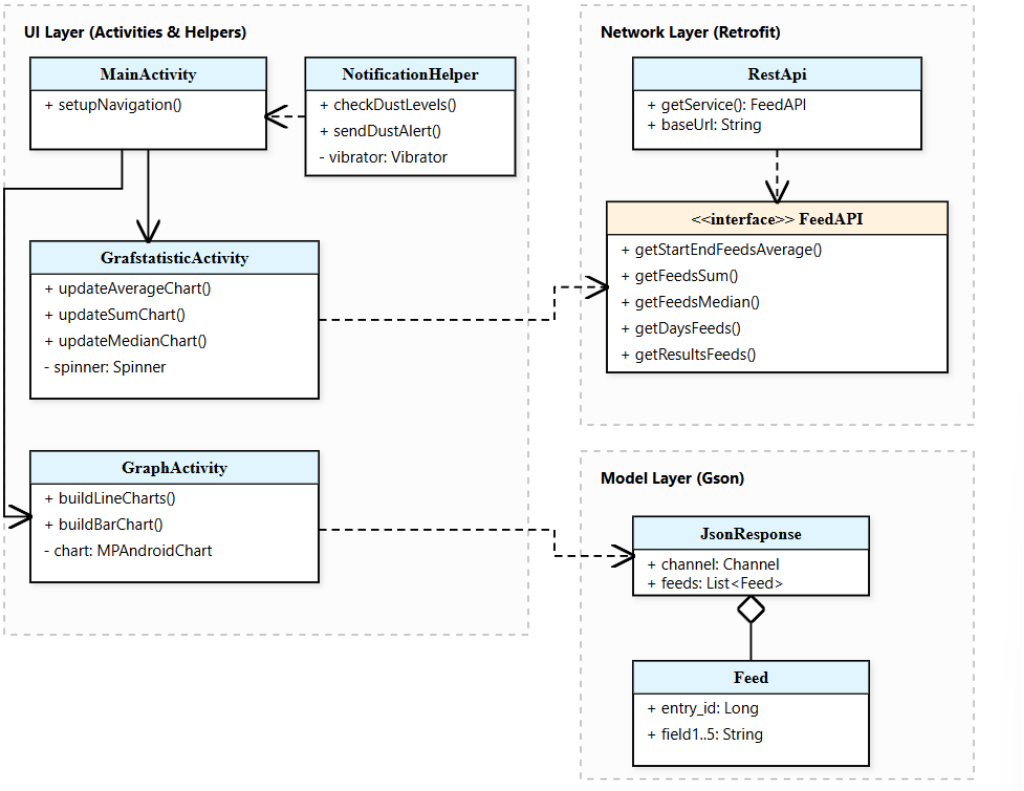

Рисунок 3.8 – UML-діаграма класів програмної системи моніторингу атмосферного повітря

Параметр timescale у методі getTimescaleFeeds дозволяє регулювати щільність вибірки вимірювань через автоматичне проріджування даних на серверній стороні, що корисно при відображенні довгострокових трендів, коли кожне окреме вимірювання не має критичної інформативної цінності, а важливіша загальна динаміка зміни показників протягом тривалого періоду.

Інтеграція з розширеним програмним інтерфейсом платформи моніторингу створює технологічну основу для реалізації складних аналітичних сценаріїв безпосередньо в мобільному додатку без необхідності розгортання власної серверної інфраструктури або виконання ресурсоємних обчислень на обмежених апаратних ресурсах мобільного пристрою. Делегування статистичної обробки на сервер дозволяє зберегти енергію батареї та мінімізувати споживання мережевого трафіку, що є важливим фактором для забезпечення якісного користувацького досвіду мобільних додатків. На рисунку 3.9 представлено інтерфейс вибору кількості записів для завантаження.

Рисунок 3.9 – Інтерфейс вибору кількості записів для завантаження з сервера

3.3 Модуль інтерактивного вибору параметрів та обчислення агрегованих статистичних характеристик

Важливою складовою розробленої інформаційної технології є реалізація модуля багатокритеріального статистичного аналізу екологічних даних через компонент GrafstatisticActivity, що надає користувачам високий рівень контролю над процесом обробки та інтерпретації вимірювальної інформації. Відмінність даного підходу від традиційних систем моніторингу полягає у делегуванні користувачу повноважень щодо вибору конкретного аналізованого параметра довкілля та способу його агрегації, замість використання фіксованого набору попередньо визначених звітних форм.

Користувацький інтерфейс модуля організовано навколо концепції інтерактивних елементів вибору, що дозволяють формувати персоналізовані сценарії аналізу даних без необхідності програмування або конфігурації складних параметрів фільтрації. Центральним елементом управління виступає випадаючий список параметрів довкілля, реалізований через компонент Spinner, що надає доступ до п'яти незалежних вимірюваних характеристик атмосферного повітря. Перший параметр представляє концентрацію дрібнодисперсних твердих частинок фракції PM2.5, що мають діаметр менше двох з половиною мікрометрів і здатні проникати глибоко в легеневу тканину, становлячи найбільшу загрозу для здоров'я людини. Другий параметр відображає концентрацію грубодисперсних частинок PM10 діаметром до десяти мікрометрів, що затримуються в верхніх дихальних шляхах. Третім параметром є температура атмосферного повітря, що безпосередньо впливає на швидкість хімічних реакцій формування вторинних забруднювачів та інтенсивність вертикального перемішування повітряних мас. Четвертий параметр - відносна вологість повітря, що визначає ефективність осадження твердих частинок та їх агрегацію у більші конгломерати. П'ятим параметром

виступає атмосферний тиск, що корелює з метеорологічними умовами розсіювання забруднень.

Після вибору цільового параметра аналізу система надає інструменти для визначення часового діапазону досліджуваного періоду через інтегрований календарний віджет CalendarView, що забезпечує візуальний спосіб специфікації дат без необхідності ручного введення числових значень. Особливістю реалізації є підтримка визначення як початкової, так і кінцевої дати періоду аналізу через єдиний календарний компонент з перемиканням режиму його роботи. Додатковий випадаючий список дозволяє користувачу вибрати, чи поточне використання календаря призначене для встановлення стартової чи фінішної часової межі, забезпечуючи гнучкість інтерфейсу при мінімізації кількості візуальних елементів на екрані.

Додатковим параметром налаштування аналізу виступає інтервал часової агрегації даних, що визначає, з якою частотою виконуватиметься вибірка вимірювань з повного набору доступних даних. Система підтримує вибір інтервалів від десяти хвилин до однієї години, дозволяючи балансувати між деталізацією результатів та обчислювальною складністю обробки великих масивів даних. Короткі інтервали агрегації надають більш детальну картину короткострокових флуктуацій параметрів довкілля, тоді як довші періоди усереднення згладжують випадкові коливання та виявляють стійкі тренди зміни екологічної ситуації.

Аналітична обробка даних реалізована через паралельне обчислення трьох фундаментальних статистичних характеристик розподілу значень обраного параметра. Перша характеристика - середнє арифметичне значення, що обчислюється через запит getStartEndFeedsAverage з параметрами початкової та кінцевої дат разом з інтервалом усереднення. Цей показник надає базову оцінку типового рівня параметра протягом аналізованого періоду, проте є чутливим до наявності екстремальних значень-викидів у даних.

Друга характеристика - сумарний показник, що обчислюється через метод getFeedsSum програмного інтерфейсу. Практична інтерпретація сумарного значення залежить від специфіки параметра: для концентрацій забруднювачів сума відображає кумулятивну експозицію організму протягом періоду, для метеорологічних параметрів цей показник має обмежену практичну цінність та використовується переважно для технічних цілей валідації даних.

Третя характеристика - медіанне значення, що отримується через виклик getFeedsMedian і представляє значення, що поділяє впорядкований ряд спостережень навпіл. Медіана є робастною статистичною оцінкою центральної тенденції, стійкою до впливу аномальних вимірювань та викидів, що робить її особливо цінною для аналізу екологічних даних з датчиків громадського моніторингу, де якість вимірювань може бути нестабільною.

Візуалізація результатів статистичного аналізу здійснюється через три незалежні стовпчикові діаграми BarChart, кожна з яких відображає просторово-часову динаміку однієї зі статистичних характеристик. Використання окремих графіків для кожного показника, замість їх комбінування на єдній діаграмі, обумовлено значними відмінностями в порядках величин різних статистик, що унеможливлює їх ефективне спільне представлення на графіку з єдиною вертикальною віссю без втрати інформативності.

Кожен графік налаштовано з підтримкою інтерактивного масштабування через жести pinch-to-zoom, що дозволяє користувачам детально досліджувати окремі ділянки часового ряду для виявлення короткочасних аномалій або тонких закономірностей у даних. Можливість вертикального та горизонтального масштабування забезпечує гнучкість візуального аналізу як окремих значень, так і загальних трендів зміни показників.

Стовпчики діаграми анотовано числовими значеннями, що відображаються безпосередньо над кожним елементом з точністю до двох

десяткових знаків. Такий підхід поєднує переваги візуального сприйняття трендів через графічне представлення з можливістю точного кількісного аналізу через доступ до числових значень без необхідності розшифровки положення стовпчика відносно координатної сітки.

Асинхронна природа мережевих операцій отримання статистичних даних вимагає ретельної організації обробки відповідей для забезпечення відгуку інтерфейсу користувача. Кожен з трьох статистичних запитів виконується незалежно через власний екземпляр Callback, що дозволяє паралельно отримувати різні агрегації без очікування завершення попередніх операцій. При успішному отриманні відповіді від сервера дані автоматично трансформуються у формат, придатний для відображення на графіках, через методи updateAverageDataChart, updateSumChart та updateMedianChart відповідно.

Трансформація серверних даних у візуальні елементи графіків включає вилучення значення обраного параметра з кожного об'єкта вимірювання через універсальний метод getFieldValue, що приймає індекс параметра та повертає відповідне поле з моделі даних. Така архітектурна абстракція дозволяє уникнути дублювання коду обробки для кожного з п'яти параметрів, централізуючи логіку вибірки даних в єдиному місці системи.

Обробка помилкових ситуацій при мережевій взаємодії та відсутності даних для обраного періоду вирішується через механізм зворотних викликів onFailure, що дозволяє інформувати користувача про тимчасову недоступність аналітичних функцій без аварійного завершення роботи додатку. Система коректно обробляє ситуації порожніх відповідей або некоректних значень параметрів через використання nullable-типів та безпечних операторів доступу Kotlin.

Реалізований модуль інтерактивного аналізу забезпечує розширення функціональності мобільного додатку від пасивного інструменту відображення попередньо сформованих звітів до активної аналітичної платформи, що надає користувачам інструменти для самостійного

дослідження даних моніторингу відповідно до їхніх специфічних інформаційних потреб та аналітичних завдань.

