

**МІНІСТЕРСТВО ФІНАНСІВ УКРАЇНИ
ДЕРЖАВНИЙ ПОДАТКОВИЙ УНІВЕРСИТЕТ**

**ФАКУЛЬТЕТ ФІНАНСІВ ТА ЦИФРОВИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І
СИСТЕМ**

КВАЛІФІКАЦІЙНА РОБОТА

на тему: ІТ ЗАСОБИ ЕКОМОНІТОРИНГУ

Здобувача вищої освіти другого (магістерського) рівня,
II курсу, групи КМІ-23-1, спеціальності 122 «Комп'ютерні науки»,
освітня програма «Інформаційні управляючі системи і технології в економіці»


(підпис)

Мадея Бориса Вікторовича
(прізвище, ім'я, по батькові)

Науковий керівник


(підпис)

Погорєловська Ірина
Дем'янівна, к.е.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь,
вчене звання)

Керівник освітньо-
професійної програми


(підпис)

Горбовий Артур Юліанович,
д.т.н, професор
(прізвище, ім'я, по батькові, науковий
ступінь, вчене звання)

Допущено до захисту за рішенням кафедри протокол від «10» січня 2025 №11

Завідувач кафедри


(підпис)

Філоненко Михайло
Миколайович, к.ф.-м.н.,
доцент
(прізвище, ім'я, по батькові,
науковий ступінь, вчене звання)

Ірпінь – 2025

Реєстрація № 13

24.01.2025
(дата)


(підпис відповідальної особи)

Результати перевірки

до захисту
(до захисту, на доопрацювання)

24.01.2025 
(дата) (підпис)

Результат захисту

4 (добре)
(оцінка)

05.02.2025 
(дата) (підпис)

Голова комісії:

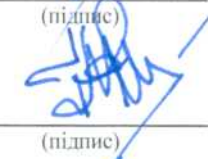

(підпис)

В.М. Франчук

Члени комісії:


(підпис)

А.Ю. Горбовий


(підпис)

М.М. Філоненко


(підпис)

І.Д. Погореловська

БЛАНК ОЦІНЮВАННЯ ДИПЛОМНОЇ РОБОТИ

Об'єкти контролю	Діапазон оцінки в балах	Фактична кількість балів	Примітки	Дата, П.І.Б.: керівника, голови комісії членів комісії, секретаря, (підписи)
ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ КОМІСІЄЮ ПО ЗАХИСТУ				
1. Зміст випускної роботи (якість теоретичного матеріалу, його зв'язок з практичним, вміння представляти матеріал в логічній послідовності, наявність висновків та пропозицій)	від 0 до 30	23		05.01.2025 Франчук В.М. <i>В.М.</i> Філоненко М.М. <i>М.М.</i> Погореловська І.Д. <i>І.Д.</i>
2. Дотримання вимог щодо оформлення	від 0 до 10	7		Горбовий А.Ю. <i>А.Ю.</i>
3. Вчасне виконання роботи на всіх етапах у встановлені кафедрою терміни	від 0 до 5	4		Суховерхівська Л.П. <i>Л.П.</i>
4. Рецензент	від 0 до 5	4		
РАЗОМ БАЛІВ	від 0 до 50	38		

Оцінювання кваліфікаційної роботи науковим керівником - 37 балів.

ПЕРЕВЕДЕННЯ ОЦІНКИ ЗА 100-БАЛЬНОЮ ШКАЛОЮ ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою	Дата	П.І.Б.: голови, членів комісії, секретаря, (підписи)
75	C	4 (добре)	05.01.2025	Франчук В.М. <i>В.М.</i> Горбовий А.Ю. <i>А.Ю.</i> Філоненко М.М. <i>М.М.</i> Погореловська І.Д. <i>І.Д.</i> Суховерхівська Л.П. <i>Л.П.</i>

ПРАВИЛА ПЕРЕВЕДЕННЯ ОЦІНОК ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою
90-100	A	5 (відмінно)
80-89	B	4 (добре)
70-79	C	
60-69	D	
50-59	E	3 (задовільно)
35-49	FX	2 (незадовільно)

Факультет фінансів та цифрових технологій
кафедра комп'ютерних та інформаційних технологій і систем
Галузь знань 12 «Інформаційні технології»
Спеціальність 122 «Комп'ютерні науки»
Освітня програма «Інформаційні управляючі системи і технології в економіці»

Затверджую

Завідувач кафедри  Філоненко М.М.

« 23 » грудня 2023

**Завдання на кваліфікаційну роботу
здобувачу вищої освіти**

Мадею Борису Вікторовичу

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи **ІТ- засоби екомоніторингу**

Затверджена наказом ректора від «20» грудня 2023 № 1701

2. Строк здачі закінченої роботи «07» січня 2025 року

3. Мета роботи: визначення ефективних інструментів для розробки мобільних додатків моніторингу якості повітря

Об'єкт дослідження: інформаційні технології та системи здійснення еко-моніторингу.

Предмет дослідження: інформаційні технології розробки засобів еко-моніторингу

База дослідження: моніторинг якості повітря у Бучанському районі

4. Вихідні дані до роботи: методи еко-моніторингу, технології та інструменти розробки та тестування мобільних додатків для еко-моніторингу

5. Перелік графічного матеріалу: копії екранів, лістинги коду, презентація роботи.

6. Зміст випускної кваліфікаційної (дипломної) роботи: РОЗДІЛ 1. СУЧАСНІ ПІДХОДИ ТА ТЕХНОЛОГІЇ ЕКО-МОНІТОРИНГУ ЯКОСТІ ПОВІТРЯ. 1.1. Огляд сучасних методів еко-моніторингу якості повітря 1.2. Огляд існуючих мобільних застосунків для еко-моніторингу якості повітря РОЗДІЛ 2 ПРОЄКТУВАННЯ ТА РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ. 2.1. Аналіз вибраної мови програмування Dart. 2.2. Програмна реалізація застосунку. РОЗДІЛ 3. ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ РОЗРОБЛЕНОГО РІШЕННЯ

7. Календарний план виконання роботи

№ п/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	
		За планом	За фактом
1	2	3	4
1	Вибір теми, підбір літературних джерел, формування структури кваліфікаційної роботи	До 15.12.2023	До 15.12.2023
2	I розділ (розкрити сутність і стан дослідження наукової проблематики роботи, після чого навести її загальну характеристику; огляд літератури за темою і вибір конкретних напрямів досліджень)	27.05.2024	27.05.2024
3	II розділ (аналіз експериментальної частини про методики досліджень)	02.09.2024	02.09.2024
4	III розділ (проведені теоретичні і (або) експериментальні дослідження та проектні рішення)	18.11.2024	18.11.2024
5	Висновки	10.12.2024	10.12.2024
6	Доопрацювання роботи з урахуванням зауважень наукового керівника	23.12.2024	23.12.2024
7	Отримання відгуку керівника	28.01.2025	28.01.2025
8	Отримання зовнішньої рецензії	31.01.2025	31.01.2025

8. Дата видачі завдання 23 грудня 2023

9. Керівник кваліфікаційної роботи

10. Керівник освітньо-професійної програми

11. Завдання до виконання отримав здобувач вищої освіти


(підпис)

Погорєловсь
(прізвище, ініціали)


(підпис)

Горбовий А.
(прізвище, ініціали)


(підпис)

Мадей Б. В.
(прізвище, ініціали)

АНОТАЦІЯ

ІТ-засоби екомоніторингу

Мета роботи: визначення ефективних інструментів для розробки мобільних додатків моніторингу якості повітря

Завдання роботи:

1. Аналіз сучасних методів еко-моніторингу якості повітря та існуючих мобільних застосунків.
2. Огляд технологій, інструментів і мов програмування для створення додатку.
3. Проектування та реалізація мобільного додатку.
4. Проведення тестування та оцінки ефективності розробленого рішення.

Предмет дослідження: інформаційні технології розробки засобів еко-моніторингу.

Об'єкт дослідження: інформаційні технології та системи здійснення еко-моніторингу.

Базою дослідження: екомоніторинг якості повітря у Бучанському районі

За результатами дослідження сформульовані висновки про те, що сучасні ІТ дозволяють розробити ефективний мобільний додаток для проведення еко-моніторингу.

Одержані результати можуть бути використані для підвищення екологічної обізнаності користувачів, надаючи інструмент для еко-моніторингу якості повітря та прийняттю превентивних заходів для захисту здоров'я.

Кваліфікаційна робота містить 52 сторінки, 1 таблиця, 3 рисунка, список використаних джерел 40 найменувань, 2 додатки.

Ключові слова: інформаційні технології, системи еко-моніторингу, мобільні додатки, MVC, API, БД, візуалізація

Рік виконання кваліфікаційної роботи 2024-2025

Рік захисту роботи 2025

ABSTRACT

IT tools for Eco-Monitoring

The purpose of this work is to explore effective tools for developing mobile applications for air quality monitoring.

Objectives of the Study:

1. Analysis of modern air quality eco-monitoring methods and existing mobile applications.
2. Review of technologies, tools, and programming languages for application development.
3. Design and implementation of a mobile application.
4. Testing and evaluation of the developed solution's effectiveness.

The subject of the study is information technologies for developing eco-monitoring tools.

The object of the study is information technologies and systems for eco-monitoring.

The research is based on air quality eco-monitoring in the Bucha district.

As a result of the study, conclusions were formulated stating that modern IT solutions enable the development of an effective mobile application for eco-monitoring.

The obtained results can be used to increase users' environmental awareness by providing a tool for air quality eco-monitoring and facilitating preventive measures to protect health.

The qualification paper consists of 52 pages, 1 table, 3 figures, a list of 40 references, and 2 appendices.

Keywords: information technologies, eco-monitoring systems, mobile applications, MVC, API, database (DB), visualization.

Year of completion: 2024-2025

Year of defense: 2025

ЗМІСТ

ВСТУП.....	3
РОЗДІЛ 1 СУЧАСНІ ПІДХОДИ ТА ТЕХНОЛОГІЇ ЕКО-МОНІТОРИНГУ ЯКОСТІ ПОВІТРЯ	5
1.1. Огляд сучасних методів еко-моніторингу якості повітря.....	5
1.2. Огляд існуючих мобільних застосунків для еко-моніторингу якості повітря	11
РОЗДІЛ 2 ПРОЄКТУВАННЯ ТА РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ	21
2.1. Аналіз вибраної мови програмування Dart.	21
2.2. Програмна реалізація застосунку	26
РОЗДІЛ 3. ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ РОЗРОБЛЕНОГО РІШЕННЯ.....	37
ВИСНОВКИ.....	41
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	43
ДОДАТКИ.....	48

ВСТУП

Інформаційні технології набули широкого поширення у вирішенні глобальних екологічних проблем, зокрема еко-моніторингу. Зростання рівня урбанізації, інтенсивний розвиток промисловості та збільшення обсягів транспортного руху призводять до підвищення рівня забруднення повітря та довкілля, що негативно впливає на здоров'я людей та стан навколишнього середовища. У цих умовах розробка ефективних інструментів для еко-моніторингу стає важливим завданням для запобігання екологічним кризам та покращення якості життя. Про актуальність та важливість дослідження проблем пов'язаних з використанням інформаційних технологій у здійсненні еко-моніторингу свідчать праці багатьох зарубіжних вчених, зокрема Е. Венгер, Г. Рейнгольд, П. Ші (США), Є.Д. Патаракін, Є.С. Полат, К. Свон, А.В. Хуторський. Серед вітчизняних учених проблемами використання інформаційних технологій у сфері еко-моніторингу займалися С.О. Довгий, В.П. Шевчук, О.М. Маринич, Г.О. Білявський, В.К. Хільчевський та інші. Їхні наукові праці сприяють розвитку методів збору, обробки та аналізу екологічних даних за допомогою інформаційних технологій.

Мобільні додатки для еко-моніторингу слугують для інформування населення про стан атмосферного середовища. Вони дозволяють отримувати дані в реальному часі, аналізувати динаміку змін і вчасно вживати заходів для захисту здоров'я. Проте аналіз існуючих рішень показує, що багато з них не повністю задовольняють потреби користувачів через обмежений функціонал або недостатню точність отриманих даних.

Актуальність розробки мобільного додатку для еко-моніторингу якості повітря полягає у створенні інструменту, який не лише забезпечує оперативний доступ до екологічної інформації, але й сприяє підвищенню обізнаності суспільства про проблеми забруднення повітря. Такий застосунок може стати важливим інструментом у боротьбі за покращення екологічної ситуації та підтримку здоров'я населення в умовах воєнного стану.

Мета кваліфікаційної роботи: визначення ефективних інструментів для розробки мобільних додатків моніторингу якості повітря.

Завданнями дослідження є:

- аналіз сучасних методів еко-моніторингу якості повітря та існуючих мобільних застосунків;
- огляд технологій, інструментів і мов програмування для створення додатку;
- проектування та реалізація мобільного додатку;
- проведення тестування та оцінки ефективності розробленого рішення.

База дослідження: еко-моніторинг якості повітря у Бучанському районі.

Об'єкт дослідження: інформаційні технології та системи здійснення еко-моніторингу.

Предмет дослідження: інформаційні технології розробки засобів еко-моніторингу.

Методи дослідження: У процесі виконання роботи застосовано низку наукових методів, що забезпечили комплексний підхід до аналізу предметної області. Системний аналіз використано при опрацюванні інформаційних джерел за темою дослідження, що дозволило встановити ключові закономірності та тенденції. Методи оцінювання застосовано під час аналізу та вибору мов програмування, враховуючи їхню ефективність, продуктивність і сферу застосування. Порівняльний аналіз здійснено з метою дослідження функціональних можливостей систем моніторингу, що дало змогу визначити їхні переваги та недоліки.

Практичне значення отриманих результатів: розроблений мобільний додаток сприяє підвищенню екологічної обізнаності користувачів, надає інструмент для еко-моніторингу якості повітря та прийняття превентивних заходів для захисту здоров'я.

Результати досліджень опубліковано у збірнику тез IV міжнародної науково-практичної Інтернет-конференції “Цифрова трансформація фінансової системи України та країн V-4 в умовах євроінтеграції” (Дубляни, 16 травня 2024 р.) [41].

РОЗДІЛ 1 СУЧАСНІ ПІДХОДИ ТА ТЕХНОЛОГІЇ ЕКО-МОНІТОРИНГУ ЯКОСТІ ПОВІТРЯ

1.1. Огляд сучасних методів еко-моніторингу якості повітря

У сучасний період стрімкого розвитку інформаційних технологій, поряд із досягненнями в галузі техніки, постають значні екологічні виклики, зокрема зростання рівня забруднення атмосферного повітря. Процеси виробництва, експлуатації та утилізації електронних пристроїв сприяють викидам шкідливих речовин у довкілля.

Сучасні методи еко-моніторингу є невід'ємною частиною екологічної науки, спрямованої на збереження довкілля та оцінку впливу людської діяльності. Супутникові дані дають змогу отримувати інформацію у реальному часі, що важливо для оцінки змін клімату, ерозії ґрунтів та забруднення середовища [16].

Ще одним ефективним методом є використання автоматизованих сенсорних систем для збору даних на локальному рівні. Ці системи включають датчики для моніторингу якості повітря, води, рівня шуму та радіації. Їх основна перевага — постійний збір точних даних, що дає змогу своєчасно реагувати на критичні зміни в екосистемах. Поєднання сенсорів із технологіями Інтернету речей (IoT) дозволяє створювати розподілені системи моніторингу, які охоплюють великі території [19].

Біологічний моніторинг також займає важливе місце серед сучасних методів. Він базується на використанні індикаторних видів рослин, тварин і мікроорганізмів для оцінки стану екосистем. Наприклад, лишайники часто використовують для визначення рівня забруднення повітря, а міграція риб може вказувати на зміни у стані водних об'єктів. Завдяки цьому методу можна виявляти навіть незначні зміни, які залишаються непомітними для інших способів спостереження [16].

У сучасних умовах великого значення набуває інтеграція даних із різних джерел і їхній аналіз за допомогою штучного інтелекту (ШІ) та машинного

навчання. Ці технології дозволяють ефективно обробляти великі обсяги даних, виявляти закономірності та прогнозувати можливі сценарії розвитку подій. Впровадження ШІ в екологічний моніторинг значно підвищує точність та швидкість аналізу, що є критично важливим у боротьбі з екологічними кризами [12].

Забруднення повітря дедалі частіше розглядається як глобальна криза через численні негативні наслідки для здоров'я людини та стану навколишнього середовища. Відповідно до дослідження Гарвардського університету, забруднення повітря, спричинене спалюванням викопного палива, є причиною приблизно кожної п'ятої смерті у світі [32].

