Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(навчально-науковий інститут)
факультет інформаційних технологій
(факультет)
Кафедра інформаційних технологій та комп'ютерної інженерії
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи ступеня бакалавра
(бакалавра, магістра)

Здобувача вищої освіти Савенка Дмитра Олександровича
(ПІБ)

академічної групи 126-21-2
(шифр)

спеціальності 126 «Інформаційні системи та технології»
(код і назва спеціальності)

спеціалізації за освітньо-професійною (освітньо-науковою) програмою
(за наявності)

(офіційна назва)

на тему Розробка інформаційної системи оповіщення про екологічну небезпеку

(назва за наказом ректора)

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	Молодець Б.В.			
розділів:				

Рецензент				

Нормоконтролер				

Дніпро
20 25

ЗАТВЕРДЖЕНО:
завідувач кафедри
__інформаційних технологій та комп'ютерної___
інженерії
(повна назва)

_____ _Гнатушенко В.В.
(підпис) (ініціали та прізвище)

«_____» _____ 2025 року

ЗАВДАННЯ

на кваліфікаційну роботу
ступеня___бакалавра___
(бакалавра, магістра)

здобувача вищої освіти___Савенко Д.О._____ **академічної групи**_____126-21-2_____
(прізвище та ініціали) (шифр)

спеціальності_____126 «Інформаційні системи та технології»_____

за освітньою-професійною програмою _____
(за наявності)

_____

на тему _Розробка інформаційної системи оповіщення про екологічну небезпеку_____

_____

затверджену наказом ректора НТУ «Дніпровська політехніка» від _05.05.2025_ № _336-с_

Розділ	Зміст	Термін виконання
Аналіз стану рішення області завдання	Аналіз існуючих систем оповіщення про екологічну небезпеку, визначення функціональних та нефункціональних вимог, вибір технологій	14.04.2025
Проєктні рішення	Розробка архітектури інформаційної системи, реалізація серверного модуля, реалізація клієнтського інтерфейсу	13.05.2025
Тестування та результати роботи	Проведення тестування коректності відображення даних, перевірка працездатності сповіщень, аналіз продуктивності системи, висновки та шляхи подальшого вдосконалення	01.06.2025

Завдання видано _____ _____Молодець Б.В._____
_____ (підпис керівника) (ініціали та прізвище)

Дата видачі __25.02.2025____

Дата подання до екзаменаційної комісії_____

Прийнято до виконання _____ _____Савенко Д.О._____
(підпис здобувача вищої освіти) (ініціали та прізвище)

РЕФЕРАТ

Пояснювальна записка: 75 с., 49 рис., 5 табл., 1 додатки, 30 джерел.

ІНФОРМАЦІЙНА СИСТЕМА, МОНІТОРИНГ ЯКОСТІ ПОВІТРЯ, ВЕБ-ДОДАТОК, STREAMLIT, FLASK, SQLITE, REST API, СИНТЕТИЧНІ ДАНІ, СПОВІЩЕННЯ.

Об’єкт кваліфікаційної роботи: інформаційна система моніторингу та оповіщення про екологічну небезпеку на основі даних про якість повітря.

Предмет кваліфікаційної роботи: методи та технології розробки веб-додатків для моніторингу якості повітря з використанням Python, Streamlit, Flask та SQLite.

Мета роботи: створення інформаційної системи для оперативного моніторингу якості повітря, аналізу екологічних даних і своєчасного сповіщення користувачів про перевищення критичних рівнів забруднення.

У вступі обґрунтовано актуальність проблеми забруднення повітря в Україні та світі, проаналізовано сучасні платформи моніторингу, такі як SaveEcoBot, OpenAQ та Eco-City, визначено мету й завдання проєкту. Розроблена система спрямована на підвищення екологічної обізнаності громадян, забезпечення швидкого реагування на небезпечні ситуації та підтримку громадського контролю за станом довкілля.

У першому розділі розглянуто сучасний стан розвитку систем оповіщення про екологічні загрози, проаналізовано їх функціональні можливості та технологічні особливості. Визначено вимоги до системи, включаючи збір даних, їх візуалізацію, сповіщення та аналіз. Описано вибір технологій (Python, Flask, Streamlit, SQLite, Pandas, Plotly) та структуру даних, що включають параметри PM2.5, PM10, AQI тощо, отримані з відкритих джерел і доповнені синтетичними.

У другому розділі представлено проєктні рішення, зокрема клієнт-серверну архітектуру системи. Описано серверний модуль (Flask) для обробки даних і API, клієнтський модуль (Streamlit) для інтерактивної візуалізації, а

також дизайн інтерфейсу з вкладками «Карта моніторингу», «Аналіз даних» і «Статистика». Детально розглянуто реалізацію модулів, інтеграцію з API SaveEcoBot, генерацію синтетичних даних і механізм сповіщень.

У третьому розділі наведено результати тестування, які підтвердили коректність відображення даних, точність фільтрації, своєчасність сповіщень і стабільність системи при обробці великих обсягів даних. Виявлено незначні затримки при високому навантаженні, що можуть бути усунуті оптимізацією кешування та валідацією даних.

У висновках узагальнено результати розробки, підкреслено відповідність системи поставленим вимогам і запропоновано напрями вдосконалення, зокрема інтеграцію з мобільними додатками та прогнозними моделями.

Практичне значення кваліфікаційної роботи полягає у створенні доступного інструменту для моніторингу якості повітря, який може використовуватися громадянами, громадськими організаціями та органами влади для оперативного виявлення екологічних загроз і прийняття рішень щодо зменшення їх впливу. Розроблене програмне забезпечення може бути запроваджено в системах громадського екологічного моніторингу, а його модульна структура дозволяє масштабування та адаптацію до нових джерел даних.

ABSTRACT

Explanatory note: 75 pages, 49 figures, 5 tables, 1 appendices, 30 sources.

INFORMATION SYSTEM, AIR QUALITY MONITORING, WEB APPLICATION, STREAMLIT, FLASK, SQLITE, REST API, SYNTHETIC DATA, NOTIFICATIONS.

Object of the qualification work: an information system for monitoring and alerting about environmental hazards based on air quality data.

Subject of the qualification work: methods and technologies for developing web applications for air quality monitoring using Python, Streamlit, Flask, and SQLite.

Purpose of the work: to create an information system for real-time air quality monitoring, analysis of environmental data, and timely notification of users about exceeding critical pollution levels.

The introduction substantiates the relevance of the air pollution problem in Ukraine and globally, analyzes modern monitoring platforms such as SaveEcoBot, OpenAQ, and Eco-City, and defines the project's purpose and objectives. The developed system aims to enhance public environmental awareness, ensure rapid response to hazardous situations, and support community oversight of environmental conditions.

The first section examines the current state of development of environmental hazard alert systems, analyzing their functional capabilities and technological features. System requirements, including data collection, visualization, notifications, and analysis, are defined. The selection of technologies (Python, Flask, Streamlit, SQLite, Pandas, Plotly) and the data structure, incorporating parameters such as PM2.5, PM10, AQI, etc., sourced from open platforms and supplemented with synthetic data, are described.

The second section presents project solutions, including the client-server architecture of the system. It describes the server module (Flask) for data processing and API, the client module (Streamlit) for interactive visualization, and the interface

design with tabs for “Monitoring Map,” “Data Analysis,” and “Statistics.” The implementation of modules, integration with the SaveEcoBot API, synthetic data generation, and the notification mechanism are discussed in detail.

The third section provides testing results, confirming the accuracy of data display, precision of filtering, timeliness of notifications, and system stability when processing large data volumes. Minor delays under high load were identified, which can be addressed through caching optimization and data validation.

The conclusions summarize the development outcomes, highlight the system’s compliance with the set requirements, and propose directions for improvement, such as integration with mobile applications and predictive models.