На рисунку 3.10 представлено інтерфейс вибору параметрів та часового діапазону для статистичного аналізу

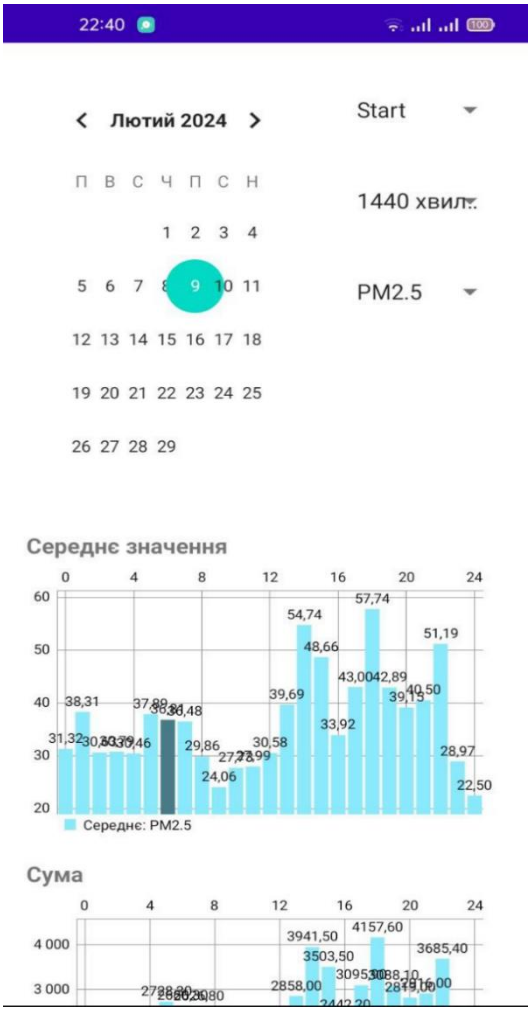

Рисунок 3.10 – Інтерфейс вибору параметрів та часового діапазону для статистичного аналізу

На рисункуна рисунку 3.11 – результати обчислення статистичних характеристик.

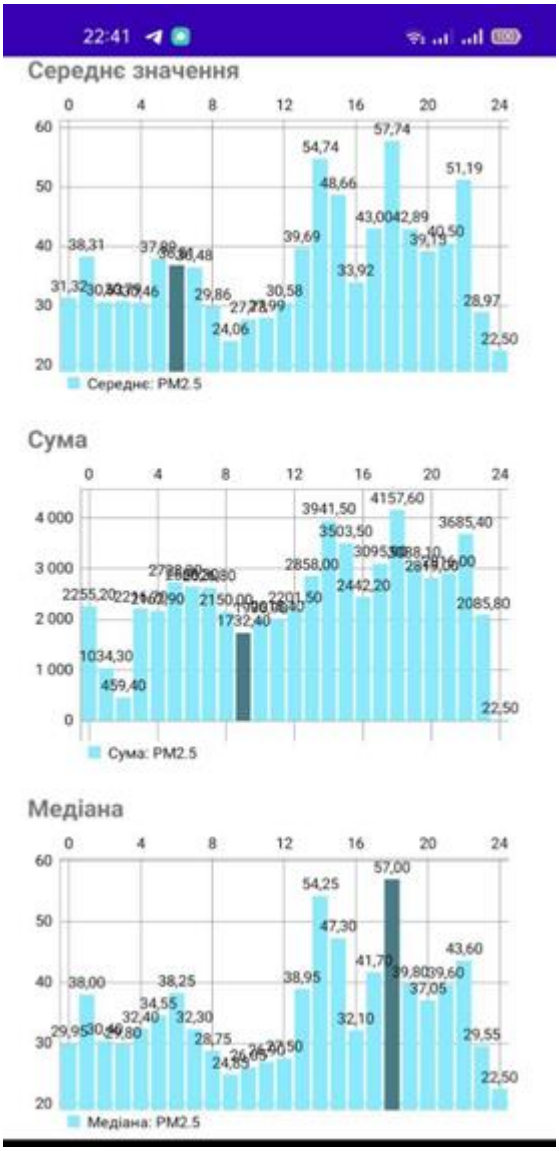

Рисунок 3.12 – Візуалізація статистичних характеристик: середнє значення, сума та медіана

3.4 Розширена система багато параметричної візуалізації даних моніторингу

Візуалізація часових рядів екологічних вимірювань є важливою функціональністю інформаційної технології моніторингу, оскільки саме через графічне представлення даних користувачі формують розуміння динаміки зміни стану атмосферного повітря та виявляють закономірності, що

залишаються прихованими при аналізі табличних або числових форм представлення інформації. Розроблена система візуалізації реалізує комплексний підхід до графічного представлення п'яти різнопланових параметрів довкілля через інтеграцію бібліотеки MPAndroidChart версії 3.1.0, що надає інструментарій створення інтерактивних графіків для мобільних додатків Android.

Архітектурна організація модуля візуалізації втілена в компоненті GraphActivity, що забезпечує одночасне відображення динаміки п'яти незалежних параметрів моніторингу через систему паралельних лінійних графіків. Перший графік відображає часову еволюцію концентрації дрібнодисперсних частинок PM2.5, другий - грубодисперсних частинок PM10, третій - температури атмосферного повітря, четвертий - відносної вологості, п'ятий - атмосферного тиску. Така організація дозволяє користувачам одночасно спостерігати кореляції між різними параметрами та виявляти причинно-наслідкові зв'язки між метеорологічними умовами та рівнями забруднення.

Додатковим режимом візуалізації виступає стовпчикова діаграма BarChart, що надає альтернативне представлення тих самих даних у формі, придатному для порівняльного аналізу дискретних значень замість безперервних трендів. Перемикання між лінійним та стовпчиковим режимами відображення реалізовано через випадаючий список Spinner, що дозволяє користувачам обирати найбільш інформативний спосіб візуалізації залежно від специфіки аналітичної задачі.

Побудова лінійних графіків ініціюється методом buildLineCharts, що приймає об'єкт JsonResponse з повним набором вимірювань та виконує їх трансформацію у структури даних, придатні для візуалізації через MPAndroidChart. Першим етапом обробки є сортування вимірювань за ідентифікатором запису entry_id для забезпечення правильного хронологічного порядку відображення точок на графіку, оскільки сервер може повертати дані в довільному порядку залежно від специфіки запиту.

Для кожного з п'яти параметрів формується окремий список пар значень, де перший елемент пари представляє часову мітку у числовому форматі, а другий - виміряне значення параметра. Конвертація текстової мітки часу з формату ISO 8601 у числове представлення виконується методом convertStringDateToFloat, що розбирає рядок дати через SimpleDateFormat та перетворює отримане значення Unix timestamp у хвилини від епохи для забезпечення компактного представлення часової осі на графіку.

Трансформація списків пар у об'єкти LineDataSet виконується універсальним методом lineDataSet, що приймає дані та текстову мітку параметра для створення налаштованого набору даних графіка. Метод конфігурує візуальні характеристики лінії графіка, включаючи колір, товщину, стиль точок даних та параметри заливки області під кривою. Використання пунктирного стилю лінії через enableDashedLine додає візуальну відмінність між графіками різних параметрів та покращує читабельність при одночасному відображенні кількох кривих.

Налаштування осей координат графіків виконується через метод setupLineChart, що конфігурує параметри відображення горизонтальної та вертикальної осей. Горизонтальна вісь X використовує власний форматер DateAxisValueFormatter для перетворення числових значень хвилин від епохи назад у зрозумілу текстову форму дати у форматі YYYY-MM-DD, що забезпечує інтуїтивну інтерпретацію часової шкали користувачами. Вертикальні осі використовують LargeValueFormatter для форматування числових значень з додаванням розділювачів тисяч, що покращує читабельність великих чисел на графіках.

Побудова стовпчикової діаграми через метод buildBarChart реалізує аналогічну логіку трансформації даних, але з використанням об'єктів BarEntry замість Entry та BarDataSet замість LineDataSet. Ключова відмінність стовпчикового представлення полягає у використанні порядкового індексу вимірювання як координати X замість часової мітки, що спрощує візуальне порівняння окремих значень без необхідності інтерпретації часової осі.

Налаштування ширини стовпчиків через властивість barData.barWidth дозволяє контролювати щільність розміщення візуальних елементів на діаграмі, балансуючи між інформативністю відображення окремих значень та можливістю охоплення широкого часового діапазону на обмеженій площі екрану мобільного пристрою. Встановлене значення 0.3 забезпечує достатні проміжки між стовпчиками для чіткого візуального розділення при збереженні компактності представлення.

Інтерактивність графіків забезпечується через вбудовані можливості MPAndroidChart для обробки жестів користувача. Підтримка масштабування через pinch-to-zoom дозволяє користувачам інтерактивно збільшувати окремі ділянки графіків для детального вивчення короткочасних флуктуацій або аномальних значень. Плавне прокручування дозволяє навігувати по широких часових діапазонах даних без втрати контексту поточної позиції на загальній шкалі часу.

Обробка відсутніх або некоректних значень у вихідних даних реалізована через використання nullable-типів Kotlin та фільтрації списків перед конвертацією в об'єкти Entry. Метод convertToEntries виконує трансформацію списку пар у список Entry з автоматичним виключенням записів з null-значеннями, що запобігає аварійному завершенню додатку при зустрічі пропусків у часових рядах вимірювань.

Асинхронне завантаження даних для візуалізації організоване через стандартний механізм Retrofit з обробкою відповідей у методі onResponse. Використання CoroutineScope з Dispatchers.IO для виконання трансформації даних та runOnUiThread для оновлення графіків забезпечує неблокуючу обробку великих обсягів вимірювань без зависання інтерфейсу користувача, що критично важливо для підтримки відгуку додатку при роботі з тисячами точок даних.

Обробка помилкових ситуацій відсутності даних реалізована через перевірку порожнечі списку feeds у відповіді сервера з відображенням інформативного повідомлення Toast для користувача про тимчасову

недоступність вимірювань. Така обробка запобігає відображенню порожніх графіків та забезпечує чітку комунікацію проблем з отриманням даних.

Реалізована система візуалізації надає користувачам інструменти для сприйняття складних багатопараметричних часових рядів екологічних вимірювань через інтерактивні графіки, що поєднують інформативність наукової візуалізації даних з простотою використання мобільних додатків. На рисунку 3.13 представлено екран візуалізації п'яти параметрів моніторингу у вигляді лінійних графіків.

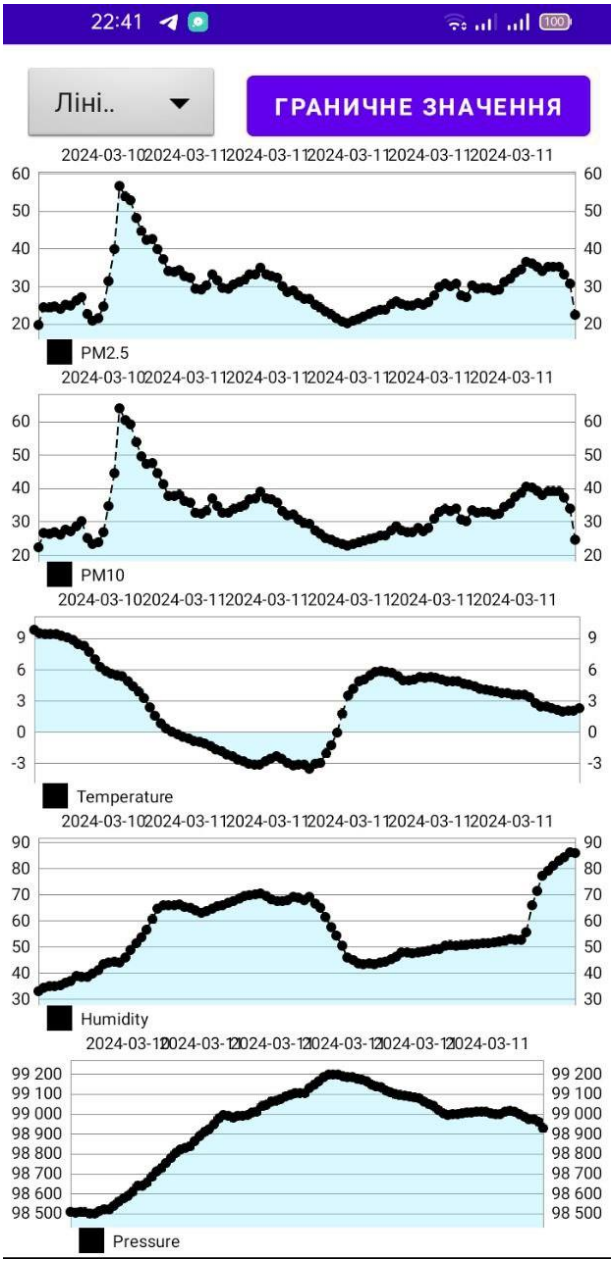

Рисунок 3.13 – Візуалізація п'яти параметрів моніторингу у вигляді лінійних графіків

3.5 Система проактивного інформування про небезпечні рівні забруднення атмосферного повітря

Важливою складовою розробленої інформаційної технології є реалізація системи автоматичного виявлення та проактивного інформування користувачів про перевищення санітарно-гігієнічних нормативів концентрації твердих частинок в атмосферному повітрі. Відмінність такого підходу від пасивних систем моніторингу, що вимагають від користувача самостійного періодичного перегляду показників для виявлення небезпечних ситуацій, полягає у делегуванні функції контролю якості повітря автоматизованій підсистемі, що оперативно сповіщає про виникнення загроз здоров'ю без необхідності активної участі користувача.