Особливо небезпечним вважається забруднення твердими частинками, яке асоціюється з процесами спалювання викопного палива. За оцінками, воно є причиною близько 8 мільйонів передчасних смертей щорічно. Водночас як гострий, так і кумулятивний вплив забруднення атмосферного повітря може спричинити до 10 мільйонів смертей на глобальному рівні [34].

Еко-моніторинг якості повітря передбачає збір та аналіз проб атмосферного повітря для визначення рівня його забруднення. Отримані дані порівнюються зі встановленими стандартами чистоти повітря, історичними показниками, а також із параметрами, що демонструють вплив забруднення на здоров'я людини та стан навколишнього середовища. Це дозволяє оцінити поточний стан повітря та визначити необхідність відповідних заходів.

Також стрімке зростання міського населення та мільйони передчасних смертей, спричинених забрудненням повітря щорічно, підкреслюють критичну потребу в більш глибокому розумінні якості міського повітря. Особливості урбанізованих територій, такі як розташування джерел забруднення та специфічний рельєф, сприяють неоднорідному розподілу концентрації забруднюючих речовин. [12]. Ці фактори часто залишаються поза увагою традиційних методів еко-моніторингу

Хоча класичні системи еко-моніторингу забезпечують високу точність даних, їх великогабаритність, складність обслуговування та висока вартість

обмежують можливості широкого використання. Це призводить до низької просторової роздільної здатності, що ускладнює отримання детальної картини забруднення. У цьому контексті використання мережі недорогих вимірювальних станцій відкриває перспективи для значного підвищення просторової деталізації даних та зміни підходів до еко-моніторингу якості повітря [10].

Також отримання надійної та високоякісної інформації від систем еко-моніторингу такого типу супроводжується низкою труднощів. Вже на етапі проєктування виникають виклики, пов'язані зі створенням надійних великомасштабних мереж, здатних забезпечувати безперервний еко-моніторинг у міських умовах. Додаткові ускладнення пов'язані з обмеженнями сучасних сенсорних технологій. Недорогі хімічні сенсори, які часто використовуються в таких системах, мають низку технічних недоліків. Зокрема, це нестабільність роботи, низьке співвідношення сигнал/шуму, перехресна чутливість (до параметрів довкілля або інших хімічних речовин), а також повільна динамічна реакція [11].

Незважаючи на ці труднощі, розвиток технологій сенсорів та алгоритмів обробки даних поступово вдосконалює можливості таких систем. Інноваційні методи калібрування, використання машинного навчання для підвищення точності вимірювань, а також розробка нових матеріалів для датчиків сприяють підвищенню їх ефективності та надійності. Водночас інтеграція недорогих сенсорних станцій у широкомасштабні мережі дозволяє компенсувати їх обмеження завдяки збільшенню обсягу зібраних даних і поліпшенню просторової роздільної здатності еко-моніторингу [14].

Традиційний підхід до еко-моніторингу якості повітря базується на використанні регіональних мереж стаціонарних станцій вимірювання, наприклад, станцій моніторингу Держгідромету, муніципальних постів екологічного контролю або приватних екологічних лабораторій. Стаціонарний еко-моніторинг здійснює збір даних про рівень забруднення повітря з фіксованих точок спостереження, які зазвичай розташовані поблизу

потенційних джерел забруднення, таких як промислові підприємства або дороги з інтенсивним транспортним потоком [3].

Такі станції, як-от міські екологічні пости, станції контролю промислового забруднення та мобільні лабораторії, часто встановлюються в районах, де населення зазнає непропорційно високого впливу забрудненого повітря, з метою оцінки його впливу на здоров'я та навколишнє середовище. Вони оснащені спеціалізованим вимірювальним обладнанням, яке дозволяє досліджувати рівень забруднення повітря та його вплив на здоров'я людини і стан навколишнього середовища. Експлуатацією цього обладнання займаються державні установи, наукові організації або приватні структури. Стаціонарні станції зазвичай великі за розмірами та обладнані дорогими високоточними приладами, такими як маспектрометри, газові хроматографи, лазерні аналізатори, абсорбційні спектрофотометри, а також датчики для вимірювання концентрації твердих частинок, озону, оксидів азоту тощо тощо [16]. Окрім вимірювання концентрації забруднювачів, ці станції також контролюють відповідні метеорологічні параметри, такі як температура, опади, швидкість і напрямок вітру, що дозволяє отримати комплексну картину стану атмосферного повітря.

Ефективний еко-моніторинг якості повітря вимагає комплексного підходу, що поєднує різні методи збору даних для отримання повної картини забруднення. Висока вартість стаціонарних станцій зумовлює формування надзвичайно розріджених мереж для еко-моніторингу. Одним із альтернативних методів збору даних про забруднення повітря є персональний еко-моніторинг, який широко використовується в епідеміологічних дослідженнях для вивчення впливу забруднення на людину. Цей метод передбачає використання портативного обладнання, яке носить досліджувані особи протягом визначеного періоду часу (наприклад, одна доба або 24 години). Таке обладнання часто розміщується в рюкзаку або на тілі досліджуваного, після чого повертається для передачі зібраних даних і їх подальшої обробки [1].

Перевага персонального еко-моніторингу полягає в можливості врахування індивідуальних умов впливу забруднення, таких як мобільність людини та її перебування в різних середовищах (приміщення, транспорт, відкритий простір). Це дозволяє отримати детальніші дані про вплив забруднення на організм у реальному житті, що неможливо досягти лише за допомогою стаціонарних станцій [38].

Для персонального еко-моніторингу часто застосовуються дифузійні пасивні пробовідбірники, які використовуються для вимірювання концентрації діоксиду азоту (NO_2) та озону (O_3). Ці пристрої є відносно дешевими, та функціонують за принципом накопичення забруднюючих речовин протягом визначеного часу. Однак, вони не здатні фіксувати короточасні коливання концентрації забруднювачів. Для отримання точних даних зібрані проби потребують лабораторної обробки та аналізу [1].

Для інших типів забруднювачів, таких як тверді частинки та чадний газ (CO) доступні портативні реєстратори, які забезпечують реєстрацію даних у реальному часі. Проте їх висока вартість суттєво обмежує кількість пристроїв, доступних для досліджень. Хоча такі реєстратори можуть зменшити накладні витрати, пов'язані з лабораторною обробкою, їхня ціна залишається істотним бар'єром для масштабного використання в дослідженнях.

Персональний еко-моніторинг відіграє ключову роль у дослідженні впливу забруднюючих речовин на індивідуальному рівні, забезпечуючи детальну інформацію про взаємозв'язок між концентраціями забруднювачів у зовнішньому середовищі та в приміщеннях. Завдяки високій практичній цінності таких систем, вони були обрані як прототип для реалізації в рамках даної роботи [11].

У цьому контексті важливим є розвиток бездротових сенсорних мереж (WSN), які відкривають нові можливості для еко-моніторингу якості повітря. Ці мережі використовують відносно недорогі сенсорні пристрої, здатні здійснювати вимірювання, обробку даних та передачу інформації. Завдяки масовому розгортанню сенсорів на певній території можна отримувати

високоточні дані про концентрацію забруднювачів із високою просторовою роздільною здатністю, що є критично важливим для оцінки якості повітря на місцевому рівні [16].

Окрім екологічних аспектів, забруднення повітря має значний вплив на економіку, оскільки погана якість повітря безпосередньо впливає на здоров'я населення. За даними Всесвітньої організації охорони здоров'я (ВООЗ) та Організації економічного співробітництва та розвитку (ОЕСР), у 2022 році витрати, пов'язані з передчасною смертністю та інвалідністю через забруднене повітря в Європі, становили приблизно 1,6 трлн доларів США. Це підкреслює важливість інноваційних підходів до еко-моніторингу, які можуть сприяти зниженню негативних наслідків як на рівні здоров'я, так і в економічній сфері [32].

За даними дослідження Стенфордського університету, економічні збитки від забруднення повітря в США становлять близько 6% ВВП, що у 2023 році склало 790 мільярдів доларів. Найбільші втрати зафіксовані у секторах сільського господарства, комунальних послуг, виробництва та транспорту, де забруднення повітря суттєво впливає на ефективність діяльності та довкілля [24].

В Україні проблема забруднення повітря також має значний вплив на економіку та суспільство. За даними Державної служби статистики України, у 2021 році викиди забруднюючих речовин в атмосферу від стаціонарних джерел склали понад 2 млн т. [4]. Основними джерелами забруднення є промисловість, енергетика та транспорт, що призводить до погіршення якості повітря, зниження продуктивності праці та збільшення витрат на охорону здоров'я. Експерти також зазначають, що через погану якість повітря втрачаються значні обсяги врожаю сільськогосподарських культур [1].

Різні негативні економічні наслідки є результатом поганої якості повітря, зокрема впливу дрібнодисперсних частинок PM_{2.5}. До ключових проблем належать:

- Зниження врожайності сільськогосподарських культур

- Втрата навчальних і робочих днів.
- Зниження продуктивності праці.

Еко-моніторинг якості повітря є критично важливим для оцінки шкоди, яку завдає забруднення нашому здоров'ю та навколишньому середовищу. В Україні розробка та впровадження сучасних систем еко-моніторингу дозволить отримувати високоточні дані про стан атмосфери, що є важливим кроком до зменшення впливу забруднення. Створення добре розроблених мереж еко-моніторингу сприятиме більш ефективному управлінню якістю повітря, що допоможе зменшити економічні втрати та покращити стан довкілля [39].

1.2. Огляд існуючих мобільних застосунків для еко-моніторингу якості повітря

Перед тим як аналізувати міжнародні мобільні застосунки для еко-моніторингу якості повітря, доцільно зосередитися на вивченні ринку України. Українські платформи для еко-моніторингу мають особливе значення, оскільки враховують локальні особливості та джерела забруднення, які не завжди відображаються у глобальних системах. Зокрема, в Україні функціонують інтерактивні мапи та ресурси, орієнтовані на надання інформації про стан повітря в режимі реального часу. Ці ресурси є важливими інструментами як для громадян, так і для наукових досліджень, особливо в умовах воєнного стану.

Одним із найвідоміших українських проєктів є EcoCity - інтерактивна мапа, яка відображає якість повітря у великих містах України та загальний стан по областях. Ця платформа є одним із небагатьох ресурсів, розроблених локально, що забезпечує високу актуальність даних для українського контексту [7].

EcoCity є українським проєктом, орієнтованим на інформування населення про стан атмосферного повітря. Застосунок дозволяє користувачам отримувати інформацію про рівень забруднення повітря у великих містах України та в регіонах [18].

Основними функціональними особливостями є:

- Інтерактивна мапа: Мапа відображає якість повітря з використанням кольорових індикаторів. Наприклад, синій колір сигналізує про чисте повітря, а фіолетовий попереджає про надзвичайно високий рівень забруднення.

- Розділ "Статистика": Користувачі можуть аналізувати історичні дані, які допомагають оцінити динаміку змін якості повітря у певних регіонах.

- Джерела даних: Система збирає інформацію від локальних сенсорів, розміщених у різних містах України, що забезпечує точні та актуальні дані.

Перевагою EcoCity є те, що проєкт адаптований до українських реалій і враховує особливості місцевих джерел забруднення, таких як промислові підприємства, транспортна інфраструктура та локальні природні умови [7].

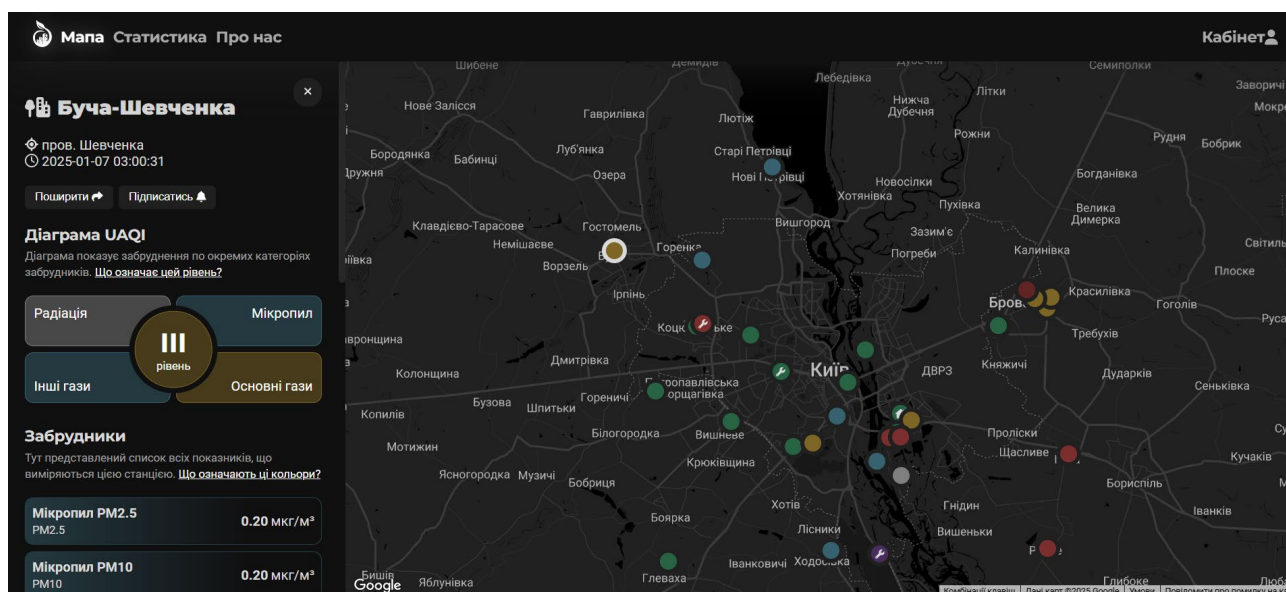


Рис. 1.1. Система EcoCity [7]

Також одним із популярних застосунків в Україні можна виділити SaveEcoBot, який є багатофункціональною екологічною системою, та надає комплексну інформацію про стан довкілля в Україні. Ця платформа дозволяє користувачам отримувати не лише дані про якість повітря, а й інформацію про інші важливі екологічні показники, такі як радіаційний фон, наявність пожеж і напрямок вітру. Завдяки інтеграції декількох джерел даних, SaveEcoBot

забезпечує широкий спектр інформації для підвищення екологічної обізнаності та швидкого реагування на зміни довкілля [8].

До основних функцій можна віднести:

- Інтерактивну мапу якості повітря, де користувачі можуть візуалізувати рівні забруднення в реальному часі та отримувати актуальну інформацію для регіонів України.
- Рівень радіації: Застосунок надає інформацію про радіаційний фон, який особливо важливий для територій з підвищеним екологічним ризиком.
- Еко-моніторинг пожеж: Інтеграція даних про пожежі сприяє інформуванню про екстрені ситуації та потенційний вплив на стан повітря.
- Дані про підприємства-забруднювачі: SaveEcoBot містить інформацію про підприємства, які здійснюють шкідливі викиди, та масштаби цих викидів та дозволяє користувачам дізнаватися більше про джерела локального забруднення.
- Напрямок вітру, де показник напрямку та швидкості вітру додає контекст до аналізу якості повітря, дозволяючи оцінити поширення забруднюючих речовин [9].