Practical significance of the qualification work lies in creating an accessible tool for air quality monitoring, which can be used by citizens, public organizations, and authorities for timely detection of environmental threats and decision-making to mitigate their impact. The developed software can be implemented in public environmental monitoring systems, and its modular structure enables scalability and adaptation to new data sources.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API – Інтерфейс програмування додатків (Application Programming Interface)

AQI – Індекс якості повітря (Air Quality Index)

PM2.5 – Тверді частинки розміром до 2,5 мкм

PM10 – Тверді частинки розміром до 10 мкм

NO2 – Діоксид азоту

SO2 – Діоксид сірки

O3 – Озон

CO – Чадний газ

IoT – Інтернет речей (Internet of Things)

REST – Представницький стан передачі (Representational State Transfer)

JSON – Формат обміну даними (JavaScript Object Notation)

SQL – Мова структурованих запитів (Structured Query Language)

ORM – Об’єктно-реляційне відображення (Object-Relational Mapping)

UI – Користувацький інтерфейс (User Interface)

CSV – Формат файлу з роздільниками комами (Comma-Separated Values)

GeoJSON – Формат даних для географічних об’єктів

HTTP – Протокол передачі гіпертексту (Hypertext Transfer Protocol)

UUID – Унікальний ідентифікатор (Universally Unique Identifier)

ВСТУП

Розробка інформаційної системи оповіщення про екологічну небезпеку є актуальним напрямом у сфері інформаційних технологій та екологічного моніторингу. Система спрямована на забезпечення оперативного інформування населення про стан якості повітря, зокрема про перевищення допустимих рівнів забруднення. Реалізована система на основі веб-додатка (client.py) та серверної частини (server.py) використовує дані з відкритих джерел, таких як SaveEcoBot, для збору, обробки та візуалізації інформації про екологічні параметри. Вона включає функціонал відображення даних на карті, аналізу часових рядів, статистичних показників, а також автоматичних сповіщень про небезпечні рівні забруднення.

Актуальність проєкту. Погіршення екологічної ситуації, зокрема забруднення атмосферного повітря, є глобальною проблемою, яка безпосередньо впливає на здоров’я населення та якість життя. В Україні, де промислові викиди, транспортне навантаження та сезонні фактори (наприклад, горіння сухої рослинності) сприяють забрудненню повітря, створення систем моніторингу та оповіщення є особливо актуальним. Відкриті дані про якість повітря, які надають платформи на кшталт SaveEcoBot чи OpenAQ, створюють можливості для розробки доступних інструментів громадського контролю. Актуальність проєкту також зумовлена зростанням попиту на цифрові рішення, які забезпечують оперативне інформування та дозволяють громадянам приймати обґрунтовані рішення для захисту власного здоров’я.

Метою проєкту є розробка інформаційної системи, яка забезпечує моніторинг якості повітря в реальному часі, аналіз екологічних даних та своєчасне оповіщення користувачів про перевищення критичних рівнів забруднення. Система покликана підвищити рівень обізнаності населення про екологічні ризики, сприяти швидкому реагуванню на небезпечні ситуації та підтримувати громадський контроль за станом довкілля.

Для досягнення поставленої мети було визначено такі **завдання**:

– розробити клієнтську частину системи на основі Streamlit для інтерактивного відображення даних про якість повітря, включаючи карти, графіки та таблиці;

– створити серверну частину з використанням Flask та SQLite для збору, зберігання та обробки даних із зовнішніх джерел;

– реалізувати механізм генерації синтетичних даних на основі реальних для демонстрації роботи системи;

– забезпечити функціонал автоматичних сповіщень про перевищення порогових значень забруднюючих речовин;

– інтегрувати можливості фільтрації, експорту даних та статистичного аналізу для поглибленого вивчення екологічної ситуації;

– розробити зручний та адаптивний інтерфейс для користувачів із підтримкою темної теми та довідковими матеріалами;

Об’єктом роботи є інформаційна система моніторингу та оповіщення про екологічну небезпеку, яка базується на обробці даних про якість повітря. Система охоплює як програмні компоненти (к клієнтський і серверний модулі), так і процеси збору, обробки, аналізу та візуалізації екологічних даних, отриманих із відкритих джерел.

Предметом роботи є методи та технології розробки інформаційної системи, зокрема використання Python-бібліотек (Streamlit, Flask, Pandas, Plotly), бази даних SQLite, а також алгоритми обробки даних і генерації сповіщень. Особлива увага приділяється інтеграції даних із зовнішніх API, створенню інтерактивного веб-інтерфейсу та забезпеченню надійності й масштабованості системи.

Практична цінність проєкту полягає у створенні доступного інструменту для моніторингу якості повітря, який може бути використаний як окремими громадянами, так і громадськими організаціями чи місцевими органами влади. Система дозволяє оперативно виявляти екологічні загрози, інформувати населення та сприяти прийняттю рішень щодо зменшення впливу

забруднення. Її модульна структура та використання відкритих даних забезпечують можливість подальшого масштабування, наприклад, додавання нових джерел даних чи розширення функціоналу (прогнозування, інтеграція з мобільними додатками). Крім того, проєкт сприяє підвищенню екологічної свідомості та залученню громадськості до вирішення екологічних проблем.

РОЗДІЛ 1. АНАЛІЗ СТАНУ ОБЛАСТІ РІШЕННЯ ЗАВДАННЯ

1.1. Огляд існуючих систем оповіщення про екологічні загрози

Сучасний розвиток інформаційних технологій сприяв появі різноманітних систем оповіщення про екологічні загрози, які спрямовані на моніторинг стану навколишнього середовища, аналіз даних про забруднення та оперативне інформування населення. Такі системи відіграють ключову роль у забезпеченні безпеки громадян, своєчасному реагуванні на екологічні ризики та підтримці сталого розвитку. У контексті аналізу реалізованого проєкту інформаційної системи, що використовує дані з відкритих джерел, таких як SaveEcoBot, OpenAQ та Eco-City, розглянемо сучасний стан розвитку систем оповіщення про екологічні загрози, їх основні функціональні можливості, технології та виклики.

Одним із ключових напрямів розвитку систем оповіщення є моніторинг якості повітря. Такі платформи, як OpenAQ, надають відкритий доступ до даних про концентрацію забруднюючих речовин, таких як PM2.5, PM10, NO2, SO2, O3 та CO, зібраних із тисяч датчиків по всьому світу [1]. OpenAQ агрегує дані з державних, дослідницьких та громадських джерел, що дозволяє створювати глобальну картину стану повітря. Система підтримує API для інтеграції з іншими програмами, що є важливим для розробки локальних рішень, подібних до показаної у коді системи. Проте OpenAQ не має вбудованих функцій оповіщення, що обмежує її використання для оперативного інформування населення.

В Україні значний внесок у моніторинг якості повітря робить платформа SaveEcoBot, яка поєднує дані з власних датчиків, державних станцій та інших джерел [2]. SaveEcoBot пропонує користувачам веб-інтерфейс та чат-бот для отримання інформації про якість повітря в реальному часі, а також можливість налаштування сповіщень про перевищення критичних рівнів забруднення. Функціонал SaveEcoBot включає інтерактивні карти, аналітичні звіти та

підтримку кількох мов, що робить її доступною для широкої аудиторії. У контексті створеної системи SaveEcoBot виступає одним із джерел даних, а її API використовується для отримання актуальної інформації про стан повітря.

Ще одним прикладом є система Eco-City, яка фокусується на автоматизованому моніторингу якості повітря в Україні за допомогою мережі власних станцій [3]. Eco-City надає дані через веб-платформу та мобільні додатки, дозволяючи користувачам відстежувати показники в реальному часі. Система підтримує функцію сповіщень, яка попереджає про небезпечні рівні забруднення, що є схожим до функціоналу створеної інформаційної системи. Однак Eco-City обмежена географічно, охоплюючи переважно великі міста України, що може бути викликом для масштабування на національний рівень.