Технічна реалізація системи сповіщень інкапсульована в спеціалізованому класі NotificationHelper, що концентрує всю логіку створення каналів нотифікацій, перевірки порогових значень параметрів та генерації багатомодальних повідомлень для привернення уваги користувача. Архітектурна організація передбачає ініціалізацію helper при створенні компонентів додатку з автоматичним налаштуванням каналу нотифікацій для забезпечення готовності системи до миттєвого реагування на виявлення критичних ситуацій.

Створення каналу нотифікацій реалізовано методом createNotificationChannel, що виконується в конструкторі класу та налаштовує параметри системного каналу для Android версій 8.0 та вище, де каналізація сповіщень є обов'язковою вимогою платформи. Канал конфігурується з рівнем важливості IMPORTANCE_HIGH для забезпечення пріоритетного відображення повідомлень незалежно від поточного режиму роботи пристрою. Активація вібрації через enableVibration та світлової індикації через enableLights забезпечує багатомодальне привертання уваги користувача навіть при вимкненому звуковому супроводі або заблокованому екрані пристрою.

Критерії класифікації рівнів забруднення як небезпечних базуються на науково обґрунтованих порогових значеннях граничнодопустимих концентрацій, визначених міжнародними організаціями охорони здоров'я. Для дрібнодисперсних частинок PM2.5 встановлено поріг п'ятдесят мікрограмів на кубічний метр, перевищення якого асоціюється з підвищеним ризиком респіраторних та серцево-судинних захворювань. Для грубодисперсних частинок PM10 встановлено поріг сто мікрограмів на кубічний метр відповідно до рекомендацій Всесвітньої організації охорони здоров'я щодо допустимих рівнів експозиції населення забрудненому повітрю.

Автоматична перевірка дотримання нормативів реалізована методом checkDustLevels, що приймає значення трьох полів вимірювань разом з часовою міткою та виконує послідовну верифікацію кожного параметра на відповідність встановленим пороговим значенням. Використання безпечних операторів Kotlin для обробки nullable-типів через toDoubleOrNull дозволяє коректно обробляти ситуації відсутності даних або некоректних значень без ризику аварійного завершення через NullPointerException.

При виявленні перевищення порогового значення система ініціює генерацію комплексного сповіщення через метод sendDustAlert, що приймає фактичне значення параметра, його тип та дату вимірювання для формування інформативного повідомлення. Структура нотифікації включає короткий заголовок з емодзі-символом попередження для візуального привертання уваги, стислий текст з основною інформацією про перевищений параметр та його значення, а також розгорнутий текст через BigTextStyle з детальними рекомендаціями щодо обмеження перебування на вулиці для мінімізації впливу забрудненого повітря на здоров'я.

Інтеграція з системою навігації додатку реалізована через PendingIntent, що дозволяє користувачу безпосередньо з нотифікації перейти до головного екрану додатку для перегляду детальної інформації про поточну екологічну ситуацію. Налаштування прапорців FLAG_ACTIVITY_NEW_TASK та FLAG_ACTIVITY_CLEAR_TASK забезпечує очищення стеку активностей

при переході з нотифікації для уникнення створення складних ланцюжків навігації.

Тактильний зворотний зв'язок реалізовано через налаштування патерну вібрації через метод setVibrate з масивом часових інтервалів, що визначає послідовність увімкнення та вимкнення вібромотора. Встановлений патерн longArrayOf(0, 500, 200, 500) створює відчутну вібраційну послідовність з паузою, що ефективно привертає увагу користувача навіть при розташуванні пристрою в кишені або сумці.

Пріоритизація нотифікацій через встановлення PRIORITY_HIGH гарантує відображення сповіщення навіть при встановленому режимі економії енергії або обмеженнях фонової активності додатків, що критично важливо для системи екологічного моніторингу, де затримка інформування про небезпечні умови може мати серйозні наслідки для здоров'я користувачів.

Автоматичне закриття нотифікації після взаємодії користувача реалізовано через прапорець setAutoCancel, що очищує панель сповіщень від оброблених повідомлень та запобігає захараженню інтерфейсу застарілими нотифікаціями про вже минулі епізоди забруднення. Унікальний ідентифікатор нотифікації NOTIFICATION_ID дозволяє оновлювати існуюче повідомлення при виявленні нових перевищень замість створення окремих сповіщень для кожного випадку.

Реалізована система проактивного інформування забезпечує розширення функціональності мобільного додатку моніторингу від пасивного інструменту перегляду даних до активного захисного механізму, що автономно відстежує стан атмосферного повітря та оперативно попереджає користувачів про виникнення загроз їх здоров'ю, дозволяючи приймати своєчасні рішення щодо зміни поведінки або переміщення в менш забруднені локації для мінімізації експозиції шкідливим речовинам. На рисунку 3.14 представлено приклад сповіщення про перевищення граничнодопустимої концентрації твердих частинок

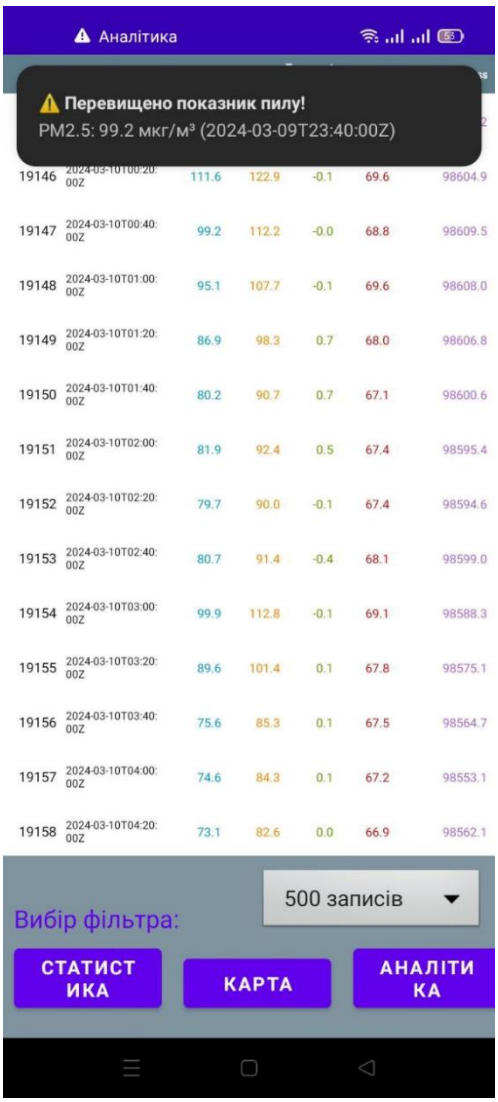

Рисунок 3.14 – Сповіщення про перевищення граничнодопустимої концентрації PM2.5

3.6 Реалізація користувацького інтерфейсу та принципів юзабіліті

Проектування користувацького інтерфейсу мобільного додатку базується на фундаментальних принципах дизайн-системи Material Design, що визначає єдині стандарти візуального представлення, поведінки елементів управління та організації інформаційної архітектури для екосистеми Android-додатків. Дотримання встановлених конвенцій платформи забезпечує інтуїтивну зрозумілість інтерфейсу для користувачів через використання знайомих патернів взаємодії та очікуваної поведінки стандартних компонентів без необхідності додаткового навчання роботі з додатком.

Візуальна ідентичність додатку базується на інтеграції бібліотеки Material Components for Android версії що надає повний набір готових компонентів інтерфейсу з вбудованою підтримкою всіх візуальних ефектів Material Design, включаючи elevation-тіні для створення відчуття глибини, ripple-ефекти при торканні для тактильного зворотного зв'язку та плавні анімації переходів між станами для органічного відчуття відгуку системи.

Колористична схема додатку визначена через систему тематизації Material, що дозволяє централізовано управляти кольоровою палітрою через конфігурацію theme. Первинний колір визначає основні акцентні елементи інтерфейсу, вторинний - додаткові інтерактивні компоненти, кольори поверхонь задають фон для контентних областей, а колір помилок використовується для індикації критичних станів та попереджень. Така систематизація забезпечує візуальну узгодженість усіх екранів додатку та спрощує підтримку альтернативних тем, включаючи темний режим для комфортного використання в умовах низького освітлення.

Типографічна система базується на шрифтовій родині Roboto, що є стандартним типографічним рішенням для платформи Android та оптимізована для читабельності на екранах різних роздільностей. Використання шкали типографічних стилів Material через стильові атрибути textAppearanceHeadline, textAppearanceBody, textAppearanceCaption забезпечує правильну візуальну ієрархію інформації та покращує сканованість контенту через чітке розрізнення заголовків, основного тексту та додаткових підписів.

Адаптивність макетів до різних розмірів екранів реалізована через використання системи ConstraintLayout версії, що дозволяє створювати складні багатоелементні інтерфейси з плоскою ієрархією View-компонентів через систему взаємних обмежень між елементами. Використання відносного позиціонування замість абсолютних координат забезпечує коректне відображення інтерфейсу на пристроях з різними роздільностями екранів,

аспектними співвідношеннями та густиною пікселів без необхідності створення окремих варіантів макетів.

Навігаційна архітектура додатку організована через систему окремих Activity для кожного функціонального модуля з передачею контекстної інформації через Intent-extras при переходах між екранами. Кнопки навігації розміщені в логічних позиціях відповідно до стандартних патернів Material Design з верхньою панеллю додатку для глобальних дій та нижньою областю для первинних функціональних елементів управління.

Зворотний зв'язок про стан виконання операцій реалізовано через Toast-повідомлення для короткострокових нотифікацій про результати дій та індикатори прогресу для тривалих операцій завантаження даних. Обробка помилкових ситуацій комунікується через діалогові вікна з чітким поясненням проблеми та варіантами подальших дій для користувача.

Доступність інтерфейсу для користувачів з обмеженнями забезпечується через достатній контраст кольорів для читабельності тексту, розміри тач-таргетів не менше сорока восьми пікселів для зручного натискання та підтримку навігації через системні жести Android для інтеграції з асистивними технологіями.