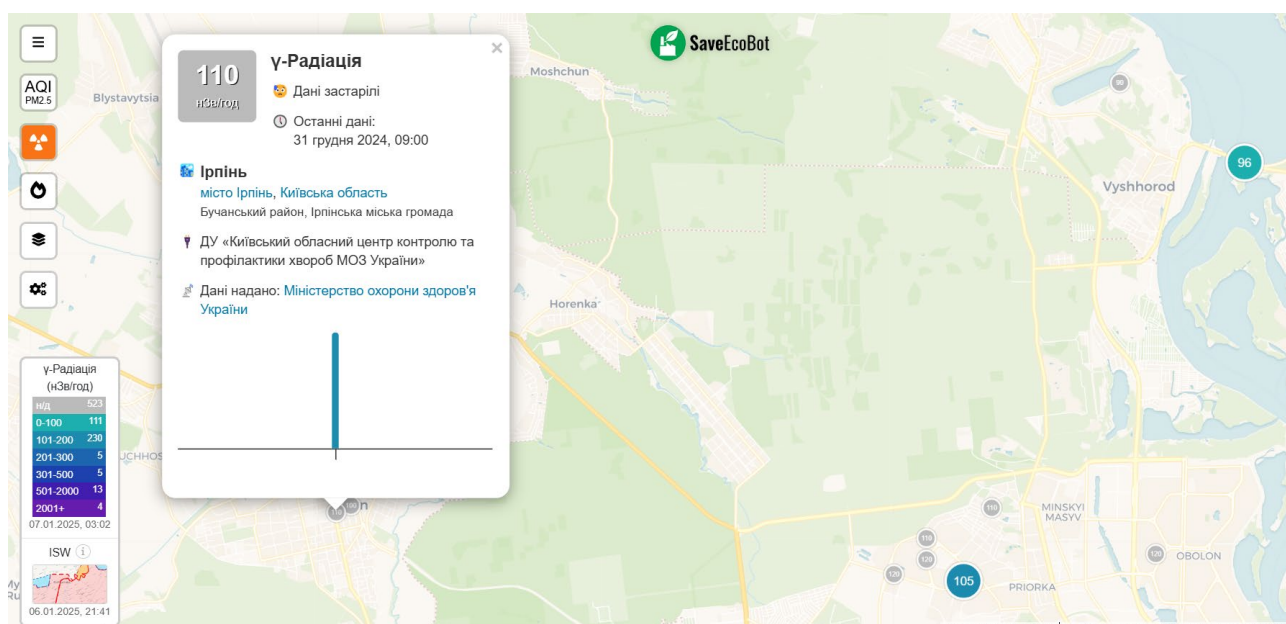


Рис. 1.2. Система SaveEcoBot [8]

“ЕкоЗагроза” — це ще один офіційний вебресурс та мобільний додаток, який розроблений Міністерством захисту довкілля України для еко-моніторингу стану навколишнього середовища та оперативного реагування на екологічні загрози. Застосунок надає користувачам доступ до інтерактивної мапи, яка відображає актуальну інформацію про якість атмосферного повітря, радіаційний фон, стан водних ресурсів, а також інші важливі екологічні показники для будь-якого населеного пункту України [5].

Основною перевагою “ЕкоЗагрози” є інтеграція різних видів екологічних даних у єдину платформу, що дозволяє не лише отримувати актуальну інформацію, але й аналізувати її в контексті загальної екологічної ситуації. Застосунок забезпечує простий і швидкий доступ до інформації як для пересічних громадян, так і для органів місцевого самоврядування. Можливість оперативного виклику екстрених служб у випадку виявлення екологічних загроз або стихійних лих є важливим елементом системи реагування на надзвичайні ситуації.

Особливістю “ЕкоЗагрози” є функція подання повідомлень про екологічні злочини. Користувачі можуть повідомляти про такі інциденти, як пожежі в лісах, розливи нафтопродуктів, незаконні викиди шкідливих речовин, забруднення водних ресурсів тощо. Це сприяє залученню громадськості до еко-моніторингу стану довкілля та дозволяє швидше реагувати на порушення, зменшуючи їхній негативний вплив на екосистему.

Додатково застосунок надає можливість сповіщення про повітряну тривогу, який надзвичайно важливий в умовах сучасних викликів, з якими стикається Україна. Інтеграція такої функції дозволяє поєднати екологічний еко-моніторинг із забезпеченням безпеки населення.

“ЕкоЗагроза” є прикладом того, як сучасні цифрові технології можуть бути використані для вирішення нагальних екологічних проблем. Завдяки комплексності та зручності у використанні цей застосунок сприяє підвищенню екологічної обізнаності населення та формуванню культури відповідального ставлення до довкілля. Його функціонал дозволяє не лише моніторити стан

природних ресурсів, але й активно впливати на процеси захисту довкілля через інструменти громадського контролю [5].

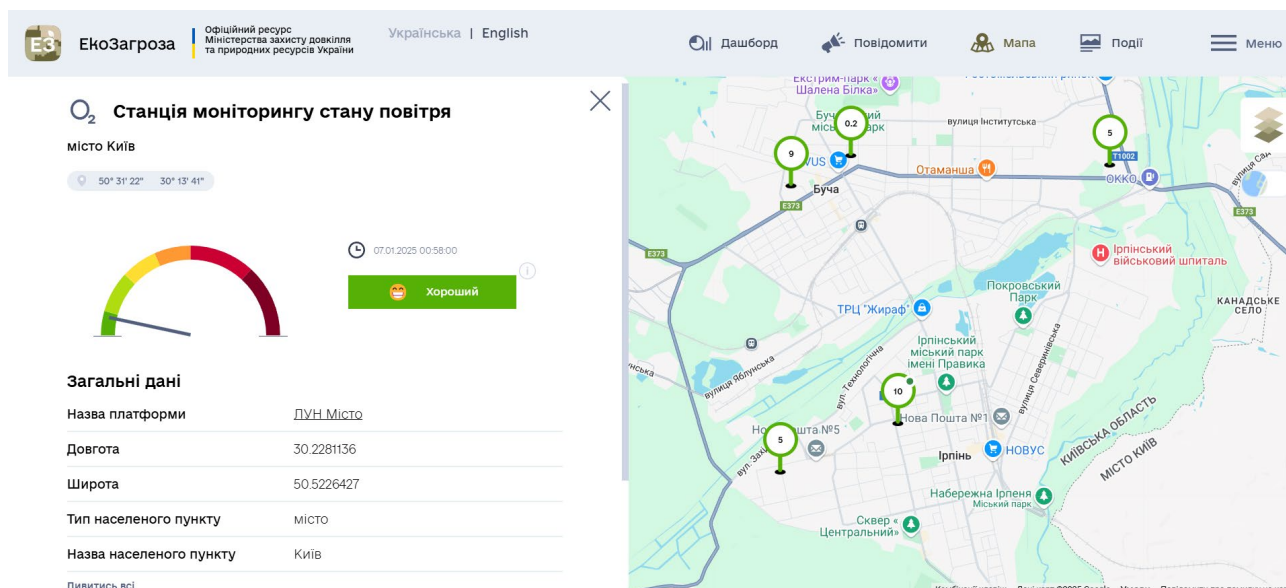


Рис.1.3. Система ЕкоЗагроза [5]

З останніх найпопулярніших застосунків в Україні можна виділити YourAirTest

YourAirTest є сучасною системою еко-моніторингу якості атмосферного повітря, яка надає користувачам точну, перевірену та актуальну інформацію про склад і стан повітря відповідно до міжнародних стандартів індексу якості повітря (AQI). Цей застосунок особливо популярний в Україні серед жителів таких міст, як Дніпро, Київ, Харків, Одеса та інші. YourAirTest дозволяє відстежувати рівень забруднення повітря у будь-якому місці, надаючи рекомендації щодо дій у випадку небезпечних концентрацій шкідливих речовин [13].

Застосунок забезпечує користувачів детальними даними про шість основних забруднювачів повітря, включаючи тверді частинки PM2.5 і PM10, озон (O_3), діоксид азоту (NO_2), діоксид сірки (SO_2) та оксид вуглецю (CO). Користувачі мають доступ до інтерактивної карти якості повітря в режимі реального часу, можуть отримувати інформацію про вплив кожного із

забруднювачів на здоров'я та використовувати рекомендації для зменшення негативного впливу на себе та свою сім'ю.

Особливістю YourAirTest є забезпечення оновлення даних щогодини, що гарантує високу оперативність. Найточніші дані доступні для локацій, поблизу яких встановлені наземні датчики якості повітря. Команда проєкту використовує відкриті дані понад 30 000 таких пристроїв у всьому світі. У випадках, коли потрібна локація знаходиться далеко від датчиків, YourAirTest застосовує математичне моделювання, що дозволяє обчислювати якість повітря з високою точністю навіть у віддалених місцевостях [19].

Для підвищення точності розрахунків система також інтегрує супутникові дані, які дозволяють враховувати глобальний рух повітряних мас. Використання методів машинного навчання допомагає системі інтерпретувати інформацію про атмосферу та прогнозувати можливі зміни якості повітря.

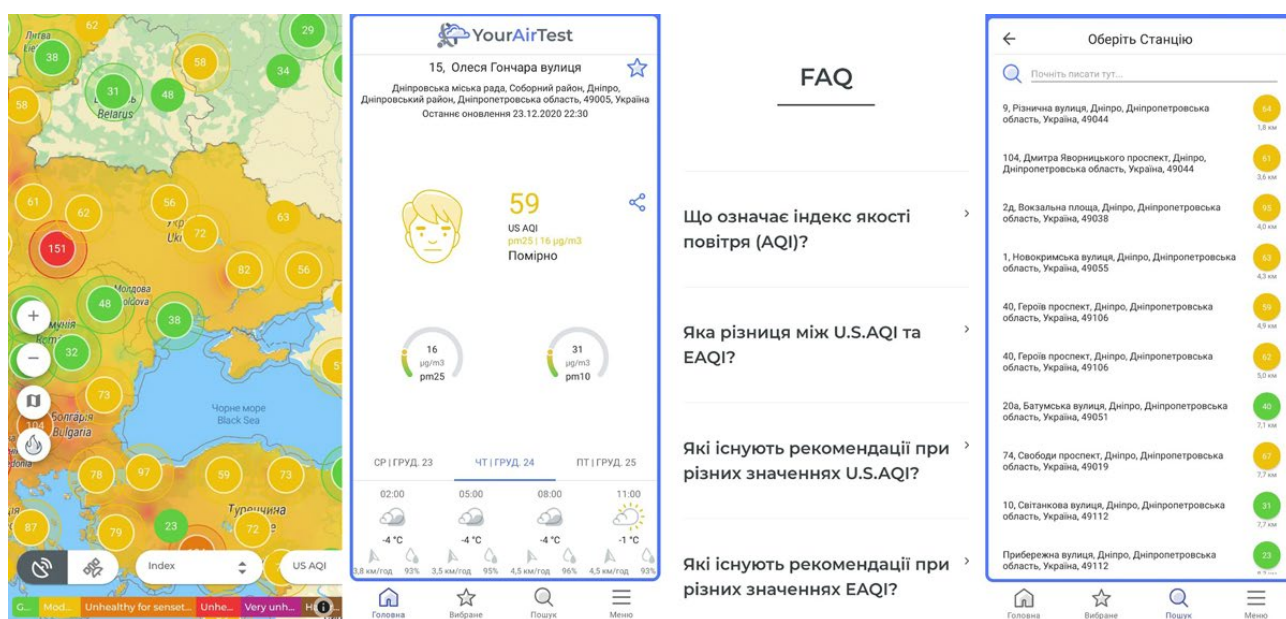


Рис. 1.4. Застосунок YourAirTest [13]

YourAirTest це приклад інтеграції високотехнологічних підходів до екомоніторингу стану повітря та надання рекомендацій для підвищення екологічної обізнаності населення. Завдяки широкій географії покриття, доступу до глобальних і локальних даних та використанню передових

технологій, цей застосунок стає важливим інструментом для зменшення впливу забруднення на здоров'я людей та стан довкілля [13].

Окрім локальних мобільних застосунків, розроблених в Україні, існує безліч міжнародних програм для еко-моніторингу якості повітря, які широко використовуються у різних країнах. Вони забезпечують доступ до актуальних даних про стан атмосфери, використовуючи глобальні сенсорні мережі, супутникові дані та сучасні алгоритми обробки. Одним із найвідоміших і найбільш популярних додатків є IQAir AirVisual.

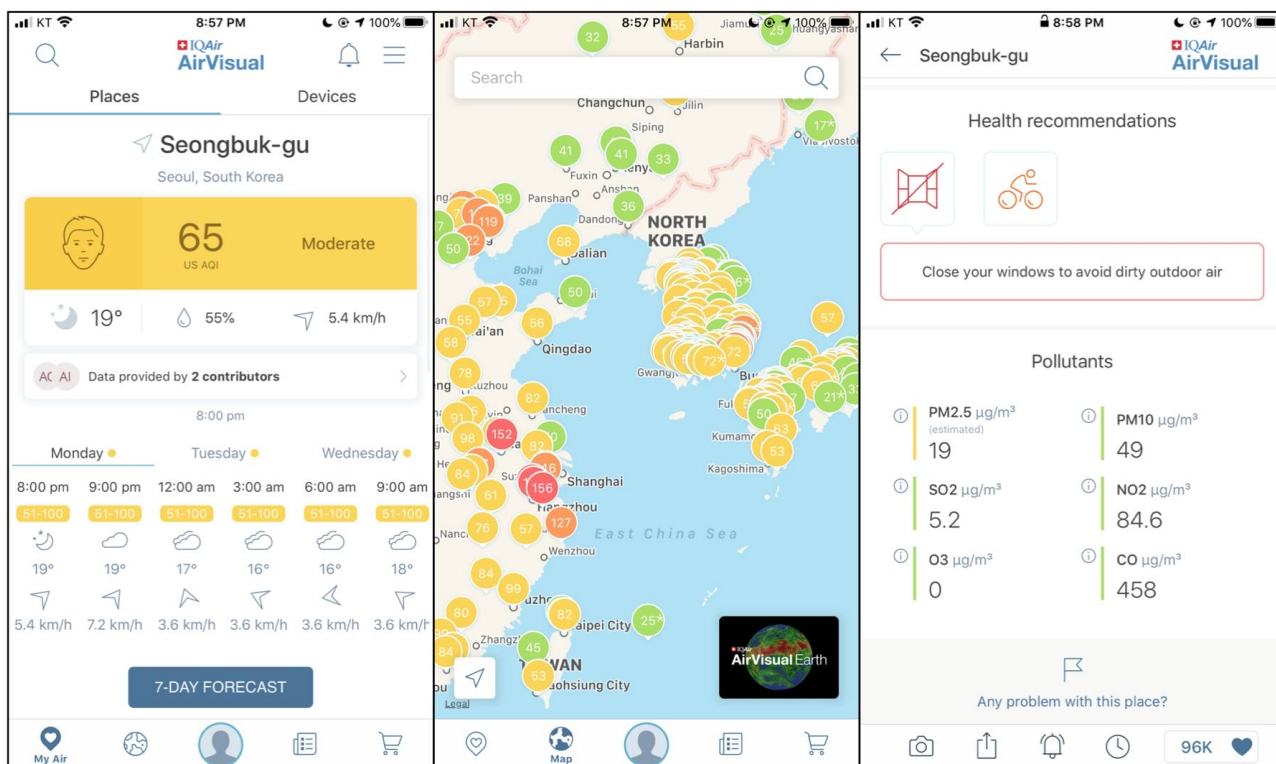


Рис. 1.5. Застосунок QAir AirVisual

QAir AirVisual — це високотехнологічний додаток, що надає повний спектр інформації про якість повітря в усьому світі. Застосунок пропонує дані про основні забруднювачі, такі як дрібнодисперсні частинки (PM2.5 і PM10), діоксид сірки (SO₂), діоксид азоту (NO₂), озон (O₃) та оксид вуглецю (CO). Користувачі можуть отримувати ці дані у зручній формі завдяки інтерактивній візуалізації [10].

Окрім мобільного додатку, IQAir також пропонує веб-сайт, що доповнює функціонал програми. Вебресурс містить інтерактивну карту світу в реальному

часі, оновлювані статті про забруднення довкілля, а також доступ до онлайн-магазину компанії, де представлені монітори якості повітря. Такий мультиплатформенний підхід забезпечує легкий доступ до інформації незалежно від типу пристрою [15].

Значною перевагою AirVisual є можливість зберігати обрані місця для швидкого доступу до інформації. Ця функція особливо зручна для тих, хто часто подорожує, оскільки дозволяє зберігати дані для різних міст. Наприклад, користувач може відстежувати стан повітря у таких великих мегаполісах, як Сеул, Нью Йорк, Лондон чи Париж [22].

Наступним популярним застосунком є AirMatters, який пропонує широкий функціонал для користувачів. Хоча він поступається за популярністю таким програмам, як AirVisual, AirMatters відрізняється сучасним дизайном і багатством функцій, що роблять його привабливим для широкої аудиторії. Інтерфейс програми виконаний у темному режимі, що забезпечує комфортне використання, зокрема, в умовах слабкого освітлення, а контрастні кольори сприяють зручності сприйняття інформації [10].



Рис. 1.6. Застосунок AirMatters

Щодо індексу якості повітря (AQI) та забруднюючих речовин, AirMatters надає ті самі ключові показники, що й AirVisual. Програма підтримує шкалу AQI, затверджену Агентством з охорони навколишнього середовища США (US EPA), і пропонує дані про концентрації основних забруднювачів, включаючи PM2.5, PM10, NO₂, O₃, CO та SO₂. Ці дані представлені у числовому вигляді та підкріплені кольоровими індикаторами, що дозволяє швидко ідентифікувати рівень забруднення.

Однією з ключових переваг AirMatters є здатність спрощувати складні екологічні показники та надавати їх у вигляді практичних рекомендацій. Наприклад, у додатку є розділ, присвячений порадам щодо здоров'я, де користувачі можуть дізнатися, чи варто використовувати маску або очищувач повітря залежно від поточного стану повітря.