На міжнародному рівні варто відзначити платформу AirNow, розроблену Агентством з охорони навколишнього середовища США (EPA) [4]. AirNow надає детальну інформацію про якість повітря, включаючи індекс AQI, та підтримує персоналізовані сповіщення для користувачів. Система використовує розвинену мережу датчиків і прогнозні моделі для оцінки майбутніх ризиків. Хоча AirNow є високоефективною, її функціонал обмежений територією США, що робить її менш релевантною для України.

Серед європейських рішень виділяється платформа European Air Quality Index (EAQI), яка розроблена Європейським агентством з навколишнього середовища [5]. EAQI інтегрує дані з країн ЄС, надаючи уніфіковану оцінку якості повітря та інтерактивні sername карти. Система дозволяє користувачам отримувати сповіщення через веб-інтерфейс, однак не має розвинених мобільних додатків, що обмежує її доступність для пересічних громадян.

Технологічною основою більшості сучасних систем є використання IoT (Інтернет речей) для збору даних із датчиків, хмарних технологій для обробки великих обсягів інформації та API для інтеграції з іншими платформами [6]. Наприклад, у створеній системі використовується Flask для серверної частини та Streamlit для створення інтерактивного інтерфейсу, що відповідає сучасним тенденціям у розробці веб-додатків. Крім того, важливим показником є

застосування аналітичних інструментів, таких як Plotly, для візуалізації даних, що полегшує сприйняття інформації користувачами.

Незважаючи на значний прогрес, існуючі системи стикаються з низкою викликів. По-перше, це нерівномірне покриття датчиками, особливо в сільських регіонах України, що обмежує точність моніторингу [7]. По-друге, відсутність єдиних стандартів для збору та обробки даних ускладнює їх інтеграцію з різних джерел. По-третє, багато систем не надають достатньо персоналізованих функцій оповіщення, що знижує їх ефективність для окремих користувачів. Нарешті, питання кібербезпеки залишається актуальним, оскільки системи, які обробляють великі обсяги даних, можуть бути вразливими до атак [8].

Отже, сучасні системи оповіщення про екологічні загрози демонструють значний прогрес у напрямку моніторингу якості повітря та інформування населення. Проте виклики, пов’язані з покриттям, стандартизацією даних, персоналізацією та безпекою, залишаються актуальними. Спроєктована інформаційна система, що базується на інтеграції даних із SaveEcoBot, OpenAQ та Eco-City, відповідає сучасним вимогам, пропонуючи інтерактивний інтерфейс, аналітичні інструменти та персоналізовані сповіщення, що робить її перспективним рішенням для локального використання.

1.2. Визначення вимог до інформаційної системи

Розробка інформаційної системи оповіщення про екологічну небезпеку вимагає чіткого визначення вимог, які забезпечують її функціональність, ефективність та відповідність сучасним викликам у сфері моніторингу якості довкілля. На основі аналізу кодів проєкту та сучасних тенденцій у галузі екологічного моніторингу сформульовано ключові вимоги до системи. Ці вимоги відображають як технічні, так і функціональні показники, що відповідають цілям проєкту та забезпечують його практичну цінність.

Функціональні вимоги показані в таблиці 1.1

Таблиця 1.1 – Функціональні вимоги

№	Вимоги	Опис вимог
1	Збір та обробка даних про якість повітря	Система повинна забезпечувати автоматичний збір даних із різних джерел, таких як SaveEcoBot, OpenAQ та Eco-City, а також підтримувати можливість інтеграції синтетичних даних для моделювання [2]. Розроблений серверний код реалізує функцію fetch_saveecobot_synthetic(), яка отримує дані з API SaveEcoBot та генерує синтетичні значення для забезпечення безперервності інформації. Це дозволяє системі працювати навіть за відсутності реальних даних у певний момент часу
2	Візуалізація даних у реальному часі	Система має надавати користувачам зручний інтерфейс для відображення даних про якість повітря, включаючи картографічні представлення, графіки та таблиці. У клієнтському коді використано бібліотеку Streamlit для створення інтерактивних вкладок ("Карта моніторингу", "Аналіз даних", "Статистика"), що відповідає цій вимозі [9]. Наприклад, функція create_air_quality_chart() генерує графіки з трендами та пороговими значеннями для параметрів, таких як PM2.5, PM10 та AQI
3	Система сповіщень	Користувачі повинні отримувати автоматичні повідомлення про перевищення встановлених порогових значень забруднювачів. Серверна функція check_thresholds_and_send_notifications() періодично перевіряє дані та надсилає повідомлення через API, а клієнтська частина відображає їх у вигляді тостів (st.toast) [10]. Це забезпечує оперативне інформування про екологічні ризики
4	Фільтрація та аналіз даних	Система повинна дозволяти користувачам фільтрувати дані за параметрами (наприклад, PM2.5, NO2), часовими інтервалами, джерелами та значеннями, а також проводити статистичний аналіз. Клієнтський код реалізує форму фільтрації (data_filter_form) та різноманітні типи аналізу, такі як часові ряди, теплові карти та добовий аналіз, що відповідає сучасним стандартам обробки екологічних даних [1]

Нефункціональні вимоги показані в таблиці 1.2.

Таблиця 1.2 – Нефункціональні вимоги

№	Вимоги	Опис вимог
1	Доступність та інтуїтивність інтерфейсу	Інтерфейс системи має бути зрозумілим для широкого кола користувачів, включаючи осіб без технічної підготовки. Використання Streamlit забезпечує адаптивний та інтерактивний вебінтерфейс із підтримкою темної та світлої тем, що підвищує зручність використання [11]
2	Масштабованість та продуктивність	Система повинна обробляти великі обсяги даних без значних затримок. Серверна частина використовує SQLite як легку базу даних, а клієнтський код застосовує кешування (@st.cache_data) для оптимізації запитів до API, що забезпечує швидкодію навіть при великій кількості точок моніторингу [12]
3	Надійність та стабільність	Система має працювати безперебійно, автоматично оновлюючи дані та очищаючи застарілі записи. Фонові задачі (update_synthetic_data_task, cleanup_old_data_task) у серверному коді забезпечують регулярне оновлення синтетичних даних та видалення записів, старших за 30 днів, що сприяє стабільності системи [13]
4	Безпека даних	Хоча система використовує унікальний ідентифікатор пристрою (get_device_id()), вона повинна гарантувати захист персональних даних користувачів, таких як підписки на сповіщення. Серверна частина реалізує базову перевірку даних у запитах API, але для промислового використання потрібні додаткові заходи, наприклад, шифрування [14]

Система має бути адаптованою до українських реалій, включаючи підтримку української мови в інтерфейсі та врахування географічних особливостей (наприклад, перевірка координат України у функції is_ukrainian_coordinates()). Крім того, вона повинна надавати довідкову інформацію про забруднювачі (PM2.5, NO2 тощо), що реалізовано у бічній панелі клієнтського інтерфейсу.

Розроблена система, представлена у лістингах, повністю відповідає сформульованим вимогам, забезпечуючи комплексний підхід до моніторингу екологічної безпеки. Вона поєднує сучасні технології обробки даних, інтерактивну візуалізацію та оперативне інформування, що робить її цінним інструментом для громадського та професійного використання.

1.3. Вибір технологій для реалізації

Розробка інформаційної системи оповіщення про екологічну небезпеку вимагає ретельного вибору технологій, які забезпечують ефективність, масштабованість, надійність та зручність використання. На основі аналізу функціональних вимог до системи, представленої у лістингу проєкту, було обрано технологічний стек, що відповідає сучасним стандартам розробки веб-додатків та обробки даних про якість повітря. Нижче детально розглянуто вибір технологій для реалізації системи, враховуючи їх відповідність поставленим задачам.