Реалізований користувацький інтерфейс забезпечує баланс між функціональною насиченістю системи моніторингу та простотою використання через дотримання встановлених платформних конвенцій, інтуїтивну організацію інформаційної архітектури та візуальну узгодженість всіх елементів інтерфейсу відповідно до принципів Material Design.

3.7 Висновки

У третьому розділі здійснено детальний опис процесу розробки інформаційної технології моніторингу якості атмосферного повітря для мобільної платформи Android з реалізацією розширеного функціоналу

багатопараметричного аналізу екологічних даних та проактивного інформування користувачів про небезпечні рівні забруднення.

Розроблена архітектура системи базується на принципах модульності та чіткого розмеження відповідальності між функціональними компонентами, що забезпечує високу підтримуваність та можливість подальшого розширення функціональності без необхідності модифікації існуючого коду. Структурна організація додатку реалізована через систему спеціалізованих Activity-компонентів, кожен з яких відповідає за виконання чітко визначеного набору функцій від візуалізації часових рядів до статистичного аналізу агрегованих показників.

Важливою складовою розробленої системи є інтеграція з розширеним програмним інтерфейсом платформи ThingSpeak, що надає двадцять різноманітних методів доступу до даних моніторингу з підтримкою серверних обчислень статистичних характеристик. Делегування агрегації даних на серверну сторону дозволило суттєво скоротити обсяг мережевого трафіку та обчислювальне навантаження на мобільний пристрій при збереженні повної функціональності аналітичної обробки екологічних вимірювань.

Реалізований модуль інтерактивного вибору параметрів та обчислення агрегованих характеристик надає користувачам високий рівень контролю над процесом аналізу даних через можливість самостійного визначення досліджуваного параметра довкілля, часового діапазону аналізу та інтервалу агрегації з автоматичним обчисленням та візуалізацією середніх, сумарних та медіанних значень для обраної конфігурації.

Інтеграція картографічного функціоналу на базі Google Maps SDK забезпечила просторову контекстуалізацію екологічних вимірювань, дозволяючи користувачам візуально співвідносити показники якості повітря з географічним розташуванням датчиків та власним місцезнаходженням для прийняття обґрунтованих рішень щодо планування переміщень в умовах неоднорідного просторового розподілу забруднення атмосфери.

Система візуалізації на основі бібліотеки MPAndroidChart реалізує графічне представлення п'яти незалежних параметрів довкілля через паралельні лінійні графіки з підтримкою альтернативного стовпчикового представлення, інтерактивного масштабування та детального аналізу окремих ділянок часових рядів для виявлення короткострокових аномалій та довгострокових трендів зміни екологічної ситуації.

Система проактивного інформування про небезпечні рівні забруднення забезпечує розширення функціональності мобільного додатку від пасивного інструменту перегляду даних до активного захисного механізму, що автономно відстежує стан атмосферного повітря та оперативно попереджає користувачів про перевищення санітарно-гігієнічних нормативів через багатомодальні сповіщення з вібраційним зворотним зв'язком та рекомендаціями щодо мінімізації експозиції шкідливим речовинам.

Реалізований користувацький інтерфейс забезпечує баланс між функціональною насиченістю системи та простотою використання через дотримання принципів Material Design, що забезпечує зрозумілість для користувачів без необхідності додаткового навчання роботі з додатком.

Розроблена інформаційна технологія повністю реалізує поставлені функціональні вимоги щодо багатопараметричного моніторингу якості атмосферного повітря з розширеними можливостями статистичного аналізу, геопросторової візуалізації та проактивного інформування користувачів, створюючи технологічну основу для підвищення екологічної свідомості населення та сприяння прийняттю обґрунтованих рішень щодо захисту здоров'я в умовах забрудненого довкілля.

4 ЕКОНОМІЧНА ЧАСТИНА

4.1 Оцінювання комерційного потенціалу розробки

Метою комерційного та технологічного аудиту є оцінювання комерційного потенціалу інформаційної технології моніторингу стану атмосферного повітря за даними громадського моніторингу з використанням мобільних пристроїв, а також визначення доцільності впровадження мобільного застосунку та можливостей його подальшої комерціалізації..

Для проведення технологічного аудиту було залучено 3-х незалежних експертів Вінницького національного технічного університету кафедри системного аналізу та інформаційних технолгій: к.т.н., доц. Горячев Г.В., к.т.н., доц. Козачко О. М., к.т.н., доц. Варчук І. В. Для проведення технологічного аудиту було використано таблицю 4.1 [23]. в якій за п’ятибальною шкалою використовуючи 12 критеріїв здійснено оцінку комерційного потенціалу.

Таблиця 4.1 – Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Кри-терій	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів

Продовження таблиці 4.1

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Кри-терій	0	1	2	3	4
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві

Продовження таблиці 4.1

Кри-терій	0	1	2	3	4
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки пові-домлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Таблиця 4.2 – Рівні комерційного потенціалу розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0-10	Низький
11-20	Нижче середнього
21-30	Середній
31-40	Вище середнього
41-48	Високий

У таблиці 4.3 подано результати експертного оцінювання комерційного потенціалу розробки, які відображають її привабливість для практичного впровадження та ринкового використання. Узагальнені висновки експертів характеризують доцільність інвестування, перспективи масштабування та конкурентоспроможність запропонованої інформаційної технології. Отримані оцінки дають змогу сформувати об’єктивне уявлення про економічну доцільність подальшого розвитку проєкту.

Таблиця 4.3 – Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта		
	Горячев Г.В.	Козачко О.М.	Варчук І.В.
	Бали, виставлені експертами:		
1	4	2	3
2	3	2	2
3	3	4	4
4	4	4	2
5	2	3	4
6	1	2	3
7	2	1	1
8	3	3	3
9	2	4	1
10	3	4	3
11	4	3	4
12	2	4	3
Сума балів	$СБ_1$=33	$СБ_2$=36	$СБ_3$=33
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_1^3 СБ_i}{3} = \frac{33 + 36 + 33}{3} = 34$		

Середньоарифметична сума балів, розрахована на основі висновків експертів склала 34 бали, що згідно таблиці 4.2 вважається, що рівень комерційного потенціалу проведених досліджень є вище середнього.

Інформаційні технології аналізу та прогнозування якості атмосферного повітря у м. Вінниці за даними громадського моніторингу, а саме програма, яка вибирає кращі місця для створення нових пунктів моніторингу буде цікава міській громаді, а також іншим містам у яких є потреба оцінити якість повітря та визначити місця для очищення повітря та покращення навколишнього середовища в місті.

4.2 Прогнозування витрат на виконання науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи групуються за такими статтями: витрати на оплату праці, витрати на соціальні заходи,

матеріали, паливо та енергія для науково-виробничих цілей, витрати на службові відрядження, програмне забезпечення для наукових робіт, інші витрати, накладні витрати.

1. Основна заробітна плата кожного із дослідників $З_О$, якщо вони працюють в наукових установах бюджетної сфери визначається за формулою:

$$З_О = \frac{М}{Т_р} * t \text{ (грн)}, \quad (4.1)$$

де $М$ – місячний посадовий оклад конкретного розробника (інженера, дослідника, науковця тощо), грн.;

$Т_Р$ – число робочих днів в місяці; приблизно $Т_Р \approx 21...23$ дні;

t – число робочих днів роботи дослідника.

Для розробки програмні засоби необхідно залучити програміста з посадовим окладом 15000 грн. Кількість робочих днів у місяці складає 30, а кількість робочих днів програміста складає 22. Зведемо сумарні розрахунки до таблиця 4.4.

Таблиця 4.4 – Заробітна плата дослідника в науковій установі бюджетної сфери

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату грн.
Керівник	25000	1136,4	5	5682
Програмний інженер	15000	681,8	30	20454
Всього				26136

2. Розрахунок додаткової заробітної плати робітників

Додаткова заробітна плата $З_д$ всіх розробників та робітників, які приймали устають в розробці нового технічного рішення розраховується як 10 - 12 % від основної заробітної плати робітників.

На даному підприємстві додаткова заробітна плата начисляється в розмірі 10% від основної заробітної плати.

$$З_д = (З_о + З_р) * \frac{Н_{дод}}{100\%} \quad (4.2)$$

$$З_д = 0{,}11 * 26136 = 2875 \text{ (грн)}.$$

3. Нарахування на заробітну плату $Н_{ЗП}$ дослідників та робітників, які брали участь у виконанні даного етапу роботи, розраховуються за формулою:

$$Н_{ЗП} = (З_о + З_д) * \frac{\beta}{100}, \quad (4.3)$$

де $З_о$ – основна заробітна плата розробників, грн.;

$З_д$ – додаткова заробітна плата всіх розробників та робітників, грн.;

β – ставка єдиного внеску на загальнообов'язкове державне соціальне страхування, % .

Дана діяльність відноситься до бюджетної сфери, тому ставка єдиного внеску на загальнообов'язкове державне соціальне страхування буде складати 22%, тоді**:**

$$Н_{ЗП} = (26136 + 2875) * \frac{22}{100} = 6382{,}42 \text{ (грн)}.$$

4. Витрати на комплектуючі вироби, які використовують при виготовленні одиниці продукції, розраховуються, згідно їх номенклатури, за формулою:

$$К = \sum_{i=1}^{n} Н_i * Ц_i * К_i , \quad (4.5)$$

де $Н_i$ – кількість комплектуючих *i*-го виду, шт.;

$Ц_i$ – покупна ціна комплектуючих *i*-го найменування, грн.;

$К_i$ – коефіцієнт транспортних витрат (1,1…1,15).

Таблиця 4.5 – Комплектуючі, що використані на розробку

Найменування матеріалу	Ціна за одиницю, грн.	Витрачено	Вартість витраченого матеріалу, грн.
Папір	150	1	150
Ручка	30	1	30
Маркер	40	1	40
Флешка	200	1	200
Всього			420
З врахуванням коефіцієнта транспортування			451

5. Програмне забезпечення для наукової роботи включає витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення необхідного для проведення дослідження.

Для написання магістерської роботи використовувалися інтернет середовище Android Studio та ThingSpeak online, які є безкоштовними.

6. Амортизація обладнання, комп’ютерів та приміщень, які використовувались під час виконання даного етапу роботи

Дані відрахування розраховують по кожному виду обладнання, приміщенням тощо.

$$A = \frac{Ц * Т}{Т_{кор} * 12}, \quad (4.6)$$

де Ц – балансова вартість даного виду обладнання (приміщень), грн.;

$Т_{кор}$ – час користування;

Т – термін використання обладнання (приміщень), цілі місяці.

Згідно пункта 137.3.3 Податкового кодекса амортизація нараховується на основні засоби вартістю понад 2500 грн. В нашому випадку для написання магістерської роботи використовувався персональний комп'ютер вартістю 30000 грн.

$$\text{А} = \frac{30000 \cdot 1}{2 \cdot 12} = 1250 \text{ (грн)}.$$

7. До статті «Паливо та енергія для науково-виробничих цілей» відносяться витрати на всі види палива й енергії, що безпосередньо використовуються з технологічною метою на проведення досліджень.

$$\text{В}_\text{е} = \sum_{\text{i}=1}^{n} \frac{W_{yt} \cdot t_i \cdot \text{Ц}_\text{е} \cdot \text{К}_{\text{впі}}}{\eta_\text{i}}, \qquad (4.7)$$

де W_{yt} – встановлена потужність обладнання на певному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$\text{Ц}_\text{е}$ – вартість 1 кВт-години електроенергії, грн;

$\text{К}_{\text{впі}}$ – коефіцієнт, що враховує використання потужності, $\text{К}_{\text{впі}} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_\text{i} < 1$.