Програма також інтегрується з системою Siri на iOS, що дозволяє користувачам швидко отримувати дані про якість повітря за допомогою голосових запитів. Це забезпечує зручний доступ до інформації, яка може бути недоступною у стандартних додатках, таких як програми погоди. Ще однією особливістю, що привертає увагу, є можливість використання віджетів. Віджети дозволяють виводити основну інформацію про AQI безпосередньо на головний екран смартфона, що спрощує доступ до даних без необхідності відкривати додаток.

Окрім еко-моніторингу забруднення повітря, AirMatters надає додаткову функціональність, таку як контроль факторів, пов'язаних з алергією. У певних регіонах додаток може відображати рівні пилку берези, трави, полину, амброзії та інших потенційно алергенних речовин, що робить його корисним для людей, чутливих до цих факторів [21].

Мобільні застосунки для моніторингу якості повітря є важливим інструментом підвищення обізнаності населення про стан довкілля та вплив забруднення на здоров'я. Їх розробка стає особливо актуальною в умовах зростання урбанізації, збільшення промислових викидів і транспортного забруднення, що суттєво погіршують якість повітря у містах та сприяють

зростанню ризиків розвитку респіраторних і серцево-судинних захворювань. Оперативний доступ до актуальної інформації про стан атмосферного середовища є критично важливим для забезпечення безпеки громадян і формування екологічної свідомості.

Застосунки для еко-моніторингу дозволяють користувачам у режимі реального часу отримувати точні дані про концентрацію ключових забруднювачів, таких як дрібнодисперсні частинки (PM_{2.5}, PM₁₀), оксиди азоту та озон. Інтеграція з локальними мережами сенсорів і аналітичними платформами забезпечує високу деталізацію та актуальність інформації, що значно підвищує ефективність таких інструментів.

Окрім інформаційної функції, такі додатки мають профілактичний потенціал, надаючи рекомендації для користувачів у разі підвищеного рівня забруднення. Це включає поради щодо уникнення перебування на відкритому повітрі, використання засобів захисту або планування активностей у більш сприятливі години. Подальший розвиток таких програм, зокрема їхня локалізація та інтеграція з місцевими сенсорними мережами, може значно покращити доступність та якість отримуваних даних, сприяючи підвищенню екологічної відповідальності населення.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ

2.1. Аналіз вибраної мови програмування Dart.

Провівши детальний аналіз існуючих мобільних додатків для еко-моніторингу якості повітря, а також ретельно вивчивши можливості різних мов програмування, було обрано мову Dart для розробки проєкту. Це рішення базується на кількох ключових аргументах, які демонструють переваги Dart у контексті створення сучасного мобільного додатку.

Dart — це мова програмування з відкритим вихідним кодом, розроблена компанією Google. Вона є мовою загального призначення, яка використовується для створення веб-додатків, програмування на стороні сервера та розробки мобільних застосунків. Dart також є основною мовою програмування для фреймворку Flutter, що забезпечує розробку кросплатформних додатків. Мова відома своєю простотою у вивченні, високою продуктивністю та підтримкою принципів об'єктно-орієнтованого програмування [28].

Основні характеристики Dart

- Простота у вивченні: Dart спроектовано таким чином, щоб розробники могли швидко освоїти мову, особливо ті, хто має досвід роботи з іншими мовами програмування, такими як JavaScript, Java чи C#.
- Висока продуктивність: Dart оптимізований для створення швидких і чуйних програм, забезпечуючи ефективну роботу додатків на різних платформах.
- Клієнтська мова програмування: Dart дозволяє створювати інтерактивні інтерфейси користувача для веб-додатків, що робить її ідеальною для програмування на стороні клієнта.
- Швидке виконання: Завдяки компіляторам "Just-In-Time" (JIT) для швидкої розробки та "Ahead-Of-Time" (AOT) для виробничих середовищ, Dart

забезпечує як швидкий запуск додатків, так і високу продуктивність у процесі їх виконання.

- **Об'єктно-орієнтований дизайн:** Dart підтримує ключові принципи об'єктно-орієнтованого програмування, такі як класи, успадкування, міксини та інтерфейси. Це дозволяє розробникам організовувати код у багаторазові, модульні компоненти, що сприяє покращенню читабельності, зручності обслуговування та масштабованості програм.

- **Підтримка інструментів:** Dart забезпечений потужним набором інструментів для розробки, таких як менеджер пакетів pub, статичний аналізатор коду dartanalyzer, налагоджувач і інші утиліти. Ці інструменти інтегруються з популярними середовищами розробки, такими як Visual Studio Code та IntelliJ IDEA, що створює комфортні умови для роботи розробників.

- **Міжплатформна сумісність:** Dart підтримує кросплатформну розробку, дозволяючи створювати код, який працює на різних платформах, таких як веб-браузери, мобільні пристрої (з використанням Flutter) і сервери. Це значно спрощує повторне використання коду та підвищує продуктивність розробників.

Dart забезпечує високий рівень продуктивності завдяки компіляції в машинний код за допомогою АОТ-компіляції (Ahead-Of-Time). Це дозволяє створювати додатки, які працюють швидко та стабільно. Крім того, підтримка як статичної, так і динамічної типізації робить мову гнучкою у використанні, дозволяючи створювати надійний і легко масштабований код. Простий та інтуїтивний синтаксис, схожий на JavaScript та C#, полегшує освоєння мови, що сприяє пришвидшенню процесу розробки [25].

Мова Dart, завдяки своїй інтеграції з кросплатформним фреймворком Flutter, забезпечує ідеальні умови для створення багатоплатформних додатків. Flutter значно розширює можливості Dart, поєднуючи продуктивність мови програмування з потужною архітектурою для розробки інтерфейсів користувача.

Flutter — це SDK інтерфейсу користувача з відкритим вихідним кодом, розроблений Google, який належить до чотирьох найпопулярніших фреймворків для створення додатків. Його головна перевага полягає у здатності розробляти кросплатформні мобільні додатки для Android та iOS за допомогою єдиної кодової бази. Ця особливість значно знижує витрати часу та ресурсів, необхідних для окремої розробки для кожної платформи.

Одним із головних компонентів Flutter є його набір готових віджетів, які дозволяють створювати складні, багатофункціональні інтерфейси користувача без необхідності використання сторонніх бібліотек. Віджети є гнучкими, налаштовуваними та відповідають принципам матеріального дизайну (Material Design) і стилю iOS. Це дозволяє забезпечувати єдиний вигляд і поведінку додатків на різних платформах.

Функція "гаряче перезавантаження" (Hot Reload) є однією з найцінніших особливостей Flutter. Вона дозволяє розробникам миттєво бачити зміни в інтерфейсі програми після внесення правок у код, що суттєво прискорює розробку та тестування.

Простота використання Flutter зробила його популярним вибором серед розробників, які прагнуть створювати кросплатформні додатки з мінімальними витратами на адаптацію до різних операційних систем. Це підтверджують великі компанії, які обрали Flutter для своїх продуктів, серед яких Google Ads, Groupon, та The New York Times. Ці програми демонструють потенціал Flutter у створенні складних і масштабованих застосунків [33].

Основні характеристики Flutter

- Швидка розробка: Завдяки функції "гарячого перезавантаження" (Hot Reload) розробники можуть миттєво спостерігати за внесеними змінами без необхідності перезапуску програми. Це суттєво зменшує час розробки та тестування, дозволяючи зосередитися на оптимізації функціоналу та дизайну.
- Експресивний інтерфейс користувача: Flutter надає багатий набір настроюваних віджетів, які дозволяють створювати візуально привабливі та високоінтерактивні інтерфейси. Завдяки цьому розробники можуть легко

реалізовувати складні дизайни, анімації та інтерактивні елементи, які відповідають сучасним стандартам користувацького досвіду.

- Єдина кодова база: Flutter дозволяє розробникам писати код лише один раз і використовувати його для створення додатків для різних платформ, включаючи iOS, Android, веб та настільні системи. Це значно скорочує час і ресурси, необхідні для розробки, забезпечуючи при цьому узгоджений досвід для користувачів на різних пристроях.

- Висока продуктивність: Завдяки використанню мови програмування Dart та сучасної реактивної архітектури Flutter забезпечує плавну роботу додатків зі швидкістю 60 кадрів на секунду навіть на пристроях із обмеженими апаратними ресурсами. Це гарантує стабільну та швидку роботу програм у різних умовах.

- Нативна продуктивність: Flutter компілює код безпосередньо у машинний код, що дозволяє досягти майже нативної продуктивності додатків. Це забезпечує користувачам швидкий відгук додатків, зберігаючи при цьому ефективність процесу розробки.

- Інтеграція з платформою: Flutter забезпечує глибоку інтеграцію з функціями та API конкретних платформ. Це дозволяє розробникам використовувати датчики пристроїв, сервіси та інші апаратні можливості, забезпечуючи додаткам нативний досвід, адаптований для кожної платформи [35].

Flutter, завдяки своїй універсальності та потужності, потребує інтегрованого середовища розробки (IDE), яке може забезпечити ефективну підтримку на кожному етапі створення додатків. У цьому контексті Android Studio стала основним вибором для розробників Flutter, пропонуючи комплексний набір інструментів для оптимізації процесу розробки [33].

Android Studio забезпечує повну інтеграцію з Flutter і мовою програмування Dart, що робить його ідеальним середовищем для створення багатоплатформних додатків. Завдяки інтуїтивно зрозумілому інтерфейсу, IDE пропонує інструменти, які полегшують розробку, тестування та оптимізацію

додатків. Від функцій автозаповнення коду до засобів візуалізації інтерфейсу, Android Studio допомагає розробникам зосередитися на функціональності та дизайні програм [30].

Однією з ключових переваг Android Studio є її потужний емулятор, який дозволяє тестувати програми Flutter на різних віртуальних пристроях. Це включає перевірку роботи додатків на різних версіях Android, роздільній здатності екрана та апаратних конфігураціях. Така функціональність дає змогу розробникам виявляти можливі проблеми з сумісністю ще на ранніх етапах розробки.

Крім того, Android Studio надає інструменти для дизайну інтерфейсу користувача. Редактор макетів і функції попереднього перегляду дають змогу створювати візуально привабливі інтерфейси, що відповідають стандартам сучасного UI/UX дизайну. Це особливо корисно для дизайнерів, які можуть тестувати свої концепції в реальному часі.

Android Studio також включає функції для управління життєвим циклом розробки. Інструменти відстеження помилок, оптимізації продуктивності та еко-моніторингу використання ресурсів допомагають створювати високоякісні, стабільні програми з мінімальними зусиллями [30].

Після ознайомлення з Dart як мовою програмування та Flutter як фреймворком для створення багатоплатформних додатків, виникає потреба розглянути інструменти, які спрощують розробку. Однією з найважливіших частин будь-якого сучасного додатка є робота з мережею, зокрема інтеграція з REST API. Для цього в екосистемі Flutter доступні численні бібліотеки, серед яких особливе місце займають Retrofit та Chopper.

Retrofit, що бере свій початок із екосистеми Android, адаптований для використання у Flutter. Ця бібліотека надає високий рівень абстракції для виконання мережових запитів, перетворюючи HTTP API у прості для використання Dart-інтерфейси. Основа роботи Retrofit базується на використанні клієнта Dio, який забезпечує стабільну роботу з HTTP-запитами,

та генератора коду `build_runner`, що автоматизує створення необхідних компонентів для роботи з API [36].

Головною перевагою Retrofit є його здатність мінімізувати обсяг стандартного коду, пов'язаного з налаштуванням мережових викликів. Розробник визначає HTTP-запити за допомогою анотацій, таких як `@GET` або `@POST`, які зіставляються з методами Dart. Це дозволяє створювати типобезпечні клієнти, що суттєво спрощує роботу з API [37].

Особливості Retrofit

- Автоматизація серіалізації JSON: Retrofit забезпечує автоматичне перетворення даних JSON у класи моделі Dart, що зменшує кількість ручної роботи та знижує ймовірність помилок.
- Гнучка конфігурація: Глобальні параметри, такі як базовий URL, заголовки, перехоплювачі та автентифікація, можуть бути легко налаштовані через конфігурацію Retrofit.
- Згенерований код: Використання `build_runner` забезпечує генерацію типобезпечного коду для клієнта API, позбавляючи розробника необхідності працювати на низькому рівні з HTTP-запитами.
- Підтримка перехоплювачів: Retrofit дозволяє налаштовувати перехоплювачі для реєстрації запитів, обробки автентифікації або керування скасуванням запитів.

Retrofit, у поєднанні з Dart та Flutter, це ефективний інструмент для спрощення роботи з REST API, забезпечуючи розробників потужними можливостями для оптимізації процесу інтеграції мережових сервісів. Завдяки автоматизації, гнучкості та зручності налаштування, ця бібліотека значно прискорює розробку та робить її більш надійною. У взаємодії з іншими інструментами Dart-екосистеми Retrofit допомагає створювати стабільні, масштабовані та функціональні додатки [23].

2.2. Програмна реалізація застосунку

Програмна реалізація застосунку для еко-моніторингу якості повітря передбачає створення багатофункціонального мобільного додатку, що поєднує

сучасні технології розробки. Основою є використання мови програмування Dart і фреймворку Flutter, які забезпечують потужний інструментарій для кросплатформної розробки. Застосунок має забезпечувати інтерактивний інтерфейс користувача, обробку даних у реальному часі та інтеграцію з REST API для отримання актуальної інформації про якість повітря [27].

Архітектурні патерни відіграють ключову роль у розробці застосунків на Flutter. У контексті еко-моніторингу якості повітря було обрано архітектуру на основі керування станом із використанням патернів Bloc (Business Logic Component) або Provider. Ці підходи дозволяють ізолювати бізнес-логіку від інтерфейсу користувача, забезпечуючи чітке розмежування відповідальностей [33].

Для забезпечення якісної реалізації застосунку важливо дотримуватися основних принципів гарної архітектури, які є фундаментом для створення масштабованого, тестованого та ремонтпридатного програмного забезпечення. Застосування цих принципів забезпечує стійкість проєкту на етапі розробки та в подальшій експлуатації.

- Розділення проблем: Кожна частина програми має відповідати за певний аспект роботи, зменшуючи взаємозалежність між компонентами. Це дозволяє розробникам фокусуватися на окремих задачах, полегшуючи процес підтримки.
- Масштабованість: Архітектура повинна забезпечувати можливість додавання нових функцій без значного ускладнення структури програми.
- Можливість тестування: Компоненти додатку мають бути простими для модульного та інтеграційного тестування, що сприяє створенню надійного програмного забезпечення.
- Ремонтпридатність: Код має бути зрозумілим, легко підтримуваним і модифікованим, що знижує витрати на розвиток і обслуговування програми.

Відповідно до цих принципів, одним із класичних архітектурних шаблонів, що широко застосовується, є Model-View-Controller (MVC). MVC забезпечує чітке розділення обов'язків між компонентами програми:

- **Модель (Model):** Відповідає за дані та бізнес-логіку. У контексті Flutter це може бути логіка обробки API або збереження даних у локальній базі.
- **Представлення (View):** Визначає, як дані відображаються користувачеві. У Flutter це віджети, які формують інтерфейс користувача.
- **Контролер (Controller):** Діє як посередник, отримуючи введені користувачем дані, обробляючи їх через модель та оновлюючи подання.

Застосування MVC у Flutter сприяє реалізації принципу розділення проблем, дозволяючи розробникам працювати над кожним компонентом незалежно. Наприклад, модифікація бізнес-логіки або зміна представлення може здійснюватися без необхідності внесення змін у контролер [33].