Для реалізації серверної частини системи обрано мову програмування Python [15], яка вирізняється своєю універсальністю, широкою екосистемою бібліотек та простотою синтаксису, що сприяє швидкій розробці та підтримці коду. Python забезпечує ефективну роботу з обробкою даних, що є ключовим для аналізу екологічних показників. Як серверний фреймворк використано Flask [16], який є легковаговим і дозволяє створювати RESTful API з мінімальними затратами ресурсів. Flask забезпечує гнучкість у налаштуванні маршрутів та обробці запитів, що відповідає потребам системи для надання даних клієнтській частині та обробки підписок на сповіщення. Для клієнтської частини використано бібліотеку Streamlit [9], яка дозволяє створювати інтерактивні веб-інтерфейси з мінімальними зусиллями, забезпечуючи відображення графіків, таблиць та карт для аналізу екологічних даних.

Для зберігання даних про точки моніторингу, вимірювання та підписки користувачів обрано SQLite [12] як легковагову реляційну базу даних. SQLite

не потребує окремого серверного процесу, що спрощує розгортання системи, особливо на етапі тестування та прототипування. Для взаємодії з базою даних використано ORM-бібліотеку SQLAlchemy [17], яка забезпечує абстракцію над SQL-запитами, спрощує роботу з моделями даних та підтримує міграцію до інших СУБД у майбутньому, якщо виникне потреба в масштабуванні. Використання SQLAlchemy дозволяє ефективно обробляти запити до даних, такі як фільтрація вимірювань за параметрами чи отримання статистики.

Для обробки даних використано бібліотеку Pandas [18], яка забезпечує швидку та зручну роботу з табличними даними, дозволяючи виконувати фільтрацію, групування та агрегацію вимірювань. Візуалізація даних реалізована за допомогою бібліотек Plotly [19] та Plotly Express, які дозволяють створювати інтерактивні графіки, теплові карти та діаграми для представлення часових рядів, розподілів значень та порівнянь між станціями моніторингу. Plotly підтримує адаптивне відображення графіків у веб-інтерфейсі Streamlit, що підвищує зручність для користувачів. Для відображення географічних даних на карті використано вбудовану функцію Streamlit st.map, яка інтегрується з бібліотеками для роботи з GeoJSON [20], забезпечуючи відображення точок моніторингу.

Для отримання даних з зовнішніх джерел, таких як SaveEcoBot, використовується бібліотека Requests [21], яка забезпечує зручну взаємодію з HTTP API. Це дозволяє системі отримувати актуальні дані про якість повітря та інтегрувати їх у власну базу даних. Для генерації синтетичних даних, що імітують реальні вимірювання, використано бібліотеку NumPy [22], яка забезпечує ефективні обчислення для створення реалістичних коливань значень параметрів.

Для реалізації фонових задач, таких як регулярне оновлення даних, перевірка порогових значень та очищення старих записів, використано модуль Threading з Python [23]. Це дозволяє виконувати асинхронні операції без блокування основного потоку сервера. Для моніторингу роботи системи та відстеження помилок реалізовано логування за допомогою модуля Logging,

що забезпечує збереження інформації про дії сервера у файлі server.log та виведення її у консоль.

Система підтримує темну та світлу теми інтерфейсу, що реалізовано через налаштування Plotly та Streamlit, підвищуючи комфорт користувачів. Для забезпечення унікальності пристроїв використано генерацію UUID, що дозволяє ідентифікувати користувачів для сповіщень. Використання кешування (@st.cache_data та @st.cache_resource) у Streamlit оптимізує продуктивність, зменшуючи час обробки запитів до API.

Отже, обраний технологічний стек, що включає Python, Flask, Streamlit, SQLite, SQLAlchemy, Pandas, Plotly, Requests, NumPy та Threading, забезпечує повноцінну реалізацію системи оповіщення про екологічну небезпеку. Ці технології відповідають вимогам до обробки великих обсягів даних, створення інтерактивного інтерфейсу та інтеграції з зовнішніми джерелами, що підтверджується функціональністю кодів проєкту.

1.4. Опис даних

Розроблена інформаційна система оповіщення про екологічну небезпеку, представлена у коді проєкту, базується на обробці, аналізі та представленні даних про якість повітря, отриманих із різних джерел. Дані, які використовуються в системі, мають чітку структуру, відповідають сучасним вимогам до моніторингу екологічних параметрів і забезпечують реалізацію функціональних можливостей системи, таких як інтерактивна візуалізація, аналіз трендів і автоматичне сповіщення про перевищення порогових значень. У цьому підрозділі детально описано характеристики даних, їх джерела, структуру, формати та особливості обробки, що відповідають специфіці кодів проєкту.

Система інтегрує дані з відкритих платформ моніторингу якості повітря, зокрема SaveEcoBot, OpenAQ та Eco-City, які є визнаними джерелами екологічної інформації в Україні та світі [1, 2, 3]. SaveEcoBot забезпечує

доступ до даних із мережі датчиків, розташованих в українських містах, таких як Київ, Харків, Дніпро, Одеса, Львів і Запоріжжя [24]. OpenAQ агрегує глобальні дані про якість повітря, включаючи вимірювання з України [25]. Eco-City, як локальна платформа, надає деталізовані дані з автоматизованих станцій моніторингу [26]. Крім того, для демонстраційних цілей система використовує синтетичні дані, згенеровані на основі реальних значень із урахуванням добових і тижневих коливань, що дозволяє імітувати реальні сценарії забруднення повітря [27].

Дані, оброблювані системою, включають вимірювання ключових параметрів якості повітря: PM2.5, PM10, NO2, SO2, O3, CO та AQI (індекс якості повітря) [28]. Кожен параметр супроводжується значенням, одиницею вимірювання (наприклад, μg/m³ для PM2.5, PM10, NO2, SO2, O3, CO), часовою міткою у форматі ISO8601 та географічними координатами (широта і довгота) точки моніторингу [29]. Наприклад, PM2.5 і PM10 відображають концентрацію твердих частинок розміром до 2,5 та 10 мкм відповідно, тоді як AQI є комплексним показником, що враховує вплив кількох забруднювачів [30]. Система також зберігає метадані про джерело (SaveEcoBot, OpenAQ, Eco-City) та назву станції моніторингу, що забезпечує їх ідентифікацію та фільтрацію [19].

У системі використовується реляційна база даних SQLite, що включає три основні таблиці: monitoring_points (інформація про точки моніторингу: ідентифікатор, назва, координати, джерело), monitoring_data (вимірювання: ідентифікатор точки, параметр, значення, одиниця вимірювання, час вимірювання) та user_subscriptions (підписки користувачів: ідентифікатор пристрою, точка моніторингу, параметр, порогове значення) [12] (рисунок 1.1). Така структура забезпечує ефективне зберігання, пошук і фільтрацію даних, а також підтримує функціонал сповіщень. Наприклад, дані з таблиці monitoring_data дозволяють будувати часові ряди та теплові карти, а таблиця user_subscriptions використовується для перевірки порогових значень і відправки повідом huddled [9].