Для написання магістерської роботи використовується персональний комп'ютер для якого розрахуємо витрати на електроенергію.

$$\text{В}_\text{е} = \frac{0{,}3 \cdot 185 \cdot 9{,}56 \cdot 0{,}5}{0{,}8} = 331{,}61 \text{ (грн)}.$$

Витрати на службові відрядження, витрати на роботи, які виконують сторонні підприємства, установи, організації та інші витрати в нашому досліджені не враховуються оскільки їх не було.

Накладні (загальновиробничі) витрати Внзв охоплюють: витрати на управління організацією, оплата службових відряджень, витрати на

утримання, ремонт та експлуатацію основних засобів, витрати на опалення, освітлення, водопостачання, охорону праці тощо. Накладні (загальновиробничі) витрати Внзв можна прийняти як (100…150)% від суми основної заробітної плати розробників та робітників, які виконували дану МКНР, тобто:

$$В_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (4.8)$$

де $Н_{нзв}$ – норма нарахування за статтею «Інші витрати».

$$В_{нзв} = 26136 \cdot \frac{100}{100\%} = 26136 \text{ (грн)}.$$

Сума всіх попередніх статей витрат дає витрати, які безпосередньо стосуються даного розділу МКНР

$$В = 26136+2875+6382,42+451+1250+331,61+26136=63562,03 \text{ (грн)}.$$

Прогнозування загальних втрат ЗВ на виконання та впровадження результатів виконаної МКНР здійснюється за формулою:

$$ЗВ = \frac{В}{\eta}, \quad (4.9)$$

де η – коефіцієнт, який характеризує стадію виконання даної НДР.

Оскільки, робота знаходиться на стадії науково-дослідних робіт, то коефіцієнт β = 0,9.

Звідси:

$$ЗВ = \frac{63562,03}{0,9} = 70624,5 \text{ (грн)}.$$

4.3 Розрахунок економічної ефективності науково-технічної розробки

У даному підрозділі кількісно спрогнозуємо, яку вигоду, зиск можна отримати у майбутньому від впровадження результатів виконаної наукової роботи. Розрахуємо збільшення чистого прибутку підприємства $\Delta П_i$, для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, за формулою

$$\Delta П_i = \sum_1^n (\Delta Ц_o * N * Ц_o * \Delta N)_i * \lambda * \rho * \left(1 - \frac{\upsilon}{100}\right), \quad (4.10)$$

де $\Delta Ц_0$ – покращення основного оціночного показника від впровадження результатів розробки у даному році.

N – основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

ΔN – покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки:

$Ц_o$ – основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки:

λ – коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт λ = 0,8333.

ρ – коефіцієнт, який враховує рентабельність продукту. ρ = 0,25;

υ – ставка податку на прибуток. У 2025 році – 18%.

Припустимо, що ціна за програмний продукт зросте на 700 грн. Кількість одиниць реалізованої продукції також збільшиться: протягом першого року на 65 шт., протягом другого року – на 45 шт., протягом третього року на 35 шт. Реалізація продукції до впровадження розробки складала 1 шт., а її ціна до

складає 9500 грн. Розрахуємо прибуток, яке отримає підприємство протягом трьох років.

$$\Delta П_1 = [700 \cdot 1 + (9500 + 700) \cdot 65] \cdot 0{,}833 \cdot 0{,}25 \cdot \left(1 + \frac{18}{100}\right) = 163\,021{,}60 \text{ (грн)}.$$

$$\Delta П_2 = [700 \cdot 1 + (9500 + 700) \cdot (65 + 45)] \cdot 0{,}833 \cdot 0{,}25 \cdot \left(1 + \frac{18}{100}\right) = 275\,717{,}33 \text{ (грн)}.$$

$$\Delta П_3 = [700 \cdot 1 + (9500 + 700) \cdot (65 + 45 + 35)] \cdot 0{,}833 \cdot 0{,}25 \cdot \left(1 + \frac{18}{100}\right) = 363\,413{,}06 \text{ (грн)}.$$

4.4 Розрахунок ефективності вкладених інвестицій та періоду їх окупності

Розрахуємо основні показники, які визначають доцільність фінансування наукової розробки певним інвестором, є абсолютна і відносна ефективність вкладених інвестицій та термін їх окупності.

Розрахуємо величину початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки.

$$PV = k_{\text{інв}} \cdot \text{ЗВ}, \qquad (4.11)$$

де $k_{\text{інв}}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо ($k_{\text{інв}}$ = 2…5).

$$PV = 2 \cdot 70624{,}5 \ = 141249 \text{ (грн)}.$$

Розрахуємо абсолютну ефективність вкладених інвестицій $E_{абс}$ згідно наступної формули:

$$\text{Е}_{\text{абс}} = (\text{ПП} - PV), \tag{4.12}$$

де ПП – приведена вартість всіх чистих прибутків, що їх отримає підприємство від реалізації результатів наукової розробки, грн.;

$$\text{ПП} = \sum_1^T \frac{\Delta\text{П}_\text{і}}{(1+\tau)^t}\ , \tag{4.13}$$

де $\Delta\text{П}_\text{і}$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДЦКР, грн.;

Т – період часу, протягом якою виявляються результати впровадженої НДДКР, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,2;

t – період часу (в роках).

$$\text{ПП} = \frac{163\ 021{,}60}{(1+0{,}2)^1} + \frac{275\ 717{,}33}{(1+0{,}2)^2} + \frac{363\ 413{,}06}{(1+0{,}2)^3} = 537624{,}48 \text{ (грн)}.$$

$$\text{Е}_{\text{абс}} = (537624{,}48 - 141249) = 396375{,}48 \text{ (грн)}.$$

Оскільки $\text{Е}_{\text{абс}} > 0$, то вкладання коштів на виконання та впровадження результатів НДДКР може бути доцільним.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $E_в$. Для цього користуються формулою:

$$E_в = \sqrt[Т_ж]{1 + \frac{E_{абс}}{PV}} - 1\,, \qquad (4.14)$$

де $Т_ж$ – життєвий цикл наукової розробки, роки.

$$E_в = \sqrt[3]{1 + \frac{396375{,}48}{141249,}} - 1 = 0{,}56 = 56\%.$$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою:

$$\tau = d + f\,, \qquad (4.15)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2025 році в Україні d = (0,14…0,2);

f – показник, що характеризує ризикованість вкладень; зазвичай, величина $f = (0{,}05\ldots0{,}1)$.

$$\tau_{min} = 0{,}18 + 0{,}5 = 0{,}23.$$

Так як $E_в > \tau_{min}$ то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій за формулою:

$$Т_{ок} = \frac{1}{E_в}\,. \qquad (4.16)$$

$$Т_{ок} = \frac{1}{0{,}56} = 1{,}10 \text{ (роки)}.$$

Так як $T_{ок} \leq 3...5$-ти років, то фінансування даної наукової розробки в принципі є доцільним.

4.5 Висновки

Проведено оцінку комерційного потенціалу інформаційної технології моніторингу стану атмосферного повітря за даними громадського моніторингу з використанням мобільних пристроїв

Прогнозування витрат на виконання науково-дослідної роботи по кожній з статей витрат складе 63562,03 грн. Загальна ж величина витрат на виконання та впровадження результатів даної НДР буде складати 70624,5 грн.

Вкладені інвестиції в даний проект окупляться через 1,10 роки, приведена вартість всіх чистих прибутків, що їх отримає підприємство від реалізації результатів наукової розробки склала 537624,48 грн.

ВИСНОВКИ

Проблема забезпечення контролю стану атмосферного повітря є однією з найважливіших екологічних задач сучасності, адже підвищення концентрації шкідливих речовин у повітрі безпосередньо впливає на здоров’я населення та якість життя. Упродовж останніх років усе більшого значення набувають системи громадського моніторингу, які забезпечують регулярний збір даних про забруднення повітря за допомогою доступних сенсорних станцій. На відміну від традиційних стаціонарних постів, такі станції формують густішу мережу спостереження та дозволяють отримувати більш деталізовану інформацію щодо стану атмосфери в конкретних локаціях. Отримані дані автоматично передаються на сервер та використовуються для інформування населення про поточний рівень забруднення. У цьому контексті важливу роль відіграють мобільні застосунки, основним завданням яких є зручне відображення результатів моніторингу та оперативне повідомлення користувачів про перевищення гранично допустимих концентрацій.

Перший розділ був присвячений аналізу об’єкта дослідження — процесу екологічного моніторингу в міських умовах. Було визначено ключові проблеми традиційних державних систем контролю, зокрема недостатню територіальну щільність офіційних станцій та відсутність оперативного інформування населення. Аналіз міжнародних і вітчизняних платформ (IQAir, AirNow, EEA, Sensor.Community, SaveEcoBot, LUN Air, Air Lviv) продемонстрував відсутність комплексного рішення, що орієнтоване на українських користувачів та інтегровано з локальними джерелами громадського моніторингу. Це засвідчило актуальність створення нової інформаційної технології, яка поєднує відкриті екологічні дані, мобільний інтерфейс та сучасні засоби візуалізації.

У другому розділі було сформульовано постановку задачі та обґрунтовано технологічні рішення для розробки системи. Вибір Android як цільової платформи визначено її домінуванням на ринку, відкритістю,

високою продуктивністю та широкими можливостями інтеграції з API. Нативний підхід до розробки забезпечив максимально ефективну роботу з інтерактивними картами та великими масивами даних. Обраний стек технологій (Kotlin, Android Studio, Gradle, Retrofit, Gson Converter, MPAndroidChart, Google Maps SDK, OSMDroid, AndroidX, Material Components) гарантує стабільність, підтримуваність та відповідність сучасним практикам розробки мобільних застосунків. Платформа ThingSpeak визначена як оптимальне джерело IoT-даних завдяки надійному REST API та підтримці статистичних серверних обчислень.

У третьому розділі детально описано процес реалізації мобільної інформаційної технології моніторингу довкілля. Побудована архітектура системи базується на принципах модульності та розмежування відповідальності, що дозволяє легко масштабувати та розширювати функціонал. Інтеграція з API ThingSpeak забезпечила доступ до широкого набору методів з обчисленням середніх, сумарних і медіанних характеристик на сервері, що зменшує навантаження на мобільний пристрій. Реалізований модуль інтерактивного аналізу дає змогу користувачам обирати параметри довкілля, часові діапазони та інтервали агрегації, отримуючи автоматично обчислені статистичні показники та їх графічне представлення

У четвертому розділі проведено оцінку комерційного потенціалу інформаційної технології моніторингу стану атмосферного повітря за даними громадського моніторингу з використанням мобільних пристроїв

Прогнозування витрат на виконання науково-дослідної роботи по кожній з статей витрат складе 63562,03 грн. Загальна ж величина витрат на виконання та впровадження результатів даної НДР буде складати 70624,5 грн.

Вкладені інвестиції в даний проект окупляться через 1,10 роки, приведена вартість всіх чистих прибутків, що їх отримає підприємство від реалізації результатів наукової розробки склала 537624,48 грн.

Для захисту інтелектуальної власності та забезпечення можливості комерціалізації технології подано документи на отримання свідоцтва про

реєстрацію авторського права на комп'ютерну програму до Державної служби інтелектуальної власності України.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. World Health Organization. WHO global air quality guidelines: particulate matter (PM2.5 and PM10), ozone, nitrogen dioxide, sulfur dioxide and carbon monoxide. Geneva: WHO, 2021. 300 p. ISBN 978-92-4-003422-8. (дата звернення: 12.11.2025).