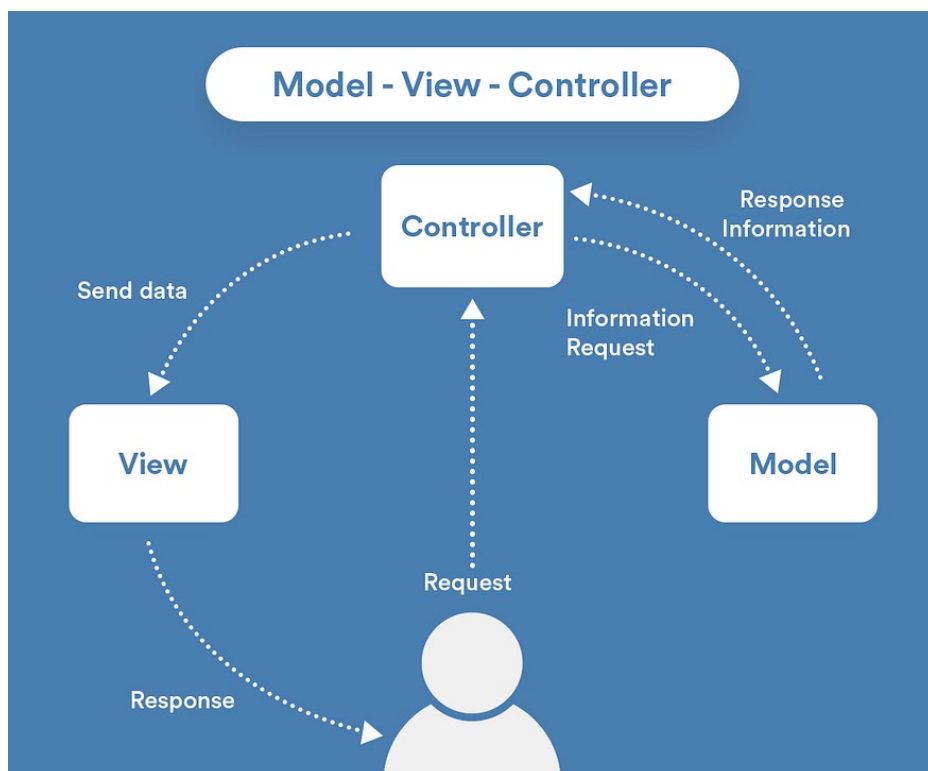


Рис. 2.1. Архітектура MVC [27]

У рамках розробки мобільного застосунку, заснованого на принципах гарної архітектури, одним із найважливіших елементів є API (інтерфейс прикладного програмування). API забезпечує інтеграцію між різними компонентами системи, а також між додатком і зовнішніми платформами, полегшуючи обмін даними та функціональність.

API відіграє подібну роль до протоколів у мережевій взаємодії, але стосується обміну даними між додатками. Це набір стандартизованих інструкцій і процедур, які дозволяють розробникам отримувати доступ до функцій і даних інших платформ. Завдяки цьому стандартизованому підходу до взаємодії, процес написання коду для інтеграції з іншими додатками стає значно простішим та ефективнішим.

На серверній стороні API є набором процедур, реалізованих за допомогою програмного коду. Ці процедури приймають запити від клієнтських додатків і надсилають відповідні відповіді. Наприклад, API може отримувати HTTP-запити на отримання даних, таких як якість повітря у вибраному регіоні, і надсилати відповідь у форматі JSON, який легко обробляється клієнтським додатком [27].

Розробники API створюють ці інтерфейси за допомогою стандартизованих операторів програмування, забезпечуючи доступ до певних функцій та ресурсів платформи. У контексті Flutter, використання API є основою для обробки REST-запитів та отримання актуальної інформації з серверів, що робить його ключовим елементом для додатків, орієнтованих на динамічні дані.

Важливим внеском у розвиток концепції API стала теорія передачі репрезентативного стану (REST), запропонована комп'ютерним вченим Роєм Філдінгом. У своїй докторській дисертації Філдінг описав REST як абстракцію архітектурних елементів для розподілених гіпермедійних систем, таких як Всесвітня павутина.

REST забезпечує чітко визначені ролі для ключових компонентів:

- Агент користувача: Наприклад, веб-браузер, який ініціює запити.

- Початковий сервер: Система, яка надає доступ до ресурсів або веб-служб.
- Посередницькі компоненти: Елементи, які передають запити та відповіді між агентом користувача та сервером.

RESTful API — це веб-сервіси, які працюють на основі стандартів REST і дозволяють системам запитувати представлення веб-ресурсів та маніпулювати ними через набір операцій, таких як GET, POST, PUT, DELETE. Важливим принципом REST є безстанність, що означає, що кожен запит до сервера містить всю необхідну інформацію, а сервер не зберігає стан сеансу між запитами. Це забезпечує простоту реалізації, масштабованість і підвищення продуктивності, оскільки сервер може обробляти кожен запит незалежно [23].

Для розробки додатку важливим аспектом є доступ до актуальних, точних і глобальних даних. REST API дозволяє інтегруватися з платформами, які надають таку інформацію, забезпечуючи безперервний обмін даними між сервером і клієнтським додатком. Одним із ключових постачальників даних про якість повітря є AirVisual, компанія, що пропонує інноваційні рішення для підтримки здоров'я та благополуччя в умовах забрудненого середовища.

AirVisual використовує передові технології великих даних, штучного інтелекту та Інтернету речей (IoT) для надання інструментів і інформації, необхідних для адаптації до умов забрудненого середовища. Їхні рішення орієнтовані як на внутрішнє, так і на зовнішнє середовище, допомагаючи користувачам оптимізувати своє здоров'я та благополуччя.

Одним із найбільш цінних продуктів компанії є IQAir AirVisual API, який забезпечує доступ до:

- Даних про якість повітря в реальному часі.
- Прогнозів рівнів забруднення.
- Історичних даних про стан повітря.

IQAir AirVisual API надає розробникам можливість інтеграції з глобальними базами даних про якість повітря. Це дозволяє створювати додатки, що адаптуються до змін екологічного середовища [22]. Наприклад:

- Дані в реальному часі: Додаток може отримувати інформацію про концентрації основних забруднювачів (PM2.5, PM10, NO₂ , O₃ , CO тощо) у певному регіоні.
- Прогнозування: Прогнози на кілька днів вперед дозволяють користувачам планувати активності на відкритому повітрі, враховуючи можливі зміни в якості повітря.
- Історичні дані: Аналіз тенденцій допомагає оцінити вплив заходів з покращення якості повітря або сезонних факторів.

```

11 enum AppState { NOT_DOWNLOADED, DOWNLOADING, FINISHED_DOWNLOADING }
12 class MyHomePage extends StatefulWidget {
13   MyHomePage({Key key}) : super(key: key);
14
15   @override
16   _MyHomePageState createState() => _MyHomePageState();
17 }
18
19 class _MyHomePageState extends State<MyHomePage> {
20
21   String _content = 'Unknown';
22   String _key = 'your api key';
23   AirQuality _airQuality;
24   AppState _state = AppState.NOT_DOWNLOADED;
25   AirQualityData _data;
26   final String apiUrl='https://api.waqi.info/feed/Ahmedabad/?token=demo';
27
28
29   @override
30   void initState() {
31     super.initState();
32     _airQuality = new AirQuality(_key);
33     download();
34   }
35
36
37   Future<Map<String,dynamic>> fetchUsers() async {
38
39     var result = await http.get(apiUrl);
40     Map<String, dynamic> jsonBody = json.decode(result.body);
41     return jsonBody;
42   }
43 }
44

```

Рис. 2.2. Фрагмент коду інтеграції IQAir AirVisual API в застосунку

У процесі розробки мобільного застосунку для моніторингу якості повітря використовувалися різноманітні бібліотеки та інструменти, які забезпечують інтеграцію функціоналу й ефективну роботу програми. Одним із ключових елементів є бібліотека `http`, яка забезпечує надсилання HTTP-запитів до REST API для отримання актуальних даних про якість повітря. Ця бібліотека дозволяє виконувати запити до серверів, отримувати відповіді у форматі JSON та забезпечувати обробку отриманих даних для подальшого використання в інтерфейсі.

Для управління станом програми та відокремлення бізнес-логіки від інтерфейсу застосовано архітектурні підходи на основі бібліотек `Provider` або `Bloc` (`Business Logic Component`). Ці бібліотеки сприяють створенню чіткої архітектури, де логіка управління даними ізольована від компонента подання. Це дозволяє забезпечити масштабованість програми, спрощує її тестування та підтримку, а також підвищує стійкість до змін.

Реалізація функцій геолокації здійснюється за допомогою бібліотеки `geolocator`, яка дозволяє отримувати точні геолокаційні дані користувача. Ця функція забезпечує можливість автоматичного визначення місця розташування користувача та надання даних про якість повітря саме для його регіону. Геолокаційні можливості є важливими для підвищення точності й актуальності інформації.

Для інформування користувачів про погіршення якості повітря використовувалася бібліотека `flutter_local_notifications`, яка дозволяє реалізовувати локальні сповіщення. Ці сповіщення надсилаються у разі значного погіршення стану повітря, що забезпечує оперативне інформування користувачів і сприяє прийняттю ними відповідних заходів.

Візуалізація даних про якість повітря, включно з історичними показниками, здійснюється за допомогою бібліотеки `charts_flutter`. Вона дозволяє створювати інтуїтивно зрозумілі графіки, які відображають зміни у показниках якості повітря за певний проміжок часу. Такі графіки сприяють

аналізу даних і допомагають користувачам отримувати більш глибоке уявлення про динаміку змін.

Для роботи з датами, форматуванням часу та локалізацією додатку було використано бібліотеку intl. Вона забезпечує коректне відображення дат і часу відповідно до налаштувань користувача, що є особливо важливим для додатків із міжнародним використанням.

Бібліотека dio використовується для обробки мережових запитів і роботи з даними у форматі JSON. Її основною перевагою є можливість обробки складних запитів, включно з авторизацією та налаштуванням часу очікування відповіді, що підвищує надійність та продуктивність застосунку.

Інтеграція інтерактивних карт, які демонструють рівень забруднення повітря у різних регіонах, здійснюється за допомогою бібліотеки flutter_map. Вона дозволяє візуально відобразити дані на мапі, полегшуючи користувачам аналіз екологічної ситуації в різних локаціях. Карти також забезпечують інтерактивність, що дозволяє переглядати детальні показники для конкретних точок на мапі.

Таким чином, використання зазначених бібліотек та інструментів сприяє створенню високоякісного, функціонального та зручного у використанні мобільного додатку для моніторингу якості повітря. Кожен компонент забезпечує реалізацію окремих функцій, необхідних для інтерактивності, точності й ефективності роботи програми.

Поряд із функціональною реалізацією та інтеграцією API, важливим етапом розробки мобільного застосунку є його дизайн. Створення ефективного дизайну забезпечує зручність використання додатку, підвищує його привабливість для користувачів і сприяє досягненню головної мети — ефективної взаємодії між людиною та продуктом. У цьому контексті UX (користувацький досвід) і UI (інтерфейс користувача) є двома взаємопов'язаними складовими, які формують цілісне враження користувача.

UX (User Experience) означає "користувацький досвід" і стосується того, як користувач сприймає взаємодію з додатком. Це не лише про фізичні

елементи дизайну, а й про загальну простоту, інтуїтивність і комфорт використання продукту. Ефективний UX-дизайн має забезпечувати легкість виконання користувачем своїх завдань, мінімізуючи бар'єри на шляху до досягнення мети.

UI (User Interface) означає "інтерфейс користувача" і охоплює всі візуальні та інтерактивні компоненти додатку, такі як екрани, кнопки, текстові поля та графічні елементи. UI відповідає за естетичність продукту, а також за створення зрозумілих точок взаємодії користувача з додатком [39].

Дизайн має бути зрозумілим навіть для нових користувачів. Це особливо важливо для додатку з еко-моніторингу якості повітря, де швидкість доступу до інформації є критичною.

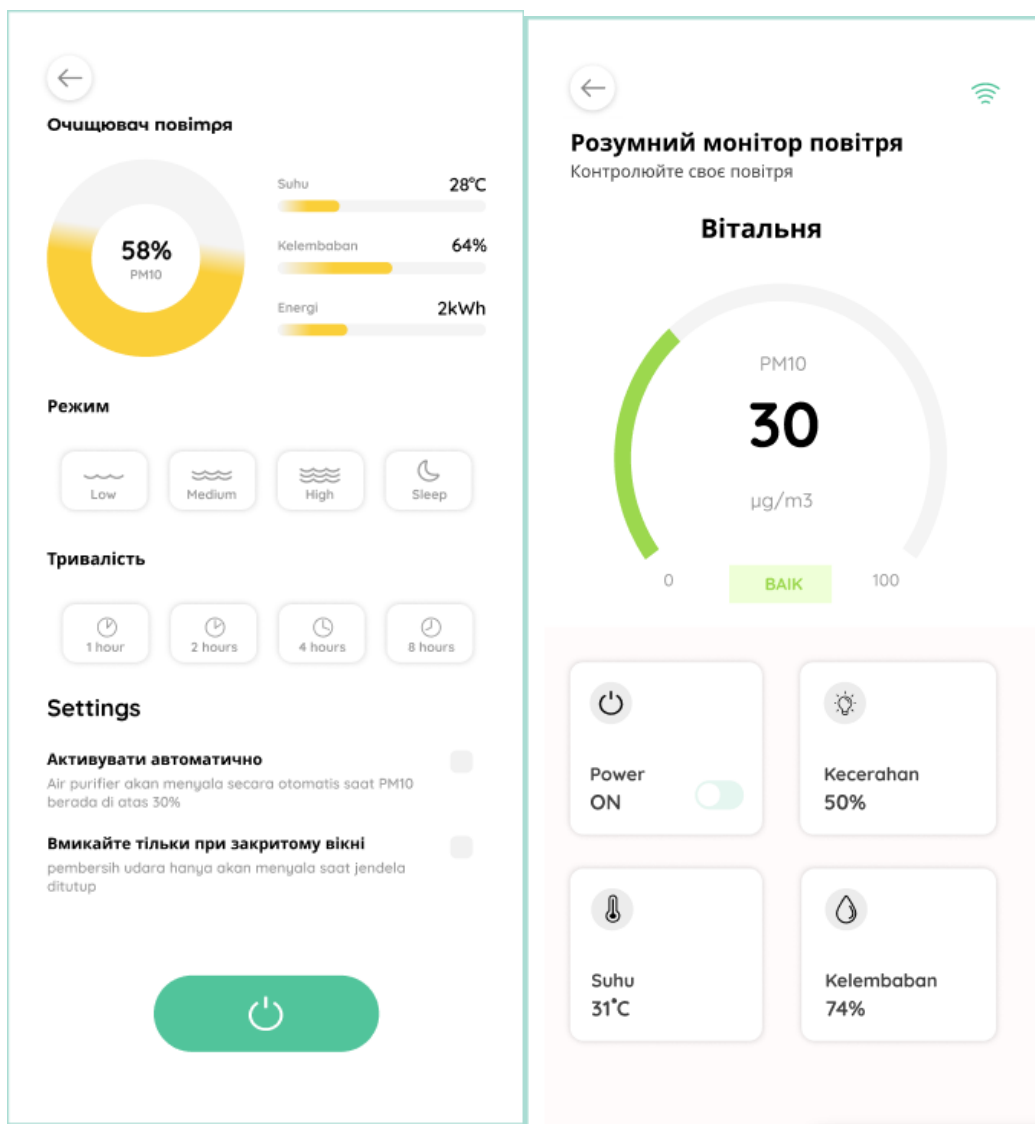


Рис. 2.3. Інтерфейс застосунку для еко-моніторингу якості повітря

Для додатку з еко-моніторингу якості повітря UX та UI відіграють особливу роль. UX-дизайн забезпечує простоту доступу до даних, таких як рівень забруднення в реальному часі, прогнози чи історичні дані. UI-дизайн має забезпечити зручну навігацію по мапі, інтуїтивність фільтрів забруднюючих речовин і легкість налаштувань сповіщень. Особливу увагу слід приділити кольоровій гамі: кольори повинні не лише бути приємними для очей, а й функціонально інформативними (наприклад, індикатори рівня забруднення).

На завершальному етапі розробки мобільного додатку важливо забезпечити локальне зберігання даних, що є необхідною умовою для оптимізації швидкості роботи програми. Локальне сховище дозволяє зменшити залежність від мережевих підключень і забезпечує оперативний доступ до інформації, що особливо актуально для застосунків, які обробляють дані в реальному часі.

SQLite є компактною системою керування реляційними базами даних (RDBMS), яка забезпечує високу продуктивність і ефективність при мінімальних апаратних вимогах. Її архітектура спроектована таким чином, щоб вона могла працювати без необхідності запуску окремого серверного процесу, що робить SQLite ідеальним вибором для застосунків, які потребують інтегрованого рішення для обробки даних.

У мобільних додатках SQLite використовується як локальна база даних для збереження інформації, яка повинна бути доступною навіть у режимі офлайн. Це особливо актуально для додатків, що працюють із динамічними даними, такими як додатки для еко-моніторингу якості повітря. Завдяки своїй легкості та мінімальним вимогам до ресурсів, SQLite забезпечує високу швидкість обробки транзакцій і одночасно підтримує складні SQL-запити, що робить його придатним як для невеликих, так і для масштабованих застосунків.