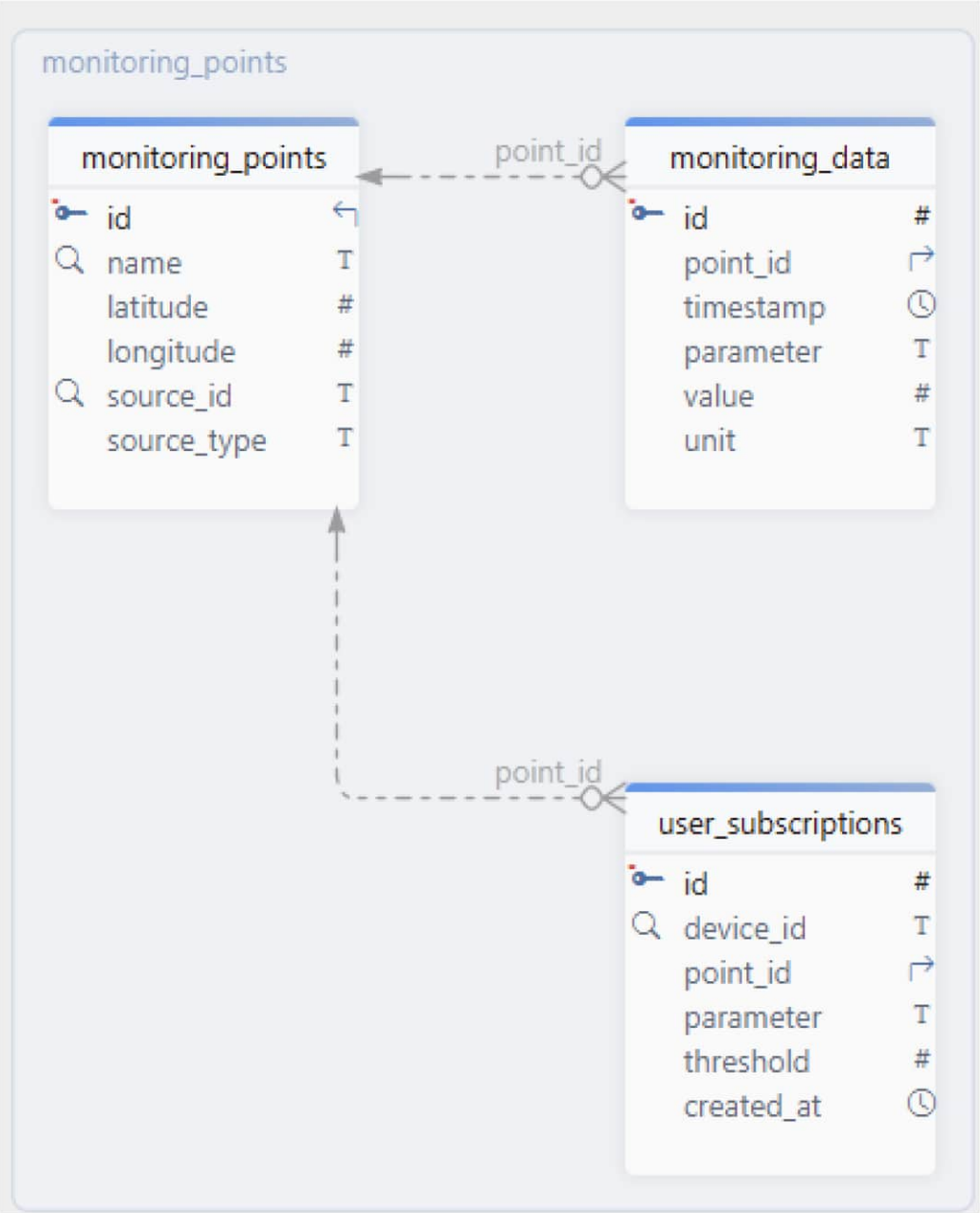


Рисунок 1.1 – Структура бази даних

Дані надходять у форматі JSON із зовнішніх API (SaveEcoBot, OpenAQ, Eco-City) і після обробки зберігаються в базі даних. Для забезпечення коректності система застосовує безпечне перетворення часових міток у формат ISO8601, обробляє пропущені значення та перевіряє координати для відповідності території України [29]. Синтетичні дані генеруються з урахуванням статистичних моделей, що включають добові (синусоїдальні) та тижневі коливання, а також випадкові відхилення, що імітують реальну мінливість параметрів [27]. Оброблені дані представляються у вигляді

таблиць, графіків (часові ряди, гістограми, теплові карти) та інтерактивних карт за допомогою бібліотек Streamlit і Plotly [19].

Система підтримує фільтрацію даних за параметрами (наприклад, PM2.5, AQI), діапазоном значень, датами, джерелами та кількістю записів, що дозволяє користувачам аналізувати конкретні показники якості повітря. Дані експортуються у формати CSV та Excel для подальшого використання [12]. Крім того, система реалізує механізм автоматичного оновлення даних кожні 15 хвилин і очищення застарілих записів (старших за 30 днів), що забезпечує актуальність інформації [9]. Для сповіщень використовується механізм перевірки порогових значень кожні 5 хвилин, а повідомлення зберігаються у файлі notifications.json для імітації пуш-повідомлень [30].

Таким чином, дані, використані в системі, є комплексними, структурованими та адаптованими до потреб моніторингу екологічної безпеки. Їх джерела, формати та обробка відповідають сучасним стандартам екологічного аналізу, забезпечуючи точність, актуальність і зручність використання для кінцевих користувачів.

РОЗДІЛ 2. ПРОЄКТНІ РІШЕННЯ

2.1. Архітектура інформаційної системи

Розробка інформаційної системи оповіщення про екологічну небезпеку базується на створенні гнучкої, масштабованої та користувацько-орієнтованої архітектури, яка забезпечує ефективний моніторинг якості повітря, аналіз даних та своєчасне інформування користувачів про потенційні екологічні загрози. Архітектура системи побудована за принципами клієнт-серверної моделі з чітким розподілом функціональних компонентів, що дозволяє оптимізувати обробку даних, забезпечити модульність та полегшити подальше масштабування. У цьому розділі детально розглянуто архітектурні рішення, включаючи структуру системи, взаємодію компонентів та їх функціональне призначення.

Інформаційна система складається з двох основних частин: серверної (бекенд) та клієнтської (фронтенд). Серверна частина, реалізована за допомогою фреймворку Flask, відповідає за обробку даних, їх зберігання, генерацію синтетичних даних та надання API для взаємодії з клієнтом. Клієнтська частина, побудована з використанням Streamlit, забезпечує інтерактивний інтерфейс для відображення даних, аналізу та налаштування сповіщень. Для зберігання даних використовується реляційна база даних SQLite, яка забезпечує легкість розгортання та достатню продуктивність для обробки даних моніторингу. Система також інтегрується з зовнішніми джерелами даних, зокрема API SaveEcoBot, для отримання інформації про якість повітря.

Архітектура системи побудована за принципом багатошаровості (таблиці 2.1).

Таблиця 2.1 – Основні шари архітектури

№	Шар	Опис шару
1	Шар збору даних	відповідає за отримання даних з зовнішніх джерел (SaveEcoBot) та генерацію синтетичних даних для моделювання реальних сценаріїв
2	Шар обробки та зберігання даних	забезпечує збереження даних у базі SQLite, їх фільтрацію, агрегацію та підготовку для аналізу
3	Шар API	надає RESTful-інтерфейс для взаємодії клієнтської частини з сервером
4	Шар представлення	реалізує візуалізацію даних, аналітичні графіки та інтерфейс для налаштування сповіщень
5	Шар сповіщень	відповідає за перевірку порогових значень параметрів та формування повідомлень для користувачів

Система використовує клієнт-серверну модель взаємодії, де клієнт (Streamlit-додаток) надсилає запити до сервера через HTTP-запити, а сервер повертає відповідні дані у форматі JSON. Основні компоненти системи та їх взаємодія описані в таблиці 2.2.

Таблиця 2.2 – Основні компоненти системи та їх взаємодія

№	Компоненти	Опис
1	Клієнтський додаток (Streamlit)	відображає карту моніторингу, графіки, таблиці та дозволяє користувачу налаштовувати сповіщення. Запити до сервера формуються через бібліотеку requests для отримання даних про точки моніторингу, статистику та історичні дані
2	Сервер (Flask)	обробляє запити клієнта, взаємодіє з базою даних через ORM SQLAlchemy та забезпечує фонові задачі, такі як оновлення синтетичних даних, перевірка сповіщень та очищення застарілих записів
3	База даних (SQLite)	зберігає інформацію про точки моніторингу, дані вимірювань та підписки користувачів. Використання SQLite дозволяє спростити розгортання системи, хоча при масштабуванні можлива заміна на більш потужну СУБД, наприклад, PostgreSQL
4	Зовнішні джерела даних	API SaveEcoBot використовується для отримання реальних даних, які доповнюються синтетичними для забезпечення безперервності моніторингу

Для забезпечення асинхронної обробки фонових задач, таких як оновлення даних та перевірка сповіщень, сервер використовує потоки (threading), що дозволяє виконувати ці операції паралельно з обробкою клієнтських запитів. Система також включає механізми кешування (за допомогою декораторів @st.cache_data та @st.cache_resource у Streamlit), що зменшує кількість запитів до сервера та підвищує швидкодію інтерфейсу.