1. Універсальна інформаційно-вимірювальна система оперативного екологічного моніторингу з використанням мобільних пристроїв Мокін В. Б., Дзюняк Д. Ю., Горячев Г. В., Бондалєтов К. О. Універсальна інформаційно-вимірювальна система оперативного екологічного моніторингу з використанням мобільних пристроїв / В. Б. Мокін, Д. Ю. Дзюняк, Г. В. Горячев, К. О. Бондалєтов // Вісник Вінницького політехнічного інституту. – 2015. – № 5. – С. 116-122. (дата звернення: 13.11.2025).

2. IQAir. World Air Quality Report 2023 URL:https://www.iqair.com/world-air-quality-report (дата звернення: 13.11.2025).

3. IQAir. Air Quality Index (AQI) Basics – URL:https://www.iqair.com/air-quality-index (дата звернення: 13.11.2025).

4. US EPA. Air Quality Index: A Guide to Air Quality and Your Health URL:https://www.airnow.gov/aqi/aqi-basics/ (дата звернення: 14.11.2025).

5. BreezoMeter. Air Quality API Documentation URL: https://docs.breezometer.com/ (дата звернення: 15.11.2025).

6. European Environment Agency. European Air Quality Index – URL: https://www.eea.europa.eu/themes/air/air-quality-index (дата звернення: 15.11.2025).

7. Sensor.Community. Open Environmental Data with Citizen Science – URL: https://sensor.community/ (дата звернення 16.11.2025)

8. SaveEcoBot. Екологічний моніторинг України URL: https://www.saveecobot.com/ (дата звернення 16.11.2025)

9. LUN Air. Моніторинг якості повітря в Києві Режим доступу: https://air.kiev.ua/. (дата звернення: 16.11.2025).

10. Air Lviv. Громадський моніторинг якості повітря у Львові. URL: https://airlviv.org/ (дата звернення: 17.11.2025).

11. Android Developers. Android Platform Architecture.URL: https://developer.android.com/guide/platform (дата звернення: 17.11.2025).

12. Material Design Guidelines. Google – URL: https://material.io/design (дата звернення: 18.11.2025).

13. Jemerov D., Isakova S. Kotlin in Action. Manning Publications, 2017. 360 p. ISBN 978-1617293290. (дата звернення: 18.11.2025)

14. Leiva A. Kotlin for Android Developers: Learn Kotlin the easy way while developing an Android App. Leanpub, 2019. 210 p. (дата звернення: 19.11.2025).

15. Android Studio User Guide URL: https://developer.android.com/studio/intro (дата звернення: 20.11.2025).

16. Google Maps Platform Documentation. Google Developers –URL: https://developers.google.com/maps/documentation/android-sdk (дата звернення: 21.11.2025).

17. MPAndroidChart: A powerful Android chart view library URL: https://github.com/PhilJay/MPAndroidChart (дата звернення 22.11.2025)

18. Retrofit: A type-safe HTTP client for Android and Java–URL: https://square.github.io/retrofit/ (дата звернення 23.11.2025)

19. Gson User Guide. Google .URL: https://github.com/google/gson/blob/master/UserGuide.md (дата звернення: 23.11.2025).

20. SQLite Documentation URL: https://www.sqlite.org/docs.html (дата звернення 26.11.25)

21. .ThingSpeak Documentation. MathWorks URL: https://www.mathworks.com/help/thingspeak/ (дата звернення 26.11.2025)

22. Козловський В. О., Лесько О. Й., Кавецький В. В. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт : уклад. Вінниця : ВНТУ, 2021. 42 с.

23. 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

Додаток А

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
«___» _____ 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ
на магістерську кваліфікаційну роботу
«ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ МОНІТОРИНГУ СТАНУ АТМОСФЕРНОГО ПОВІТРЯ З ВИКОРИСТАННЯМ МОБІЛЬНИХ ПРИСТРОЇВ»
08-34.МКР.002.02.000.ТЗ

Керівник: к.т.н., доц. каф. САІТ
_____ Георгій ГОРЯЧЕВ
«___» _____ 2025 р.

Розробив: студент гр. 2ІСТ-24м
_____ Сергій ДЖУРА
«___» _____ 2025 р.

Вінниця 2025

1. Підстава для проведення робіт

Підставою для виконання роботи є наказ № __ по ВНТУ від «__» _____ 2025 р., та індивідуальне завдання на МКР, затверджене протоколом № __ засідання кафедри САІТ від «__» _____ 2025 р.

2. Джерела розробки:

- Універсальна інформаційно-вимірювальна система оперативного екологічного моніторингу з використанням мобільних пристроїв [Текст] / В. Б. Мокін, Д. Ю. Дзюняк, Г. В. Горячев, К. О. Бондалєтов // Вісник Вінницького політехнічного інституту. – 2015. – № 5. – С. 116-122.

3. Мета і призначення роботи:

Підвищення ефективності моніторингу стану атмосферного повітря у місті шляхом розроблення інформаційної технології, що базується на даних громадського моніторингу та використовує мобільні технології для оперативного збору, обробки та візуалізації екологічних показників.

4.Вихідні дані для проведення робіт:

Набір даних моніторингу атмосферного повітря по посту 1315 наданими ресурсами Eco City .

5.Методи дослідження:

Дослідження існуючих інформаційних технологій, програмування та методи розробки програмного забезпечення, візуалізація даних, розробка UML-діаграм.

6. Етапи роботи і терміни їх виконання:

1. Загальна характеристика об’єкту дослідження ____ – ____
2. Вибір технологічного стеку та засобів розробки інформаційної технології моніторингу якості повітря з використанням мобільних пристроїв..... ____ – ____
3. Розробка інформаційної технології моніторингу якості атмосферного повітря .....____ – ____
4. Обробка та візуалізація даних громадського моніторингу для оцінювання стану атмосферного повітря.....____ – ____
5. Економічна частина.....____ – ____
6. Оформлення матеріалів до захисту МКР. .....____ – ____

7. Очікувані результати та порядок реалізації:

Забезпечити оперативний моніторинг стану атмосферного повітря шляхом розроблення мобільного застосунку, який відображає актуальні екологічні показники та автоматично визначає випадки перевищення допустимих концентрацій забруднювальних речовин.

Пояснювальна записка оформлена у відповідності до вимог «Методичних вказівок до виконання магістерських кваліфікаційних робіт для студентів

спеціальності 126 «Інформаційні системи та технології» (освітня програма «Інформаційні технології аналізу даних та зображень»)

9. Порядок приймання роботи

Публічний захист ..... «___» _____ 2025 р.

Початок розробки..... «___» _____ 2025 р.

Граничні терміни виконання МКР ..... «___» _____ 2025 р.

Розробив студент групи 2ІСТ-24м _____ Сергій ДЖУРА

Додаток Б

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: « Інформаційна технологія моніторингу стану атмосферного повітря з використанням мобільних пристроїв »

Тип роботи: магістерська кваліфікаційна робота

Підрозділ: кафедра САІТ, ФІІТА, гр. 2ІСТ-24м

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 0,98%

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне):

■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

□ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

□ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Віталій МОКІН, зав. каф. САІТ _____
(підпис)

Сергій ЖУКОВ, доц. каф. САІТ _____
(підпис)

Особа, відповідальна за перевірку _____ Сергій ЖУКОВ
(підпис)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Георгій ГОРЯЧЕВ, к.т.н., доц. каф. САІТ
(підпис)

Здобувач _____ Сергій ДЖУРА
(підпис)

Додаток В

Лістинг програмни

Код MainActivity

```
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
    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 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 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.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<*>) {
```

```
            }
        }
    }
    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()!!
                    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 MapActivity

```
package ua.edu.vntu
import CustomDataView
import CustomInfoWindowAdapter
import android.Manifest
import android.content.pm.PackageManager
import android.location.Location
import android.os.Bundle
import android.util.Log
import androidx.appcompat.app.AppCompatActivity
import androidx.core.app.ActivityCompat
import androidx.core.content.ContextCompat
import com.google.android.gms.location.FusedLocationProviderClient
import com.google.android.gms.location.LocationServices
import com.google.android.gms.maps.CameraUpdateFactory
import com.google.android.gms.maps.GoogleMap
import com.google.android.gms.maps.MapView
import com.google.android.gms.maps.OnMapReadyCallback
import com.google.android.gms.maps.SupportMapFragment
import com.google.android.gms.maps.model.LatLng
import com.google.android.gms.maps.model.LatLngBounds
import com.google.android.gms.maps.model.MarkerOptions
import okhttp3.ResponseBody
import retrofit2.Call
```

```
import retrofit2.Callback
import retrofit2.Response
import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory
import retrofit2.http.Body
import retrofit2.http.POST
import ua.edu.vntu.model.Feed
import ua.edu.vntu.model.FeedAPI
import ua.edu.vntu.model.JsonResponse
class MapActivity : AppCompatActivity(), OnMapReadyCallback {
    private lateinit var map: GoogleMap
    private lateinit var fusedLocationProviderClient: FusedLocationProviderClient
    private lateinit var currentLocation: Location
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.map_data)
        val mapFragment = supportFragmentManager.findFragmentById(R.id.map)
as SupportMapFragment
        mapFragment.getMapAsync(this)
        fusedLocationProviderClient =
LocationServices.getFusedLocationProviderClient(this)
        if (ContextCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION)
            != PackageManager.PERMISSION_GRANTED
        ) {
            ActivityCompat.requestPermissions(
                this,
                arrayOf(Manifest.permission.ACCESS_FINE_LOCATION),
                LOCATION_PERMISSION_REQUEST_CODE
            )
```

```
    } else {
        getCurrentLocation()
    }
}
override fun onMapReady(googleMap: GoogleMap) {
    map = googleMap
    map.uiSettings.isMyLocationButtonEnabled = true
}
private fun getCurrentLocation() {
    if (ActivityCompat.checkSelfPermission(
            this,
            Manifest.permission.ACCESS_FINE_LOCATION
        ) != PackageManager.PERMISSION_GRANTED
    ) {
        return
    }
    fusedLocationProviderClient.lastLocation
        .addOnSuccessListener { location: Location? ->
            fun drawMarkersOnMap(feeds: List<Feed>?) {
                map.clear() // Clear existing markers from the map
                feeds?.forEach { feed ->
                    val latitude = feed.latitude
                    val longitude = feed.longitude
                    val title = feed.field1
                    if (latitude != null && longitude != null) {
                     val markerPosition = LatLng(latitude, longitude)
                        val markerOptions = MarkerOptions()
                            .position(markerPosition)
                            .title(title)
                        val marker = map.addMarker(markerOptions)
```

```
                marker?.let {
                    it.showInfoWindow()
                }
                marker?.showInfoWindow()
            }
        }
        val bounds = LatLngBounds.Builder()
        feeds?.forEach { feed ->
            feed.latitude?.let { lat ->
                feed.longitude?.let { lng ->
                    bounds.include(LatLng(lat, lng))
                }
            }
        }
        val padding = 100
        map.setInfoWindowAdapter(CustomInfoWindowAdapter(this))
map.animateCamera(CameraUpdateFactory.newLatLngBounds(bounds.build(),
padding))
    }
    val retrofit = Retrofit.Builder()
        .baseUrl("https://api.thingspeak.com/")
        .addConverterFactory(GsonConverterFactory.create())
        .build()
    val productAPI = retrofit.create(FeedAPI::class.java)
    productAPI.getFeedsMap().enqueue(object : Callback<JsonResponse> {
        override fun onResponse(
            call: Call<JsonResponse>,
            response: Response<JsonResponse>
        ) {
            if (response.isSuccessful) {
```