Однією з головних переваг SQLite є його кросплатформна підтримка. Він працює на основних операційних системах, таких як Android, iOS, Windows, Linux, а також у середовищах веб-розробки. Це дозволяє розробникам

використовувати єдине рішення для керування даними на різних платформах, що спрощує процес розробки та тестування. Крім того, SQLite підтримує транзакції з властивостями ACID (атомарність, узгодженість, ізольованість, довговічність), що гарантує збереження цілісності даних навіть у разі несподіваних збоїв у роботі програми [27].

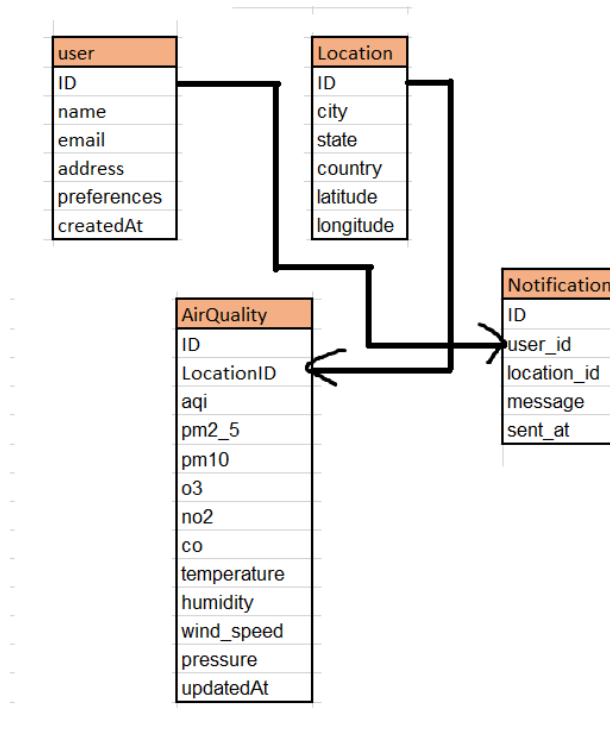


Рис. 2.4. Інфологічна модель бази даних застосунку

Для додатка еко-моніторингу якості повітря SQLite виконує роль локального сховища, де зберігаються дані, отримані через REST API, для подальшого аналізу або перегляду. Це забезпечує стабільність роботи додатку та доступність інформації для користувачів у будь-який момент. Завдяки простоті та ефективності SQLite може бути адаптованим до потреб конкретного проєкту, забезпечуючи баланс між продуктивністю та функціональністю [35].

РОЗДІЛ 3. ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ РОЗРОБЛЕНОГО РІШЕННЯ

Тестування є ключовим етапом у процесі розробки мобільного додатку для еко-моніторингу якості повітря, оскільки воно забезпечує виявлення помилок, недоліків та потенційних проблем, які можуть вплинути на функціональність, безпеку та користувацький досвід. Тестувальники виконують важливу роль у цьому процесі, фактично формуючи основу для гарантування якості програмного продукту. Їхня діяльність безпосередньо сприяє підвищенню рівня довіри користувачів до додатку та репутації компанії.

Оцінка роботи тестувальників є важливим елементом управління якістю, оскільки вона дозволяє виявляти сильні сторони команди, а також визначати сфери, які потребують удосконалення. Цей процес виходить за межі простої перевірки функціоналу продукту; він спрямований на пошук способів підвищення ефективності тестування та оптимізації ресурсів команди. Регулярне оцінювання роботи тестувальників дозволяє ідентифікувати проблемні аспекти у процесі тестування, забезпечувати конструктивний зворотний зв'язок і сприяти професійному розвитку членів команди [20].

Завдяки активній участі тестувальників та оцінці їхньої роботи вдається знизити ризик виникнення критичних помилок у додатку. Це особливо важливо для мобільного застосунку, що надає дані в реальному часі про якість повітря, оскільки користувачі покладаються на точність та оперативність отриманої інформації. Висока якість роботи тестувальників також сприяє покращенню користувацького досвіду, оскільки стабільна робота, інтуїтивно зрозумілий інтерфейс і відсутність критичних помилок створюють позитивне враження від використання додатку.

У процесі тестування мобільного додатку для еко-моніторингу якості повітря тестувальники виступають невід'ємною частиною команди розробки, забезпечуючи відповідність продукту заявленим вимогам і очікуванням користувачів. Їхній внесок є критичним для успіху проєкту, оскільки забезпечує

виявлення й усунення недоліків на ранніх етапах, що дозволяє зменшити ризики, скоротити витрати на доопрацювання та підвищити загальну якість кінцевого продукту [14].

Тестування мобільного додатку для еко-моніторингу якості повітря проводилося у кілька етапів, що дозволило всебічно перевірити його функціональність, продуктивність та відповідність очікуванням користувачів. Спочатку було реалізовано модульне тестування, яке забезпечило перевірку роботи окремих компонентів програми, таких як API-запити, обробка даних та інтерактивні елементи інтерфейсу. Цей етап дозволив виявити та виправити локальні помилки, що могли б спричинити збої у роботі системи

Наступним етапом було інтеграційне тестування, спрямоване на перевірку взаємодії між окремими модулями, зокрема між клієнтською частиною програми та сервером. Особливу увагу приділено тестуванню обміну даними через REST API, включаючи коректність обробки запитів і відповідей, обробку помилок, а також відповідність отриманих даних очікуваній структурі. Було перевірено ефективність механізмів автентифікації та авторизації, правильність роботи HTTP-методів (GET, POST, PUT, DELETE), а також швидкість обробки запитів. Окрім цього, протестовано коректне відображення отриманих даних у користувацькому інтерфейсі, що забезпечило стабільну роботу додатка в умовах реальної взаємодії з сервером.

Також відбувалося тестування продуктивності, яке дозволило оцінити швидкість роботи додатку за різних умов, включаючи низьку якість мережевого з'єднання та значне навантаження на сервери. Проведено стрес-тестування, що імітувало високий обсяг запитів до серверу, з метою перевірки стабільності роботи додатку під час пікових навантажень.

Тестування сумісності проводилося на різних пристроях і операційних системах, включаючи різні версії Android та iOS. Це дозволило переконатися, що додаток працює стабільно на більшості сучасних платформ, забезпечуючи користувачів з різними технічними параметрами однаково якісним досвідом[17].

Ручне тестування інтерфейсу, проведене за участю бета-користувачів, дало змогу зібрати відгуки про зручність навігації, візуальне оформлення та загальну простоту використання програми. Зібрана інформація дозволила внести остаточні корективи, спрямовані на покращення користувацького досвіду.

Таким чином, тестування мобільного додатку проводилося комплексно, з використанням як автоматизованих, так і ручних методів перевірки. Це забезпечило високу якість кінцевого продукту, знизило ризики помилок і гарантувало відповідність додатку вимогам користувачів та сучасним стандартам розробки.

Оцінка тестування мобільного додатку для еко-моніторингу якості повітря проводилася на прикладі Бучанського регіону, де були змодельовані сценарії використання програми у реальних умовах. Цей регіон було обрано для перевірки коректності роботи додатку з географічно конкретизованими даними, що дозволило оцінити його точність, продуктивність і зручність для кінцевих користувачів.

У ході тестування були отримані дані про якість повітря в кількох точках регіону. Наприклад, для міста Буча рівень забруднення дрібнодисперсними частинками $PM_{2.5}$ у ранкові години становив 25 мкг/м^3 , що відповідає помірному рівню забруднення згідно з індексом якості повітря (AQI). Водночас у сусідньому населеному пункті рівень PM_{10} виявився вищим і досяг 60 мкг/м^3 , що наближалось до межі високого рівня забруднення. Додаток коректно відображав ці дані на інтерактивній карті, що дозволило користувачам отримати актуальну інформацію про якість повітря у своїй місцевості [12].

Для перевірки функціональності прогнозування було протестовано здатність додатку надавати прогноз рівнів забруднення на найближчі 24 години. Наприклад, на основі історичних даних та моделей прогнозування додаток передбачав, що рівень $PM_{2.5}$ у Бучі досягне 35 мкг/м^3 у вечірній час через зниження температури та послаблення вітру. Порівняння фактичних

даних із прогнозованими показало, що похибка у прогнозуванні не перевищувала 10%, що відповідає прийнятним стандартам для таких програм.

Додаткове тестування включало перевірку роботи функції сповіщень про критичний рівень забруднення. Для цього було встановлено порогові значення, наприклад, $PM_{2.5} > 50$ $\mu\text{кг}/\text{м}^3$, після чого штучно змодельовані дані перевищення активували сповіщення. Додаток успішно надсилав попередження користувачам, дозволяючи їм вчасно вжити необхідних заходів для захисту здоров'я, таких як обмеження перебування на відкритому повітрі [17].

Також було перевірено функцію локального зберігання історичних даних про якість повітря. Тестування показало, що додаток зберігає дані за останні 30 днів без втрати інформації, дозволяючи користувачам переглядати динаміку змін. Наприклад, у Бучі рівень PM_{10} упродовж останнього тижня поступово знижувався від 65 $\mu\text{кг}/\text{м}^3$ до 40 $\mu\text{кг}/\text{м}^3$, що відображалось у вигляді інтерактивного графіка.

Тестування продуктивності додатку в Бучанському регіоні також продемонструвало його стабільність за умов високого навантаження. Наприклад, при одночасному запиті даних із п'яти різних точок регіону час відповіді сервера залишався у межах 1-1,5 секунд. Це дозволило забезпечити користувачів оперативним доступом до інформації навіть за підвищеного попиту.

Користувацьке тестування в цьому регіоні підтвердило зручність інтерфейсу додатку. Залучені бета-користувачі оцінили можливість перегляду історичних даних, функцію налаштування сповіщень про небезпечний рівень забруднення та простоту використання інтерактивної карти. Наприклад, більшість учасників відзначили, що їм було легко знайти інформацію про поточний рівень забруднення у вибраному населеному пункті або поблизу їхньої локації.

Ці додаткові результати підтверджують надійність та функціональність додатку, що забезпечує не лише оперативний доступ до даних, але й високу зручність і стабільність у щоденному використанні.

ВИСНОВКИ

У межах виконання кваліфікаційної роботи було проведено аналіз сучасних підходів і технологій еко-моніторингу якості атмосферного повітря, розроблено мобільний застосунок для надання користувачам актуальної інформації про стан атмосферного середовища, а також здійснено тестування й оцінку ефективності створеного рішення. Основною метою роботи стало вивчення інструментів і підходів, які сприяють вдосконаленню систем еко-моніторингу, зокрема для забезпечення точного та своєчасного інформування населення.

У першій частині дослідження було проаналізовано сучасні методи еко-моніторингу, що включають використання стаціонарних і персональних засобів вимірювання, а також сенсорних мереж. Виявлено, що традиційні стаціонарні станції вирізняються високою точністю та надійністю отриманих даних, однак їхня висока вартість, складність у встановленні та обмежена географічна охопленість є суттєвими недоліками. Натомість персональні засоби еко-моніторингу забезпечують мобільність і гнучкість у використанні, проте потребують покращення точності для відповідності суворим стандартам екологічного аналізу.

Особливу увагу приділено аналізу існуючих мобільних застосунків для моніторингу забруднення повітря. Зокрема, розглянуто функціональні можливості та особливості таких платформ, як AirVisual і SaveEcoBot. Проведений огляд дозволив виділити основні переваги, що сприяють їхній популярності серед користувачів: інтуїтивно зрозумілий інтерфейс, інтеграція з сенсорними мережами та можливість доступу до глобальних і локальних даних у реальному часі. Водночас ідентифіковано обмеження, які впливають на зручність і точність використання цих додатків, зокрема залежність від якості сенсорів і потребу в стандартизації даних.

На основі проведеного аналізу було розроблено власний мобільний застосунок, що об'єднує переваги сучасних рішень із урахуванням виявлених

недоліків. Результати тестування демонструють ефективність створеного рішення в контексті забезпечення доступності та точності даних для користувачів, а також підтверджують доцільність інтеграції сучасних технологій сенсорного моніторингу у подібні платформи.

У другому розділі здійснено проектування та розробку мобільного застосунку з використанням мови програмування Dart і фреймворку Flutter. Було створено застосунок AirQuality, який забезпечує інтерактивний інтерфейс, обробку даних у реальному часі, інтеграцію з REST API для отримання інформації про якість повітря, а також локальне зберігання даних для перегляду історії. Особливу увагу приділено архітектурі додатку за патерном MVC, що забезпечує масштабованість, легкість у підтримці та зручність для користувачів.

У розділі 3 проведено тестування мобільного додатку, яке включало перевірку функціональності, продуктивності, сумісності та стабільності. На прикладі Бучанського регіону було підтверджено коректність роботи додатку, точність отриманих даних і відповідність прогнозів реальним показникам. Тестування також включало оцінку точності прогнозів на основі історичних даних. Похибка у прогнозуванні виявилася мінімальною (до 10%), що відповідає вимогам до екологічних еко-моніторингових систем. Результати тестування засвідчили високу якість продукту, його готовність до впровадження та здатність задовольнити потреби кінцевих користувачів.

Таким чином, виконана робота дозволила розробити ефективне програмне рішення для еко-моніторингу якості повітря. Створений мобільний застосунок відповідає сучасним вимогам екологічного моніторингу, є зручним у використанні та може бути застосований для підвищення обізнаності громадян про стан повітряного середовища, а також для сприяння прийняттю екологічно обґрунтованих рішень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бабій В. Аналіз систем еко-моніторингу якості повітря у місті Києві : thesis. 2021. URL: <https://er.nau.edu.ua/handle/NAU/50620>
2. Гриневич Л. Методологічні засади цілісності системи еко-моніторингу якості освіти. Освіта і управління. 2010. Т. 13, ч. 4. С. 18–25
3. Забруднення повітря: як перевірити якість повітря онлайн. КОШТ | Медіа про дуже особисті фінанси. URL: <https://kosht.media/yakist-povitria-onlayn-dobirka-resursiv-dlia-perevirky/>
4. Державна служба статистики України. Викиди забруднюючих речовин і парникових газів у атмосферне повітря від стаціонарних джерел забруднення. URL: <https://www.ukrstat.gov.ua/>
5. Застосунок "екозагроза". Офіційний ресурс Міндовкілля. URL: https://ecozagroza.gov.ua/map?Id=37159&layer=air_pollution
6. Застосунок Есо-City. Есо-City Громадський еко-моніторинг стану якості повітря. URL: <https://eco-city.org.ua/?Zoom=11&lat=50.481248&lng=30.412327&station=793&random=8851753>
7. Застосунок ecocity. Ecocity: Reborn. URL: <https://reborn.eco-city.org.ua/?Station=a792c7d8-bbd5-4448-8e6a-2579aaa1c291>
8. Карта радіаційного фону в Україні онлайн: еко-моніторинг радіації - saveecobot. Єдина в Україні екологічна система - saveecobot. URL: https://www.saveecobot.com/radiation-maps#12/50.5553/30.3506/gam_еко-моніторингу_ma/ara+arc+clua+cuc+crc
9. Маймур І. Як перевірити стан повітря онлайн: мапи, міжнародні платформи, чатботи й мобільні застосунки. Журнал великого міста. URL: <https://bzh.life/ua/mesta-i-veshi/1727963056-k-pereviriti-stan-povitrya-onlayn-mapi-mizhnarodni/>
10. Масікевич Ю. Г. Оцінка якості атмосферного повітря гірських лісових екосистем : thesis. 2020. URL: <http://dspace.bsmu.edu.ua:8080/xmlui/handle/123456789/18201>