Діаграма компонентів (рисунок 2.1) відображає модульну структуру системи та взаємодію між основними її частинами. На верхньому рівні система поділяється на клієнтську та серверну частини, які взаємодіють через REST API. Клієнтська частина включає модулі для візуалізації даних (карта, графіки, таблиці), управління сповіщеннями та фільтрації даних. Серверна частина складається з модулів обробки API-запитів, управління базою даних, генерації синтетичних даних та фонових задач. База даних SQLite виступає центральним сховищем, яке зв’язує всі серверні компоненти. Зовнішнє API SaveEcoBot підключене до модуля збору даних, забезпечуючи джерело реальних вимірювань.

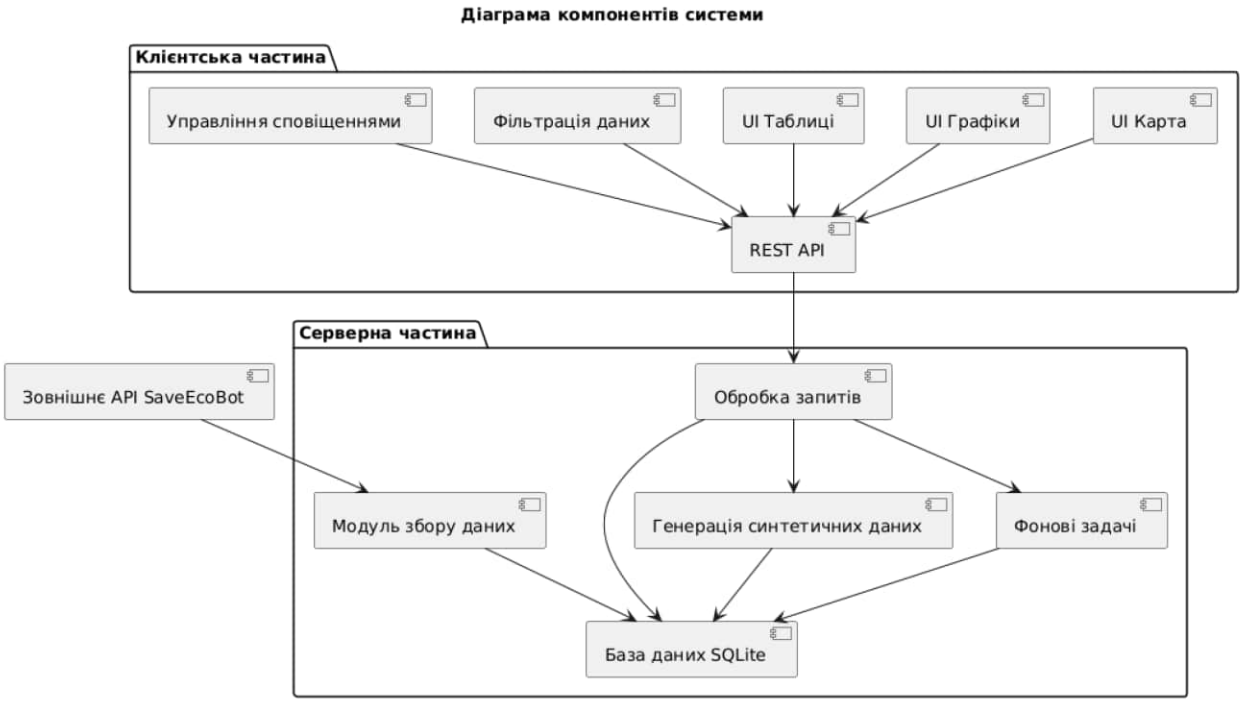


Рисунок 2.1 – Діаграма компонентів

Діаграма класів (рисунок 2.2) відображає основні моделі даних, реалізовані в серверній частині системи за допомогою SQLAlchemy. Основні класи включають:

– MonitoringPoint представляє точку моніторингу з атрибутами id, name, latitude, longitude, source_id та source_type. Має зв’язок один-до-багатьох з класом MonitoringData.

– MonitoringData відповідає за зберігання даних вимірювань з атрибутами id, point_id (зовнішній ключ до MonitoringPoint), timestamp, parameter, value та unit.

– UserSubscription описує підписку користувача на сповіщення з атрибутами id, device_id, point_id, parameter, threshold та created_at.

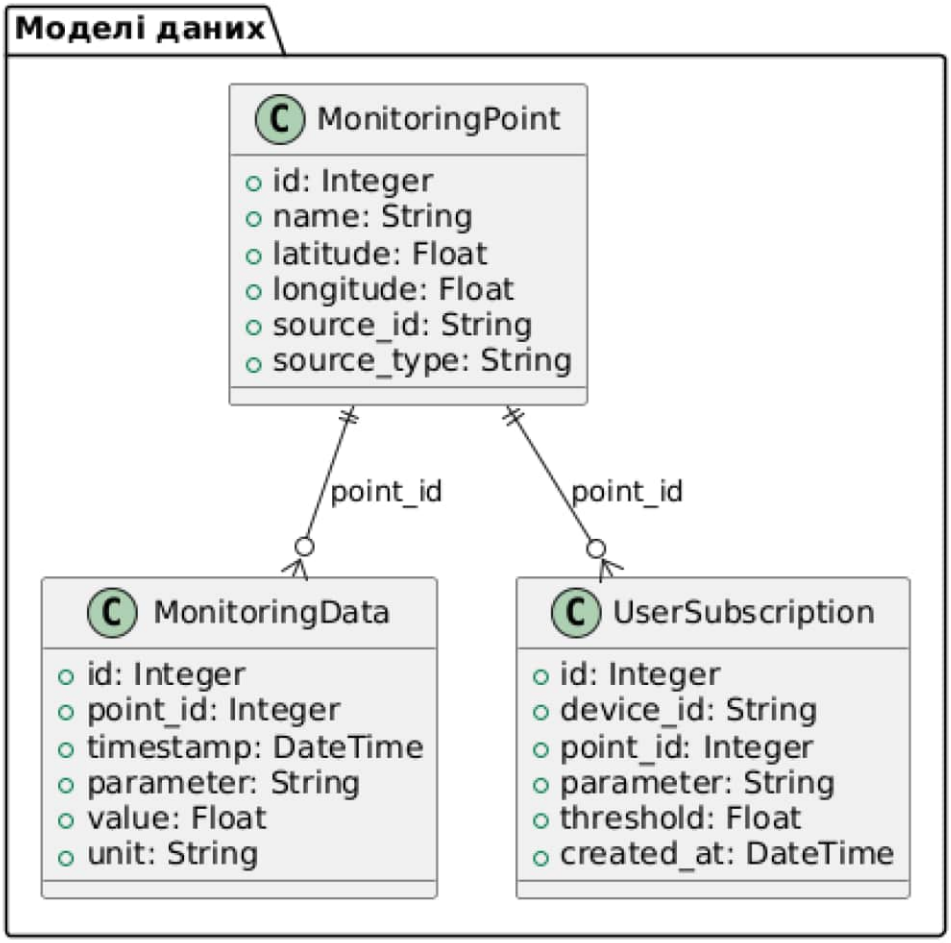


Рисунок 2.2 – Діаграма класів

Ці класи формують ядро моделі даних, забезпечуючи зв’язок між точками моніторингу, вимірюваннями та підписками. Клієнтська частина не має чітко визначених класів, оскільки Streamlit використовує декларативний підхід до побудови інтерфейсу, але логічно можна виділити модулі для обробки даних, візуалізації та управління сесіями.