```
                    val data = response.body()
                    drawMarkersOnMap(data?.feeds)
                }
            }
            override fun onFailure(call: Call<JsonResponse>, t: Throwable) {
                }
        })
    }
}
private fun sendLocationToServer(latitude: Double, longitude: Double) {
    val retrofit = Retrofit.Builder()
.baseUrl("https://api.thingspeak.com/channels/2418631/status.json?api_key=VCD
RU86G0QQWDHRR") // Replace with your actual base URL
        .addConverterFactory(GsonConverterFactory.create())
        .build()
    val locationApi = retrofit.create(LocationApi::class.java)
    val locationData = LocationData(latitude, longitude)
    val call = locationApi.sendLocation(locationData)
    call.enqueue(object : Callback<ResponseBody> {
        override fun onResponse(call: Call<ResponseBody>, response:
Response<ResponseBody>) {
            if (response.isSuccessful) {
                Log.d(TAG, "Дані про розташування успішно надіслані")
            } else {
                Log.e(TAG, "Не вдалося надіслати дані про розташування:
${response.code()}")
            }
        }
        override fun onFailure(call: Call<ResponseBody>, t: Throwable) {
```

```
                Log.e(TAG, "Помилка під час надсилання даних про розташування",
t)
            }
        })
    }
    override fun onRequestPermissionsResult(
        requestCode: Int,
        permissions: Array<out String>,
        grantResults: IntArray
    ) {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults)
        if (requestCode == LOCATION_PERMISSION_REQUEST_CODE &&
grantResults.isNotEmpty() && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
            getCurrentLocation()
        }
    }
    companion object {
        private const val LOCATION_PERMISSION_REQUEST_CODE = 1
        private const val TAG = "MapActivity"
    }
}
interface LocationApi {
    @POST("/location")
    fun sendLocation(@Body locationData: LocationData): Call<ResponseBody>
}
data class LocationData(