11. Павелко О., Куликовська Н., Тіменко А. Автоматизована система еко-моніторингу якості повітря. Modeling Control and Information Technologies. 2023. № 6. С. 238–241. URL: <https://doi.org/10.31713/mcit.2023.074>
12. РОЗВИТОК СИСТЕМИ ЕКО-МОНІТОРИНГУ ЯКОСТІ ПОВІТРЯ В М. КИСВІ / В. РАДОВЕНЧИК та ін. Матеріали міжнародної науково-практичної конференції "Екологія. Людина. Суспільство". 2024. С. 262–267. URL: <https://doi.org/10.20535/ehs2710-3315.2024.303975>
13. Система еко-моніторингу якості повітря від української команди yourairtest. Інжинірингова школа Noosphere. URL: <https://noosphereengineering.com/news/yourairtest-air-monitoring-ua>
14. Способи підвищення ефективності засобів тестування програмних систем. Репозитарій КПІ ім. Ігоря Сікорського. URL: <https://ela.kpi.ua/items/f73f9a1f-d5e3-4a99-824c-ba0a62fe3f31>
15. Тарасов Р. «Аналіз повітря в Україні порівняно з Пекіном»: Мобільні застосунки про київське повітря. Радіо Свобода. URL: <https://www.radiosvoboda.org/a/hirshe-nizh-v-pekini-mobilni-dodatky-pro-kyivske-povitria/30561662.html>
16. Твердохлебова Н. Є., Певний Р. С. Еко-моніторинг навколишнього середовища : thesis. 2012. URL: <http://repository.kpi.kharkov.ua/handle/khpi-Press/30050>
17. Типи, практики тестування продуктивності і інструменти та інше!. ZAPTEST. URL: <https://www.zaptest.com/uk/що-таке-тестування-продуктивності-гл>
18. Офіційний сайт застосунку smartcity :. Івано-Франківськ smartcity : Івано-Франківськ-розумне місто, проекти SMART City, події Івано-Франківська, новини Івано-Франківська. URL: <https://smartcity.mvk.if.ua/aboutz>
19. Як ми створили airс – пристрій для еко-моніторингу якості повітря на дорогах в реальному часі. Dou.ua. URL: <https://dou.ua/forums/topic/35107/>
20. Методи оцінки роботи тестувальника: Найкращі підходи. Foxminded. URL: <https://foxminded.ua/yak-otsinyty-robotu-testuvalnyka/>

21. 10+ Best AQI Apps (Air Quality Index Apps). Breathe Safe Air. URL: <https://breathesafeair.com/best-aqi-apps/>
22. Airvisual Platform. Iqair.com. URL: <https://www.iqair.com/air-quality-monitors/api>.
23. Ayalon I. Retrofit and Flutter like a PRO. Medium. URL: <https://medium.com/@ayalon.idan/retrofit-and-flutter-like-a-pro-e80b654545c1>
24. Basic Information about Air Emissions Monitoring | US EPA. US EPA. URL: <https://www.epa.gov/air-emissions-monitoring-knowledge-base/basic-information-about-air-emissions-monitoring>
25. Dart programming language. Dart programming language | Dart. URL: <https://dart.dev/>
26. Emerging air quality monitoring methods / Y. Li et al. Air Quality Monitoring and Advanced Bayesian Modeling. 2023. P. 105–172. URL: <https://doi.org/10.1016/b978-0-323-90266-3.00005-4>
27. Flutter Fusion. Architecture Patterns in Flutter: An In-Depth Guide. Medium. URL: <https://medium.com/@yusrasajjad613/architecture-patterns-in-flutter-an-in-depth-guide-0ca2d65c723c>
28. Flutter vs Dart: Key Differences Explained. The forefront of innovation | Flatirons. URL: <https://flatirons.com/blog/dart-vs-flutter/>
29. Fossil fuel air pollution responsible for 1 in 5 deaths worldwide | Harvard T.H. Chan School of Public Health. Harvard T.H. Chan School of Public Health. URL: <https://hsph.harvard.edu/environmental-health/news/fossil-fuel-air-pollution-responsible-for-1-in-5-deaths-worldwide/>
30. Hanansani. Harnessing the Power of Flutter with Android Studio: A Comprehensive Guide. Medium. URL: <https://medium.com/@hanansani2002/harnessing-the-power-of-flutter-with-android-studio-a-comprehensive-guide-ab58a7b8d984>

31. How To Evaluate Effectiveness Of Your QA Testing Strategy - qaoncloud. Qaoncloud. URL: <https://www.qaoncloud.com/blog/how-evaluate-effectiveness-your-qa-testing-strategy>
32. Ionel I., Popescu F. Methods for Online Monitoring of Air Pollution Concentration. Air Quality. 2010. URL: <https://doi.org/10.5772/9754>
33. Khan S. Unleashing Creativity: Exploring Architecture Patterns in Flutter. Medium. URL: <https://medium.com/@samra.sajjad0001/unleashing-creativity-exploring-architecture-patterns-in-flutter-12b7465bc927>
34. Liberti A. Modern methods for air pollution monitoring. Pure and Applied Chemistry. 2015. Vol. 44, no. 3. P. 519–534. URL: <https://doi.org/10.1351/pac197544030519>
35. Nayak S. K. Flutter Development: VS Code vs. Android Studio. Medium. URL: <https://shivamkumarnayak.medium.com/flutter-development-vs-code-vs-android-studio-9e93f4917cc>
36. Retrofit Implementation In Flutter - Flutterexperts. Flutterexperts. URL: <https://flutterexperts.com/retrofit-implementation-in-flutter/>
37. Retrofit Implementation In Flutter - Flutterexperts. Flutterexperts. URL: <https://flutterexperts.com/retrofit-implementation-in-flutter/>
38. Senkiv V. РОЗРОБКА СИСТЕМИ ЕКО-МОНІТОРИНГУ АТМОСФЕРНОГО ПОВІТРЯ МІСТА ДРОГОБИЧА. Acta Carpathica. 2022. № 1. С. 52–61. URL: <https://doi.org/10.32782/2450-8640.2021.1.7>
39. The purpose and importance of air quality monitoring. Low-Cost Air Quality Monitoring & Measurement | Clarity Movement Co. URL: <https://www.clarity.io/blog/what-is-air-quality-monitoring-why-is-it-important>
40. Official site Android. Android. URL: <https://www.android.com/what-is-android/>
41. Мадей Б.В. Актуальність розробки іт-засобів моніторингу якості повітря /Мадей Б.В., Погорєловська І.Д. //Цифрова трансформація фінансової системи України та країн V-4 в умовах євроінтеграції. DIGITAL

TRANSFORMATION OF THE FINANCIAL SYSTEM OF UKRAINE AND V4 COUNTRIES IN THE CONTEXT OF EUROPEAN INTEGRATION. Збірник тез IV міжн. Наук.-практ. Інтернет-конф. (Дубляни, 16 травня 2024 р.). Частина II. Дубляни: ЛНУП, 2024. С. 323-326.
<https://lnup.edu.ua/attachments/article/3201/%D0%97%D0%91%D0%86%D0%A0%D0%9D%D0%98%D0%9A%20%D0%A2%D0%95%D0%97%202%20%D1%87%D0%B0%D1%81%D1%82%D0%B8%D0%BD%D0%B0.pdf>

ДОДАТКИ

Додаток А

```

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.theindianinnovation.air_pollution">
    <!-- io.flutter.app.FlutterApplication is an android.app.Application that
         calls FlutterMain.startInitialization(this); in its onCreate method.
         In most cases you can leave this as-is, but you if you want to provide
         additional functionality it is fine to subclass or reimplement
         FlutterApplication and put your custom class here. -->
    <application
        android:name="io.flutter.app.FlutterApplication"
        android:label="air_pollution"
        android:icon="@mipmap/ic_launcher">
        <activity
            android:name=".MainActivity"
            android:launchMode="singleTop"
            android:theme="@style/LaunchTheme"

            android:configChanges="orientation|keyboardHidden|keyboard|screenSize|smallestScreenSize|local
            e|layoutDirection|fontScale|screenLayout|density|uiMode"

            android:hardwareAccelerated="true"
            android:windowSoftInputMode="adjustResize">
            <!-- Specifies an Android theme to apply to this Activity as soon as
                 the Android process has started. This theme is visible to the user
                 while the Flutter UI initializes. After that, this theme continues
                 to determine the Window background behind the Flutter UI. -->
        <meta-data
            android:name="io.flutter.embedding.android.NormalTheme"
            android:resource="@style/NormalTheme"
        />
        <!-- Displays an Android View that continues showing the launch screen
             Drawable until Flutter paints its first frame, then this splash

```

screen fades out. A splash screen is useful to avoid any visual

Продовження додатку А

gap between the end of Android's launch screen and the painting of

Flutter's first frame. -->

```
<meta-data
  android:name="io.flutter.embedding.android.SplashScreenDrawable"
  android:resource="@drawable/launch_background"
/>
```

```
<intent-filter>
  <action android:name="android.intent.action.MAIN"/>
  <category android:name="android.intent.category.LAUNCHER"/>
</intent-filter>
```

```
</activity>
```

```
<!-- Don't delete the meta-data below.
```

This is used by the Flutter tool to generate GeneratedPluginRegistrant.java -->

```
<meta-data
  android:name="flutterEmbedding"
  android:value="2" />
```

```
</application>
```

```
</manifest>
```

```
package com.theindianinnovation.air_pollution
```

```
import io.flutter.embedding.android.FlutterActivity
```

```
class MainActivity: FlutterActivity() {
}
```

```
import 'dart:ui';
```

```
import 'package:flutter/material.dart';
```

```
class AppColors {
  static const Color primary = Colors.white;
```

```
static const Color background = Colors.white;
```

Продовження додатку А

```
static const Color textPrimary = Color(0xFF263238);
}
import 'package:flutter/material.dart';
import 'package:air_pollution/app/app_colors.dart';

class AppThemeDataFactory {
  static ThemeData prepareThemeData() => ThemeData(
    brightness: Brightness.light,
    primaryColor: AppColors.primary,
    accentColor: AppColors.textPrimary,
    backgroundColor: AppColors.background,
    fontFamily: "Roboto",
    textTheme: TextTheme(
      title: TextStyle(
        fontSize: 72,
        fontWeight: FontWeight.w500,
        color: AppColors.textPrimary,
      ),
      subtitle: TextStyle(
        fontSize: 28,
        fontWeight: FontWeight.w600,
        color: AppColors.textPrimary,
      ),
      subhead: TextStyle(
        fontSize: 24,
        fontWeight: FontWeight.w600,
        color: AppColors.textPrimary,
      ),
    ),
  );
}
```

```

    body1: TextStyle(
      fontSize: 16,

fontWeight: FontWeight.w500,
      color: AppColors.textPrimary,
    ),
  ),
);
}

import 'package:flutter/material.dart';
import 'package:air_pollution/ui/home_page.dart';
import 'package:air_pollution/app/app_theme.dart';

class SmogeApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: AppThemeDataFactory.prepareThemeData(),
      home: MyHomePage(),
    );
  }
}

import 'package:flutter/material.dart';
import 'package:air_pollution/widgets/animated_percentage_widget.dart';
import 'package:air_pollution/widgets/activities/activity_widget.dart';
import 'package:air_pollution/widgets/activities/activities_widget.dart';
import 'package:air_pollution/widgets/video_player_widget.dart';
import 'package:air_pollution/app/app_theme.dart';
import 'package:http/http.dart' as http;
import 'dart:convert';

```

Продовження додатку А

```
import 'package:air_quality/air_quality.dart';
```

Продовження додатку А

```
enum AppState { NOT_DOWNLOADED, DOWNLOADING, FINISHED_DOWNLOADING }
```

```
class MyHomePage extends StatefulWidget {
```

```
  MyHomePage({Key key}) : super(key: key);
```

```
  @override
```

```
  _MyHomePageState createState() => _MyHomePageState();
```

```
}
```

```
class _MyHomePageState extends State<MyHomePage> {
```

```
  String _content = 'Unknown';
```

```
  String _key = 'your api key';
```

```
  AirQuality _airQuality;
```

```
  AppState _state = AppState.NOT_DOWNLOADED;
```

```
  AirQualityData _data;
```

```
  final String apiUrl='https://api.waqi.info/feed/Ahmedabad/?token=demo';
```

```
  @override
```

```
  void initState() {
```

```
    super.initState();
```

```
    _airQuality = new AirQuality(_key);
```

```
    download();
```

```
}
```

```
Future<Map<String,dynamic>> fetchUsers() async {

    var result = await http.get(apiUrl);
    Map<String, dynamic> jsonBody = json.decode(result.body);

    return jsonBody;

}
```

Продовження додатку А

```
String _name(dynamic user){
    return (user['data'][0]['station']['name']).toString();

}
```

```
Future download() async {

    _data = null;
    setState(() {
        _state = AppState.DOWNLOADING;
    });
    AirQualityData feedFromCity = await _airQuality.feedFromCity("Kiev");

    // Update screen state
    setState(() {
        _data=(feedFromCity);
        _state = AppState.FINISHED_DOWNLOADING;
    });
}
```

```
Widget _buildTitle() =>
```

```
  Text(
    _data.place,
    textAlign: TextAlign.center,
    style: Theme
```

Продовження додатку А

```
    .of(context)
    .textTheme
    .subhead,
  );
```

```
Widget _buildAirPollutionContent() =>
```

```
  Expanded(
    child: Column(

      mainAxisAlignment: MainAxisAlignment.center,
      children: <Widget>[
        AnimatedPercentageWidget(
          fromValue: 0.0,
          toValue: double.parse(_data.airQualityIndex.toDouble().toString()),
        ),
        Text(
          _data.airQualityLevel.toString(),
          style: Theme
            .of(context)
            .textTheme
            .subtitle,
        ),
      ],
    ),
```

```
);
```

```
Widget _buildActivitiesWidget() =>
```

```
  Padding(
```

```
    padding: EdgeInsets.only(bottom: 16),
```

```
    child: ActivitiesWidget(
```

Продовження додатку А

```
      activities: {
```

```
        ActivityType.walking:
```

```
        _data.airQualityIndex<=60?ActivityQuality.good:ActivityQuality.bad,
```

```
        ActivityType.running:
```

```
        _data.airQualityIndex<=60?ActivityQuality.good:ActivityQuality.bad,
```

```
        ActivityType.biking:
```

```
        _data.airQualityIndex<=60?ActivityQuality.good:ActivityQuality.bad,
```

```
      },
```

```
    ),
```

```
  );
```

```
Widget _buildDetailsWidget() =>
```

```
  Expanded(
```

```
    child: Column(
```

```
      mainAxisAlignment: MainAxisAlignment.center,
```

```
      children: <Widget>[
```

```
        Opacity(
```

```
          opacity: 0.5,
```

```
          child: Text(
```

```
            "Details",
```

```
            style: Theme
```

```
              .of(context)
```

```
              .textTheme
```

```
              .body1,
```

```
          ),
```

```

    ),
    SizedBox(height: 5),
    Image.asset("assets/images/icon_arrow_down.png",
      width: 7, height: 7),
  ],
),

```

Продовження додатку А

```

);

Widget _buildExpandedContent() =>
  Expanded(
    child: Column(
      children: <Widget>[
        Expanded(child: Container()),
        _buildAirPollutionContent(),
        _buildDetailsWidget(),
      ],
    ),
  );

```

```

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('User List'),
    ),
    body: Container(
      child: FutureBuilder<Map<String,dynamic>>(

```

```

future: fetchUsers(),
builder: (BuildContext context, AsyncSnapshot snapshot) {
  if(snapshot.hasData){
    print((snapshot.data));
    return Text(_name(snapshot.data));
  }else {
    return Center(child: CircularProgressIndicator());

```

Продовження додатку А

```

    }}
  ),
),
);
}
}
import 'package:flutter/material.dart';
import 'package:air_pollution/app/smoge_app.dart';

void main() {
  runApp(SmogeApp());
}
import 'package:flutter/material.dart';

class AnimatedPercentageWidget extends StatefulWidget {
  const AnimatedPercentageWidget({
    Key key,
    @required this.fromValue,
    @required this.toValue,
    this.duration = const Duration(milliseconds: 1500),
  }) : assert(fromValue != null),
        assert(toValue != null),

```

```

    assert(fromValue <= toValue),
    assert(duration != null),
    super(key: key);

```

```

final double fromValue;
final double toValue;
final Duration duration;

```

Продовження додатку А

```

@override
State<StatefulWidget> createState() => _AnimatedPercentageWidgetState();
}

class _AnimatedPercentageWidgetState extends State<AnimatedPercentageWidget>
    with SingleTickerProviderStateMixin {
    Animation<double> _animation;
    AnimationController _controller;

    String _number;

    @override
    void initState() {
        super.initState();

        _controller = AnimationController(
            duration: widget.duration,
            vsync: this,
        );

        _animation = Tween<double>(begin: widget.fromValue, end: widget.toValue)

```

```

        .animate(_controller)
        ..addListener(() {
          setState(() {
            _number = _animation.value.toStringAsFixed(0);
          });
        });

        _controller.forward();
      }

```

Продовження додатку А

```

@override
Future<void> dispose() async {
  _controller.dispose();

```

```

    super.dispose();
  }

```

```

@override
Widget build(BuildContext context) =>
  Text("$_number%", style: Theme.of(context).textTheme.title);
}

```

```

import 'package:flutter/material.dart';
import 'package:video_player/video_player.dart';

```

```

class VideoPlayerWidget extends StatefulWidget {
  const VideoPlayerWidget({
    Key key,
    @required this.videoPath,
  }) : assert(videoPath != null),
        super(key: key);

```

```
final String videoPath;
```

```
@override
```

```
_VideoPlayerState createState() => _VideoPlayerState();
}
```

```
class _VideoPlayerState extends State<VideoPlayerWidget> {
  VideoPlayerController _controller;
```