Для реалізації системи використано сучасний технологічний стек, який забезпечує ефективність, простоту розгортання та підтримку (таблиця 2.3).

Таблиця 2.3 – Технологічний стек

№	Технологія	Опис
1	Python	основна мова програмування для клієнтської та серверної частин
2	Flask	легкий веб-фреймворк для створення REST API
3	Streamlit	інструмент для швидкої розробки інтерактивних веб-інтерфейсів
4	SQLAlchemy	ORM для зручної роботи з базою даних
5	SQLite	легка реляційна база даних для зберігання даних
6	Plotly	бібліотека для створення інтерактивних графіків і візуалізацій
7	Pandas	інструмент для обробки та аналізу даних

Архітектура системи передбачає можливість масштабування шляхом заміни SQLite на більш продуктивну СУБД, наприклад, PostgreSQL, та використання контейнеризації (Docker) для розгортання. Модульна структура дозволяє легко додавати нові джерела даних, розширювати набір параметрів моніторингу та інтегрувати додаткові функції, такі як прогнозування якості повітря або інтеграція з мобільними додатками.

Архітектура інформаційної системи оповіщення про екологічну небезпеку поєднує клієнт-серверну модель, багатошарову організацію та сучасні технології для забезпечення ефективного моніторингу якості повітря. Чіткий розподіл компонентів, використання REST API та фонових задач забезпечують швидкодію, гнучкість і зручність для користувача. Діаграми

компонентів і класів ілюструють модульність і структурованість системи, що полегшує її підтримку та подальший розвиток.

2.2. Реалізація модулів системи

Розробка інформаційної системи оповіщення про екологічну небезпеку передбачає створення комплексного програмного продукту, який забезпечує моніторинг якості повітря, аналіз даних та своєчасне сповіщення користувачів про перевищення небезпечних рівнів забруднення. Система складається з двох основних модулів: серверного (server.py) та клієнтського (client.py), які взаємодіють через API. Нижче детально описано реалізацію основних модулів системи, їх функціонал, архітектурні рішення та використані технології.

2.2.1. Серверний модуль (server.py)

Серверний модуль є основою системи, відповідаючи за обробку даних, їх зберігання та надання клієнтській частині через RESTful API. Реалізація цього модуля базується на фреймворку Flask, який забезпечує легковагу та гнучку обробку HTTP-запитів. Основні компоненти серверного модуля включають:

1) Модель бази даних та управління даними

Для зберігання даних використано реляційну базу даних SQLite, яка інтегрована через ORM-бібліотеку SQLAlchemy. Визначено три основні моделі:

– MonitoringPoint – таблиця для зберігання інформації про точки моніторингу (назва, координати, джерело даних).

– MonitoringData – таблиця для даних вимірювань (параметр, значення, одиниця виміру, часовий штамп).

– UserSubscription – таблиця для підписок користувачів на сповіщення (ідентифікатор пристрою, точка моніторингу, параметр, порогове значення).

Ці моделі дозволяють ефективно структурувати дані та забезпечують зв'язок між точками моніторингу, вимірюваннями та підписками. Використання SQLite забезпечує портативність і простоту розгортання системи.

2) Отримання та генерація даних

Сервер інтегрується з API SaveEcoBot для отримання реальних даних про якість повітря. Функція fetch_saveecobot_synthetic обробляє відповідь API, фільтрує станції за координатами України та додає їх до бази даних. Для демонстраційних цілей реальні дані доповнюються синтетичними, які генеруються функцією generate_synthetic_data. Ця функція враховує добові та тижневі коливання, додаючи випадкові відхилення для створення реалістичних даних. Синтетичні дані генеруються як для минулих періодів (24 години назад), так і для прогнозних значень (24 години вперед), що дозволяє моделювати динаміку забруднення.

3) API-ендпоінти

Сервер надає набір API-ендпоінтів для взаємодії з клієнтом:

– /api/points – повертає список усіх точок моніторингу з їх останніми даними.

– /api/points/<point_id> – повертає детальні дані конкретної точки.

– /api/data – забезпечує доступ до всіх даних із можливістю фільтрації за параметрами, значеннями, датами та джерелами.

– /api/statistics – надає статистичні показники (кількість точок, середні значення параметрів).

– /api/subscriptions – дозволяє створювати підписки на сповіщення.

– /api/notifications – зберігає повідомлення у JSON-файл для імітації пуш-сповіщень.

– /api/regenerate_data – запускає перегенерацію даних.

– /api/cleanup – видаляє застарілі дані.

Використання RESTful API забезпечує стандартизовану та масштабовану взаємодію між сервером і клієнтом.

4) Фонові задачі

Для забезпечення актуальності даних і автоматичного сповіщення реалізовано три фонові задачі, які працюють у окремих потоках:

– Оновлення синтетичних даних (update_synthetic_data_task) кожні 15 хвилин.

– Перевірка порогових значень і відправка сповіщень (check_notifications_task) кожні 5 хвилин.

– Очищення даних, старших за 30 днів (cleanup_old_data_task), щоденно.

Використання потоків дозволяє виконувати ці задачі асинхронно, не перериваючи основний цикл обробки запитів.

5) Логування та обробка помилок

Сервер використовує бібліотеку logging для запису подій у файл server.log та виведення в консоль. Логуються запити до API, помилки обробки даних і виконання фонових задач. Це полегшує діагностику проблем і моніторинг роботи системи.

2.2.2. Клієнтський модуль (client.py)

Клієнтський модуль реалізовано з використанням фреймворку Streamlit, який забезпечує створення інтерактивного веб-інтерфейсу з мінімальними зусиллями. Модуль відповідає за візуалізацію даних, взаємодію з користувачем і відображення сповіщень. Основні компоненти включають:

1) Інтерфейс користувача

Інтерфейс організовано у вигляді трьох вкладок:

– Карта моніторингу відображає точки моніторингу на карті з кольоровим кодуванням (зелений – добра якість, оранжевий – середня, червоний – погана). Користувач може вибрати точку для перегляду детальних даних.

– Аналіз даних дозволяє фільтрувати дані за параметрами, значеннями, датами та джерелами, а також будувати графіки (таблиці, гістограми, часові ряди, теплові карти, добовий аналіз).

– Статистика відображає загальні метрики (кількість точок, вимірювань) і детальну статистику за параметрами та джерелами.

Використання бібліотеки Plotly забезпечує інтерактивні графіки з підтримкою масштабування, фільтрації та спливаючих підказок.

2) Взаємодія з сервером

Клієнт взаємодіє з сервером через HTTP-запити, використовуючи бібліотеку requests. Основні функції для отримання даних:

– get_all_points – отримує список точок моніторингу.

– get_point_data – повертає дані конкретної точки.

– get_all_data – забезпечує фільтрацію даних.

– get_statistics – отримує статистичні показники.

Для оптимізації продуктивності використано декоратор @st.cache_data, який кешує результати запитів на 60 секунд (для даних) або 300 секунд (для статистики).

3) Обробка сповіщень

Клієнт підтримує підписку на сповіщення через форму, де користувач вказує точку, параметр і порогове значення. Функція create_subscription відправляє запит на сервер для створення підписки. Сповіщення перевіряються функцією check_notifications, яка читає JSON-файл notifications.json і відображає нові повідомлення у вигляді тостів (спливаючих вікон). Унікальний ідентифікатор пристрою генерується за допомогою UUID і зберігається локально.

4) Візуалізація даних

Для створення графіків використано бібліотеку Plotly, яка підтримує різноманітні типи візуалізацій:

– Гістограми для аналізу розподілу значень.