    val latitude: Double,
    val longitude: Double
)
```

Додаток Г

ІЛЮСТРАТИВНА ЧАСТИНА

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ МОНІТОРИНГУ СТАНУ АТМОСФЕРНОГО ПОВІТРЯ З ВИКРИСТАННЯМ МОБІЛЬНИХ ПРИСТРОЇВ

Нормоконтроль: к.т.н., доцент

_____ Сергій ЖУКОВ

«___» _____ 2025 р.

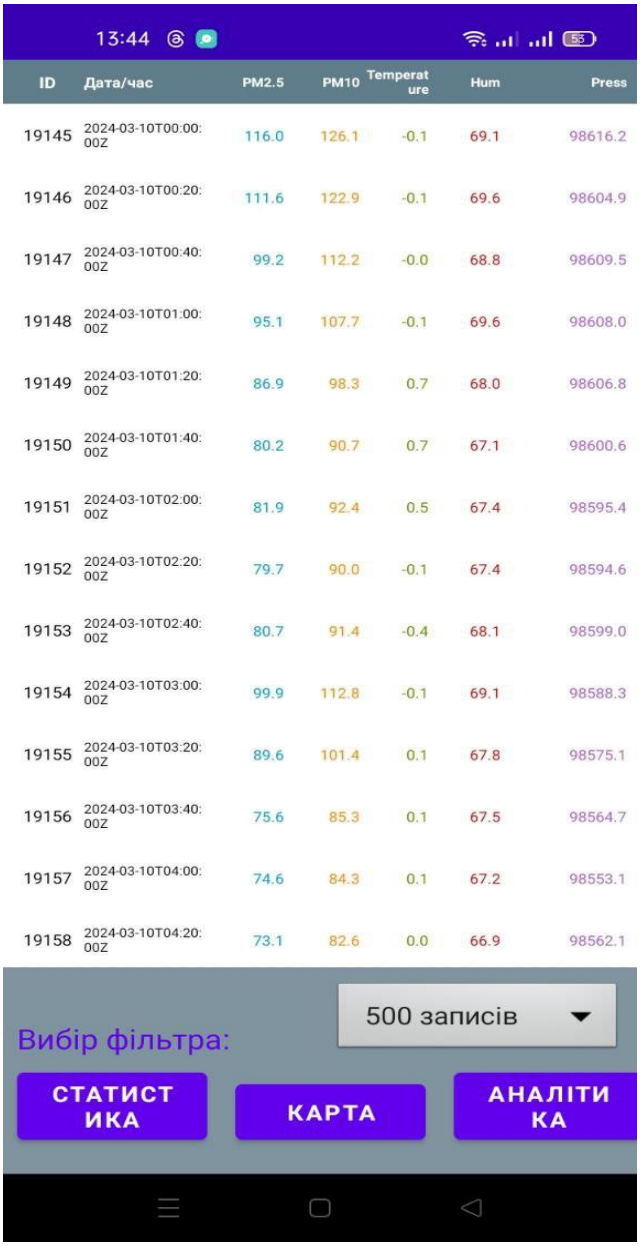

Рисунок Г.1 – Головний екран додатку з табличним представленням даних моніторингу

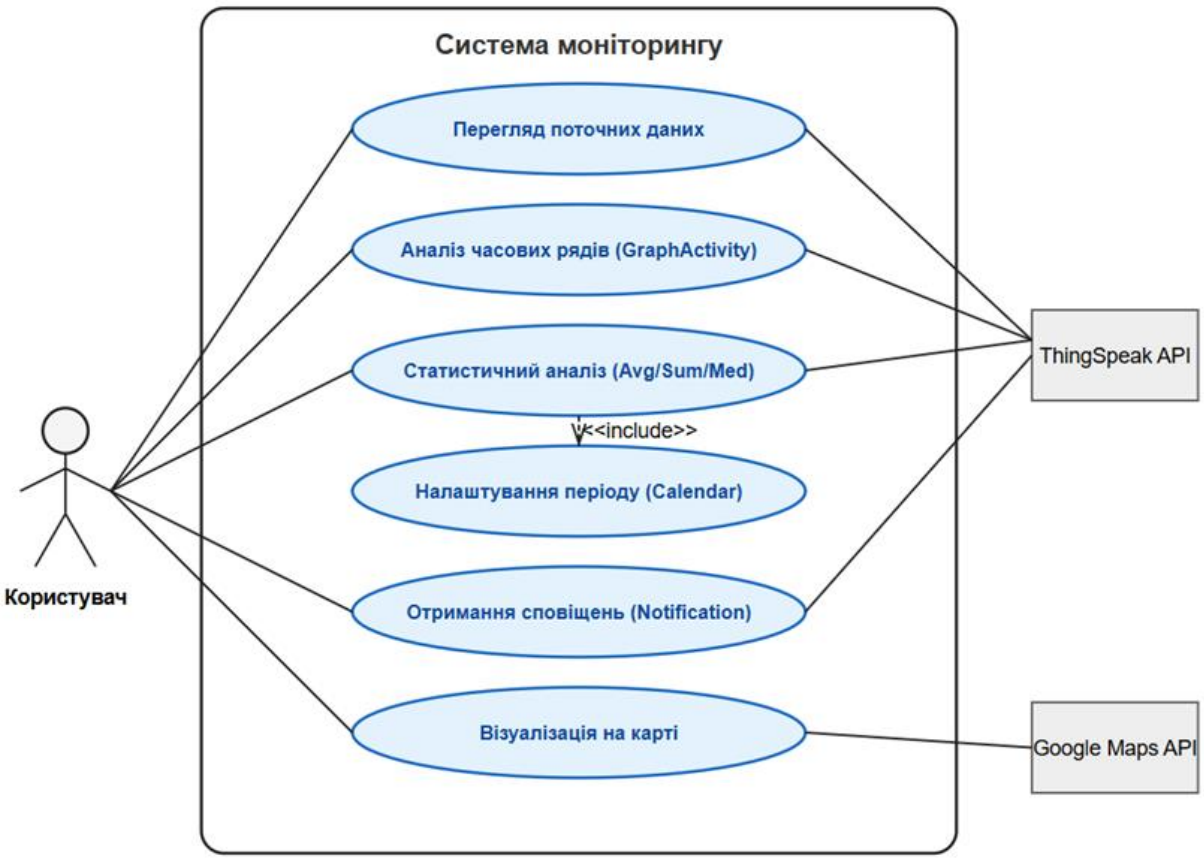

Рисунок Г.2 – UML-діаграма прецедентів системи моніторингу

Рисунок Г.3 – Візуалізація показників у ThingSpeak

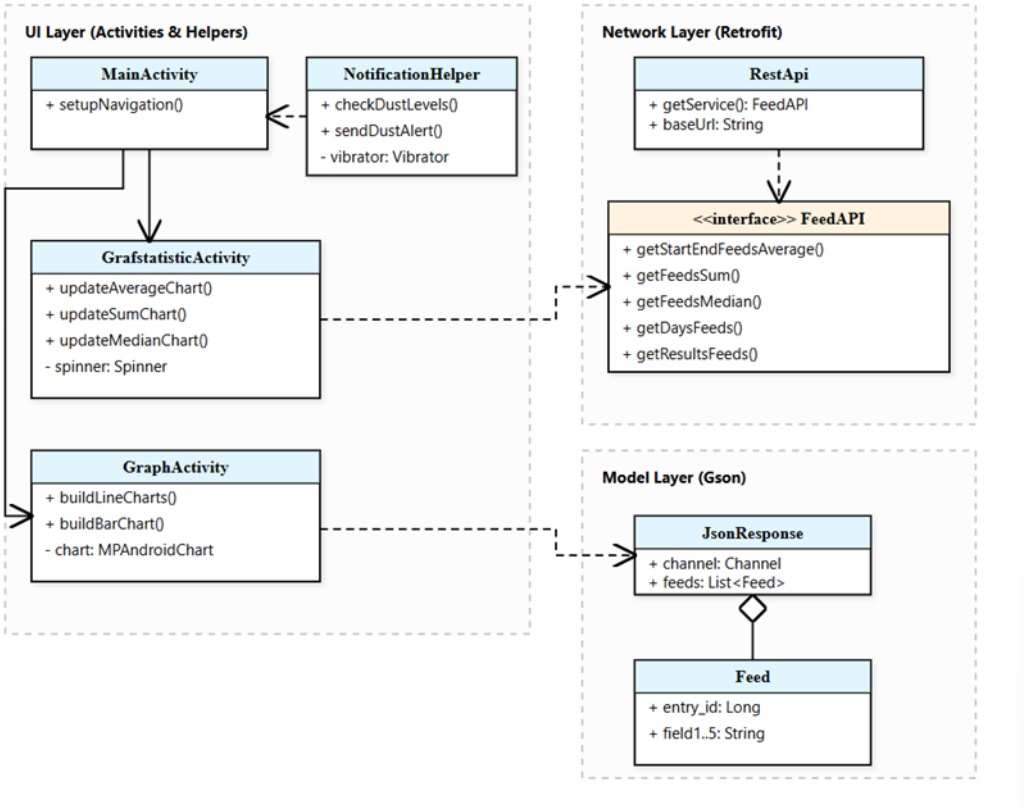

Рисунок Г.4 – UML-діаграма класів програмної системи моніторингу атмосферного повітря

Рисунок Г.5 – Інтерфейс вибору кількості записів для завантаження з сервера

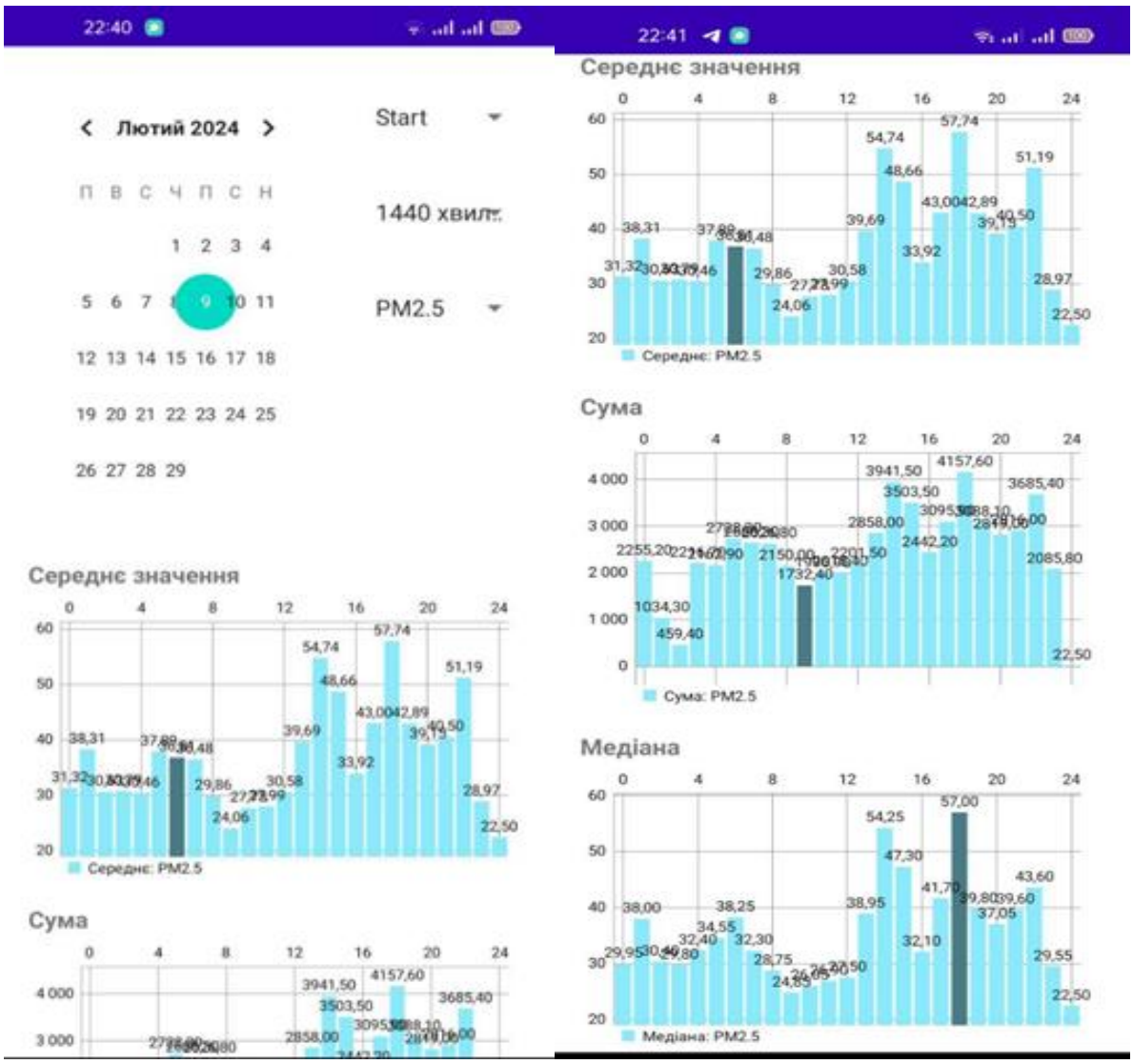

Рисунок Г.6 – – Візуалізація статистичних характеристик: середнє значення, сума та медіана

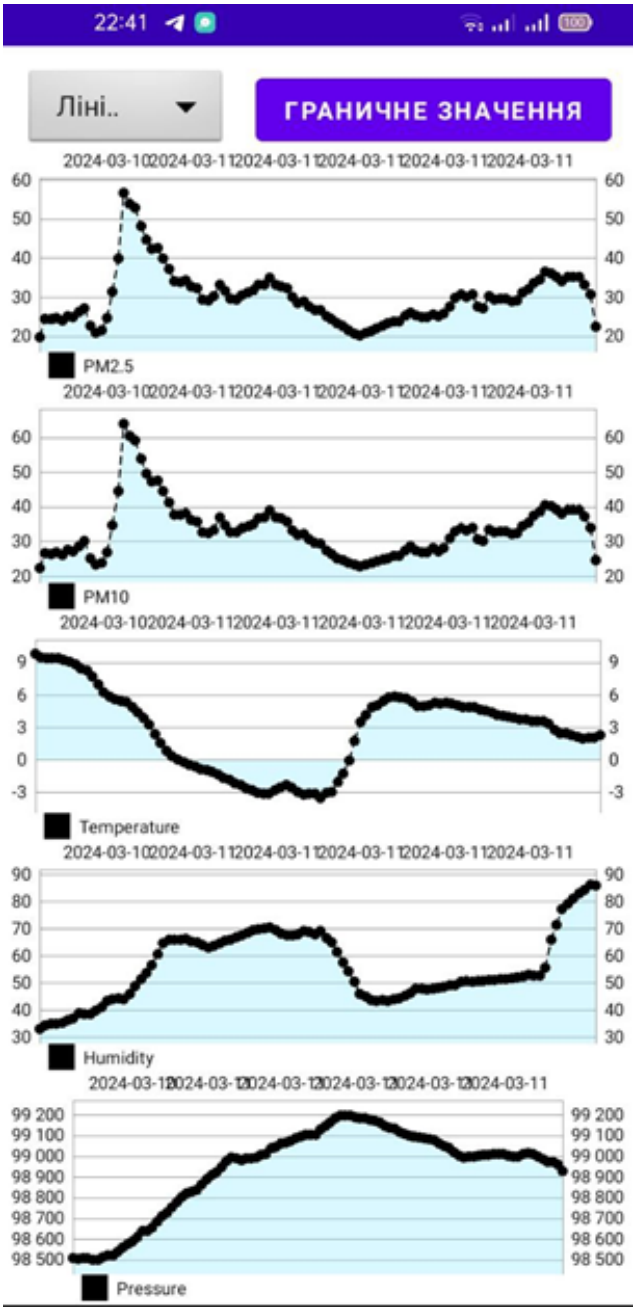

Рисунок Г.7 – Візуалізація п'яти параметрів моніторингу у вигляді лінійних графіків

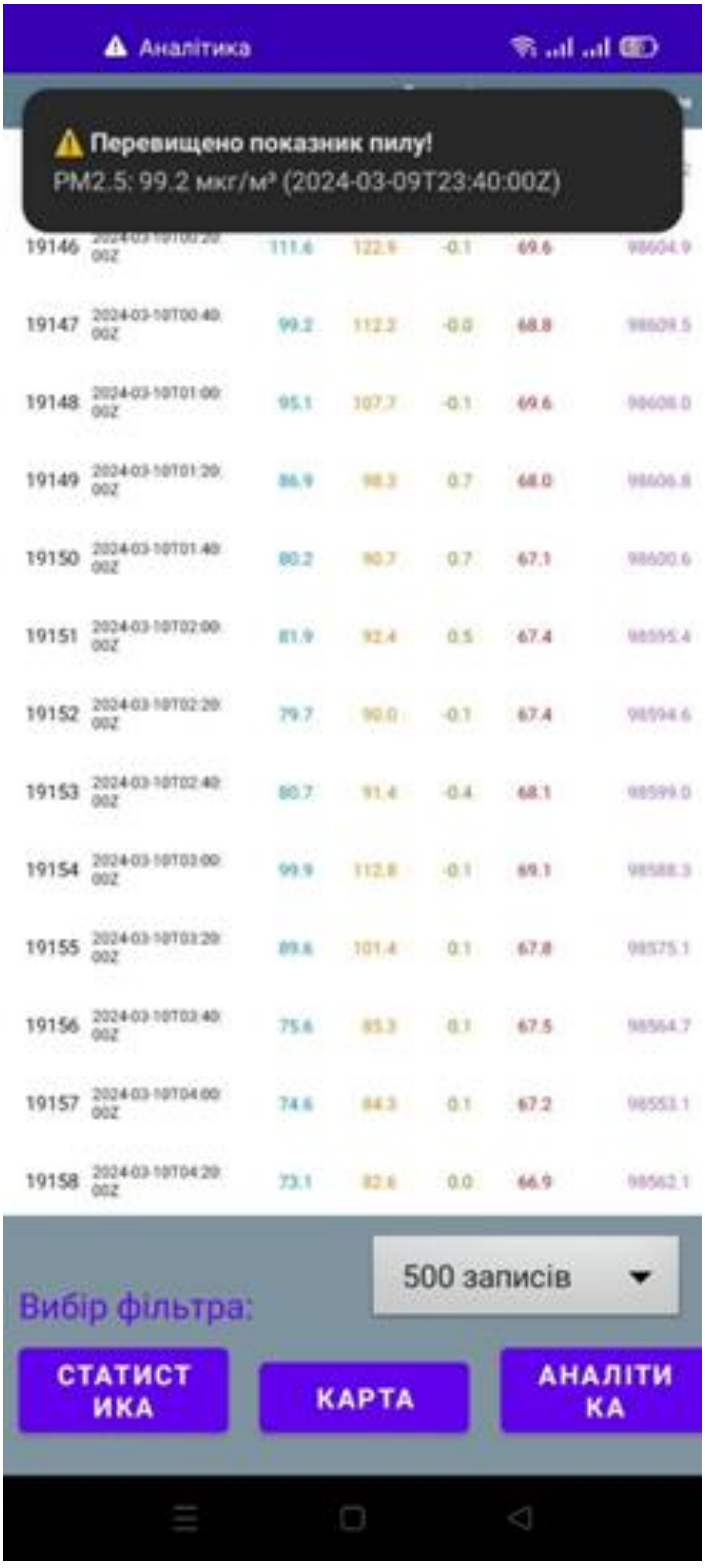

Рисунок Г.8 – Сповіщення про перевищення граничнодопустимої концентрації PM2.5

Додаток Д

Свідоцтво про реєстрацію авторського права на твір

УКРАЇНА

СВІДОЦТВО

про реєстрацію авторського права на твір

№ 140631

Комп'ютерна програма «Мобільний додаток для візуалізації та аналізу даних отриманих з пристроїв IoT» («IT Mobile Monitor»)

(вид, назва твору)

Автор (співавтори) **Горячев Георгій Володимирович, Джура Сергій Вікторович**

(прізвище, ім'я, по батькові (за наявності), псевдонім (за наявності))

Авторські майнові права належать повністю **Вінницький національний технічний університет, вул. Хмельницьке шосе, 95, м. Вінниця, Вінницька обл., 21021**

(прізвище, ім'я, по батькові (за наявності) фізичної особи / найменування юридичної особи, адреса)

Дата реєстрації 7 листопада 2025 р.

Директор Державної організації «Український національний офіс інтелектуальної власності та інновацій» **Олена ОРЛЮК**

УКРНОІВІ