Продовження додатку А

```
@override
```

```
void initState() {
  super.initState();
  _controller = VideoPlayerController.asset(widget.videoPath)
    ..initialize().then((_) {
      _controller
        ..play()
        ..setLooping(true);

      setState(() {});
    });
}
```

```
@override
```

```
Widget build(BuildContext context) => VideoPlayer(_controller);
```

```
@override
```

```
void dispose() {
  _controller.dispose();
}
```

```

    super.dispose();
  }
}
import 'package:flutter/material.dart';
import 'package:air_pollution/widgets/activities/activity_widget.dart';

abstract class _Constants {
  static const double iconSize = 30;
  static const double itemSpacing = 20;
  static const double badgeSize = 15;

```

Продовження додатку А

```

class ActivitiesWidget extends StatelessWidget {
  const ActivitiesWidget({
    Key key,
    this.activities,
  }) : super(key: key);

  final Map<ActivityType, ActivityQuality> activities;

  @override
  Widget build(BuildContext context) => Row(
    mainAxisAlignment: MainAxisAlignment.spaceEvenly,
    children: _buildActivities(),
  );

  Widget _buildSpacing() => SizedBox(
    width: _Constants.itemSpacing - _Constants.badgeSize / 2,

```

```
);
```

```
List<Widget> _buildActivities() {
```

```
  List<Widget> widgets = [SizedBox(width: _Constants.itemSpacing)];
```

```
  activities.forEach((activityType, activityQuality) {
```

```
    widgets
```

```
    ..add(
```

```
      ActivityWidget(
```

```
        activityType: activityType,
```

```
        activityQuality: activityQuality,
```

```
      ),
```

```
    )
```

```
    ..add(_buildSpacing());
```

```
  });
```

```
  return widgets;
```

```
}
```

```
}
```

```
import 'package:flutter/material.dart';
```

```
import 'package:air_pollution/app/app_theme.dart';
```

```
abstract class _Constants {
```

```
  static const double iconSize = 30;
```

```
  static const double itemSpacing = 20;
```

```
  static const double badgeSize = 15;
```

```
}
```

```
enum ActivityType { walking, running, biking }
```

```
enum ActivityQuality { good, bad }
```

Продовження додатку А

```

class ActivityWidget extends StatelessWidget {
  const ActivityWidget({Key key, this.activityType, this.activityQuality})
    : super(key: key);

  final ActivityType activityType;
  final ActivityQuality activityQuality;

  @override
  Widget build(BuildContext context) => Expanded(
    child: Stack(
      children: <Widget>[
        Padding(
          padding: EdgeInsets.only(
            top: _Constants.badgeSize / 2,
            right: _Constants.badgeSize / 2,
          ),
          child: _buildActivityContainer(
            context,
            _imageForActivityType(activityType),
            _titleForActivityType(activityType),
          ),
        ),
        Positioned(
          right: 0,
          top: 0,
          child: _buildWarningBadge(activityQuality),
        ),
      ],
    ),
  );

```

Продовження додатку А

);

Widget _buildActivityContainer(

BuildContext context,

Image activityIcon,

String title,

) =>

Container(

decoration: BoxDecoration(

color: Colors.white,

borderRadius: BorderRadius.all(Radius.circular(10)),

boxShadow: [

BoxShadow(

color: Colors.grey.withAlpha(60),

Продовження додатку А

blurRadius: 5.0,

spreadRadius: 1.0,

),

],

),

child: Padding(

padding: EdgeInsets.symmetric(vertical: 12, horizontal: 16),

child: Center(

child: _buildActivityColumn(context, activityIcon, title),

),

),

);

Widget _buildActivityColumn(

BuildContext context,

```

Image activityIcon,
String title,
) =>
Column(
  mainAxisAlignment: MainAxisAlignment.center,
  children: <Widget>[
    SizedBox(
      width: _Constants.iconSize,
      height: _Constants.iconSize,
      child: activityIcon,
    ),
    SizedBox(height: 10),
    Opacity(
      opacity: 0.75,
      child: Text(

```

Продовження додатку А

```

title,
  style: Theme
    .of(context)
    .textTheme
    .body1,
  maxLines: 1,
  overflow: TextOverflow.ellipsis,
),
)
],
);

```

```

Widget _buildWarningBadge(ActivityQuality quality) {
  switch (quality) {

```

```

case ActivityQuality.good:
    return Container();
case ActivityQuality.bad:
    return Container(
        width: _Constants.badgeSize,
        height: _Constants.badgeSize,
        child: Image.asset("assets/images/icon_warning.png"),
    );
}

return Container();
}

Image _imageForActivityType(ActivityType activityType) {
    switch (activityType) {
        case ActivityType.walking:

            return Image.asset("assets/images/icon_walking.png");
        case ActivityType.running:
            return Image.asset("assets/images/icon_run.png");
        case ActivityType.biking:
            return Image.asset("assets/images/icon_cycle.png");
    }

    return null;
}

String _titleForActivityType(ActivityType activityType) {
    switch (activityType) {
        case ActivityType.walking:

```

Продовження додатку А

```

        return "Walking";
    case ActivityType.running:
        return "Running";
    case ActivityType.biking:
        return "Biking";
    }

    return null;
}
}

// This is a basic Flutter widget test.
//
// To perform an interaction with a widget in your test, use the WidgetTester
// utility that Flutter provides. For example, you can send tap and scroll
// gestures. You can also use WidgetTester to find child widgets in the widget
// tree, read text, and verify that the values of widget properties are correct.

```

Продовження додатку А

```

import 'package:flutter/material.dart';
import 'package:flutter_test/flutter_test.dart';

import 'package:air_pollution/main.dart';

void main() {
  testWidgets('Counter increments smoke test', (WidgetTester tester) async {
    // Build our app and trigger a frame.
    await tester.pumpWidget(MyApp());

    // Verify that our counter starts at 0.
    expect(find.text('0'), findsOneWidget);
  });
}

```

```

expect(find.text('1'), findsNothing);

// Tap the '+' icon and trigger a frame.
await tester.tap(find.byIcon(Icons.add));
await tester.pump();

// Verify that our counter has incremented.
expect(find.text('0'), findsNothing);
expect(find.text('1'), findsOneWidget);
});
}
import 'package:airquality/services/pollution.dart';
import 'package:airquality/widgets/healt_recomendation.dart';
import 'package:airquality/widgets/weather_update.dart';
import 'package:flutter/material.dart';
import 'package:flutter_speedometer/flutter_speedometer.dart';
import 'package:google_fonts/google_fonts.dart';

```

Продовження додатку А

```

class HomePage extends StatefulWidget {
  final locationstats;

  HomePage({
    this.locationstats,
  });

  @override
  _HomePageState createState() => _HomePageState();
}

```

```

class _HomePageState extends State<HomePage> {
  PollutionModel pollutionModel = PollutionModel();

  @override
  void initState() {
    super.initState();
    updateUI(widget.locationstats);
    print(widget.locationstats);
  }

  var city;
  var state;
  var country;
  var temprature;
  var humidity;
  var aqIndex;
  String precaution;

  String cityName;

  void updateUI(dynamic locationData) {
    city = locationData['data']['city'];
    state = locationData['data']['state'];
    country = locationData['data']['country'];
    temprature = locationData['data']['current']['weather']['tp'];
    humidity = locationData['data']['current']['weather']['hu'];
    aqIndex = locationData['data']['current']['pollution']['aqius'];
    precaution = pollutionModel.getPollution(aqIndex);
  }

```

Продовження додатку А

```

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text(
        'AIR VISUAL',
        style: GoogleFonts.architectsDaughter(color: Colors.black),
      ),
      centerTitle: true,
      backgroundColor: Colors.white,
      elevation: 0,
    ),
    backgroundColor: Colors.white,
    body: SingleChildScrollView(
      child: SafeArea(
        child: Padding(
          padding: const EdgeInsets.all(8.0),
          child: Column(
            children: [
              SizedBox(

height: 8,
            ),
              WeatherUpdates(
                temprature: temprature,
                humidity: humidity,
                city: city,
                state: state,
                country: country,
            ),

```

Продовження додатку А

```

    SizedBox(
      height: 20,
    ),
    Speedometer(
      size: 200,
      minValue: 0,
      maxValue: 310,
      currentValue: aqIndex,
      warningValue: 150,
      backgroundColor: Colors.white,
      meterColor: Colors.greenAccent,
      warningColor: Colors.red,
      kimColor: Colors.black,
      displayNumericStyle: GoogleFonts.gayathri(
        fontSize: 28,
        color: Colors.blue,
      ),
      displayText: 'Air Quality Index',
      displayTextStyle: GoogleFonts.robotoMono(
        fontSize: 18,
      ),
    ),

```

Продовження додатку А

```

    ),
    SizedBox(
      height: 10,
    ),
    Container(
      child: Text(
        '$precaution',
        style: GoogleFonts.teko(

```

```

        fontSize: 35,
      ),
    ),
  ),
  SizedBox(
    height: 10,
  ),
  Container(
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        Text(
          'HEALTH RECOMMENDATIONS',
          style: GoogleFonts.teko(
            color: Colors.blue,
            fontSize: 25,
          ),
        ),
        Text(
          'How to protect yourself from pollution in $city ?',
          style:
            GoogleFonts.openSans(fontWeight: FontWeight.bold),
        ),
      ],
    ),
    SizedBox(
      height: 20,
    ),
    NotificationListener<OverscrollIndicatorNotification>(
      // ignore: missing_return
      onNotification: (overscroll) {

```

Продовження додатку А

```

        overscroll.disallowGlow();
    },
    child: SingleChildScrollView(
      scrollDirection: Axis.horizontal,
      child: Row(
        children: [
          HealthRecomedation(
            healthrec:
              'Sensitive groups should wear a mask outside',
            howto: 'GET A MASK',
            image: 'assets/images/mask.png',
          ),
          SizedBox(
            width: 10,
          ),
          HealthRecomedation(
            healthrec: 'Run an air purifier',
            howto: 'GET AN AIR PURIFIER',
            image: 'assets/images/airpurifier.png',
          ),
        ],
      ),
    ),
  ),
),
],
),
),
],
),

```

Продовження додатку А

```

    ),
    ),
    ),
  );
}
}

import 'package:airquality/screens/homepage.dart';
import 'package:airquality/services/networking.dart';
import 'package:flutter/material.dart';
import 'package:flutter_spinkit/flutter_spinkit.dart';

import '../services/location.dart';

class LoadingScreen extends StatefulWidget {
  @override
  _LoadingScreenState createState() => _LoadingScreenState();
}

class _LoadingScreenState extends State<LoadingScreen> {
  double longitude;
  double latitude;

  Future<void> getLocationData() async {
    Location location = Location();

    await location.getCurrentLocation();

    latitude = location.latitutde;
    longitude = location.longitude;

    NetworkHelper networkHelper = NetworkHelper(
      'http://api.airvisual.com/v2/nearest_city?lat=$latitude&lon=$longitude&key=1be424d4-4480-4930-a76a-c43df2e17640');

```

Продовження додатку А

```

var locationData = await networkHelper.getData();
Navigator.push(context, MaterialPageRoute(builder: (ctx) {
  return HomePage(
    locationstats: locationData,
  );
}));
}

```

```

@override
void initState() {
  getLocationData();

  super.initState();
}

```

```

@override
Widget build(BuildContext context) {
  return Scaffold(
    body: Center(
      child: SpinKitChasingDots(
        color: Colors.black,
        size: 50,
      ),
    ),
  ),

```

Продовження додатку А

```

),
);
}
}
import 'package:geolocator/geolocator.dart';

```

```

class Location {
  double longitude;
  double latitutde;

  Future<void> getCurrentLocation() async {
    try {
      Position position =
        await getCurrentPosition(desiredAccuracy: LocationAccuracy.best);

      longitude = position.longitude;
      latitutde = position.latitude;
    } catch (e) {}
  }
}

import 'dart:convert';

import 'package:http/http.dart' as http;

class NetworkHelper {
  final String url;
  NetworkHelper(this.url);

  Future getData() async {
    http.Response response = await http.get(url);

    if (response.statusCode == 200) {
      String data = response.body;
      return jsonDecode(data);
    } else {

```

Продовження додатку А

```

    print(response.statusCode);
  }
}

class PollutionModel {
  String getPollution(int aqi) {
    if (aqi >= 0 && aqi <= 50) {
      return 'Good, Air quality is satisfactory, and air pollution poses little or no risk';
    } else if (aqi >= 51 && aqi <= 100) {
      return 'Moderate, Air Quality is Acceptable';
    } else if (aqi >= 101 && aqi <= 150) {
      return 'Unhealthy for Sensitive Group';
    } else if (aqi >= 151 && aqi <= 200) {
      return 'Unhealthy, Members of Sensitive group may experience severe effects';
    } else if (aqi >= 201 && aqi <= 300) {
      return 'Health alert: The risk of health effects is increased for everyone';
    } else {
      return 'Hazardous, Everyone is more likely to be Affected ';
    }
  }
}

import 'package:flutter/material.dart';
import 'package:google_fonts/google_fonts.dart';

class HealthRecomedation extends StatelessWidget {
  final String healthrec;

  final String howto;
  final String image;

```

Продовження додатку А

```
const HealthRecomedation({Key key, this.healthrec, this.howto, this.image})
  : super(key: key);
```

```
@override
```

```
Widget build(BuildContext context) {
```

```
  return GestureDetector(
```

```
    child: Container(
```

```
      decoration: BoxDecoration(
```

```
        color: Colors.pink[50],
```

```
        borderRadius: BorderRadius.circular(10),
```

```
      ),
```

```
      child: Padding(
```

```
        padding: const EdgeInsets.all(8.0),
```

```
        child: Flex(
```

```
          crossAxisAlignment: CrossAxisAlignment.start,
```

```
          direction: Axis.vertical,
```

```
          children: [
```

```
            Image.asset(
```

```
              '$image',
```

```
              scale: 1.5,
```

```
            ),
```

```
            SizedBox(
```

```
              height: 15,
```

```
            ),
```

```
            Text(
```

```
              '$healthrec',
```

```
              style: GoogleFonts.aladin(
```

```
                fontSize: 18,
```

```
            ),
```

Продовження додатку А

```

    ),
    SizedBox(
      height: 5,
    ),
    Text(
      '$howto',
      style: GoogleFonts.abel(
        color: Colors.grey,
        fontSize: 20,
        fontWeight: FontWeight.w600),
    ),
    SizedBox(height: 10),
  ],
),
),
),
);
}
}

import 'package:flutter/material.dart';
import 'package:google_fonts/google_fonts.dart';
import 'package:weather_icons/weather_icons.dart';

class WeatherUpdates extends StatelessWidget {
  const WeatherUpdates({
    Key key,
    @required this.temprature,
    @required this.humidity,

    @required this.city,

```

Продовження додатку А

```

    @required this.state,
    @required this.country,
  }) : super(key: key);

```

```

final int temprature;
final int humidity;
final String city;
final String state;
final String country;

```

```

@override
Widget build(BuildContext context) {
  return Container(
    decoration: BoxDecoration(
      color: Colors.pink[50],
      borderRadius: BorderRadius.circular(10),
    ),
    height: MediaQuery.of(context).size.height / 4,
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.center,
      mainAxisAlignment: MainAxisAlignment.spaceAround,
      children: [
        Row(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            BoxedIcon(
              WeatherIcons.cloud,
              size: 30,
            ),

```

Продовження додатку А

```

    SizedBox(
      width: 25,
    ),
    Text(
      '$temprature °C',
      style: GoogleFonts.architectsDaughter(
        fontSize: 35, fontWeight: FontWeight.bold),
    ),
  ],
),
Row(
  mainAxisAlignment: MainAxisAlignment.center,
  children: [
    BoxedIcon(
      WeatherIcons.humidity,
      size: 30,
    ),
    SizedBox(
      width: 25,
    ),
    Text(
      '$humidity %',
      style: GoogleFonts.architectsDaughter(
        fontSize: 35, fontWeight: FontWeight.bold),
    ),
  ],
),
Text(
  '$city | $state | $country',
  style: TextStyle(color: Colors.black, fontSize: 25),

```

Продовження додатку А

```

    ),
    ],
    ),
);
}
}

import 'package:airquality/screens/loading_screen.dart';
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    SystemChrome.setPreferredOrientations([
      DeviceOrientation.portraitUp,
      DeviceOrientation.portraitDown,
    ]);
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Flutter Demo',
      theme: ThemeData(),
      home: LoadingScreen(),
    );
  }
}

```