– Часові ряди з трендами та категоріями якості.

– Стовпчикові діаграми для порівняння станцій.

– Теплові карти для аналізу значень за параметрами та станціями.

– Добовий аналіз із відображенням середніх значень за годинами та днями тижня.

Графіки адаптуються до темної або світлої теми, яку користувач може перемикати.

5) Експорт даних

Користувач може експортувати відфільтровані дані у форматах CSV та Excel. Для створення Excel-файлів використано бібліотеку openpyxl, а для CSV – вбудовані можливості pandas. Це забезпечує гнучкість для подальшого аналізу даних поза системою.

2.2.3. Інтеграція та оптимізація

Інтеграція клієнтського та серверного модулів здійснюється через RESTful API, що забезпечує чітке розділення функціоналу та можливість масштабування. Для підвищення продуктивності використано кешування запитів на клієнтській стороні та оптимізацію запитів до бази даних на сервері (наприклад, обмеження кількості повернених записів через параметри limit і offset). Обробка помилок реалізована на обох рівнях: сервер повертає відповідні HTTP-коди помилок (400, 404, 500), а клієнт відображає користувачеві зрозумілі повідомлення.

Система підтримує адаптивний дизайн завдяки Streamlit, що дозволяє комфортно працювати як на настільних комп’ютерах, так і на мобільних пристроях. Локалізація інтерфейсу українською мовою та використання

українських назв міст (Київ, Харків, Одеса тощо) забезпечують зручність для цільової аудиторії.

Реалізація модулів системи оповіщення про екологічну небезпеку базується на сучасних технологіях (Flask, Streamlit, SQLAlchemy, Plotly), що забезпечують високу функціональність і зручність використання. Серверний модуль ефективно обробляє, зберігає та генерує дані, тоді як клієнтський модуль надає інтуїтивно зрозумілий інтерфейс для моніторингу, аналізу та сповіщень. Інтеграція модулів через API та оптимізація продуктивності створюють надійну основу для подальшого розвитку системи, наприклад, додавання нових джерел даних або розширення функціоналу сповіщень.

2.3. Дизайн інтерфейсу користувача

Дизайн інтерфейсу користувача (UI) інформаційної системи оповіщення про екологічну небезпеку розроблено з урахуванням принципів зручності, інтуїтивності та адаптивності. Інтерфейс реалізовано за допомогою бібліотеки Streamlit, яка забезпечує швидке створення веб-додатків із підтримкою інтерактивних елементів. Основна мета дизайну – забезпечити користувачам легкий доступ до даних про якість повітря, можливість аналізу цих даних та налаштування сповіщень про екологічні загрози. У цьому підрозділі детально описано всі екранні форми системи, їх функціонал, структуру та візуальні особливості, що сприяють ефективній взаємодії з користувачем.

2.3.1. Загальна структура інтерфейсу

Інтерфейс системи побудовано за принципом вкладкової навігації, що дозволяє логічно організувати функціонал і уникнути перевантаження інформацією. Основний екран поділено на три вкладки: Карта моніторингу, Аналіз даних та Статистика. Кожна вкладка відповідає за окремий елемент роботи з даними про якість повітря. Додатково, бічна панель (sidebar) містить

допоміжні елементи управління, такі як перемикач темного/світлого режиму, кнопка оновлення даних, довідкову інформацію та унікальний ідентифікатор пристрою для сповіщень.

Візуальний стиль інтерфейсу адаптивний і підтримує дві теми оформлення – темну та світлу, що забезпечує комфортне використання в різних умовах освітлення. Темна тема використовується за замовчуванням, оскільки вона зменшує навантаження на очі при тривалій роботі з даними. Шрифти, кольори та компоненти відповідають сучасним стандартам дизайну, а саме: чітка ієрархія елементів, контрастність для легкого сприйняття та мінімалістичний підхід до оформлення.

2.3.2. Опис екранних форм

Нижче наведено детальний опис кожної екранної форми, її компонентів, функціоналу та особливостей дизайну.

Головний екран та вкладка "Карта моніторингу". Вкладка "Карта моніторингу" (рисунок 2.3) є основною точкою входу для користувача. Вона призначена для відображення географічного розташування точок моніторингу якості повітря та детальної інформації про вибрану станцію.

Рисунок 2.3 – Вкладка "Карта моніторингу"

Компоненти інтерфейсу:

1) Заголовок додатку розташований у верхній частині екрана, виконаний великим шрифтом із текстом "Система оповіщення про екологічну небезпеку" (рисунок 2.4).

Рисунок 2.4 – Заголовок додатку

2) Карта (рисунок 2.5) – основний елемент вкладки, який займає більшу частину лівої колонки (2/3 ширини екрана). Карта відображає точки моніторингу у вигляді маркерів, колір яких залежить від рівня забруднення (зелений – добра якість, помаранчевий – середня, червоний – погана). Для реалізації карти використовується вбудований компонент Streamlit st.map.

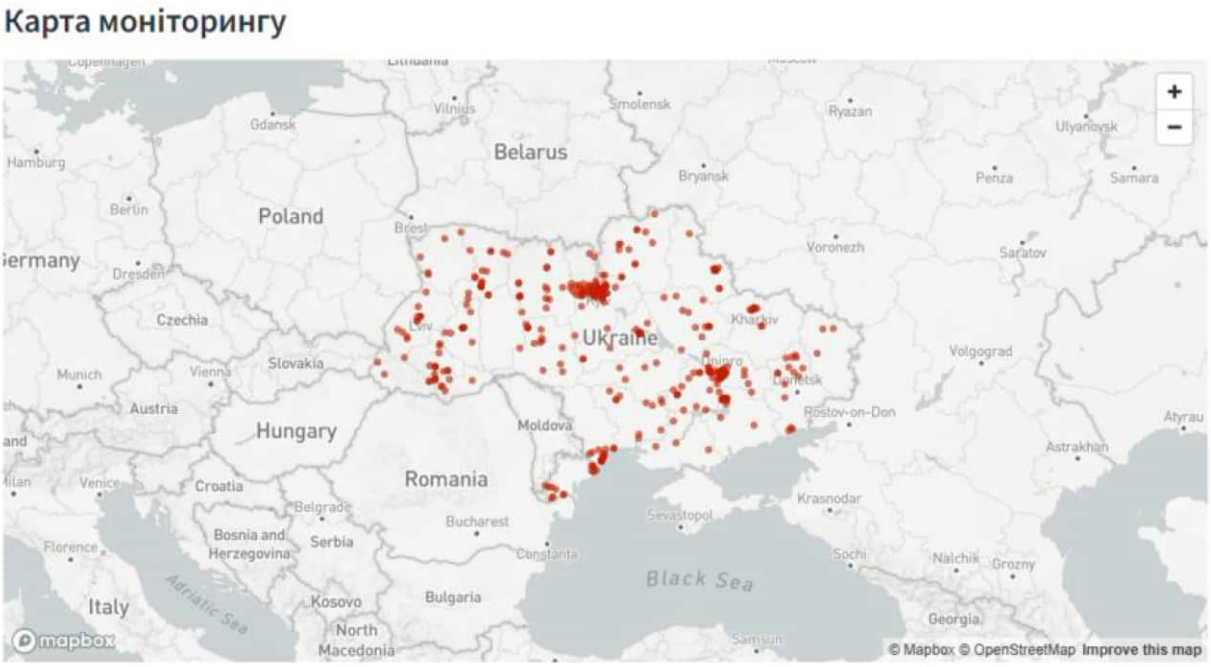


Рисунок 2.5 – Карта моніторингу

3) Легенда маркерів (рисунок 2.6) під картою розташовано текстове пояснення кольорів маркерів (зелений, помаранчевий, червоний) у форматі Markdown, що допомагає користувачу зрозуміти значення кольорів.

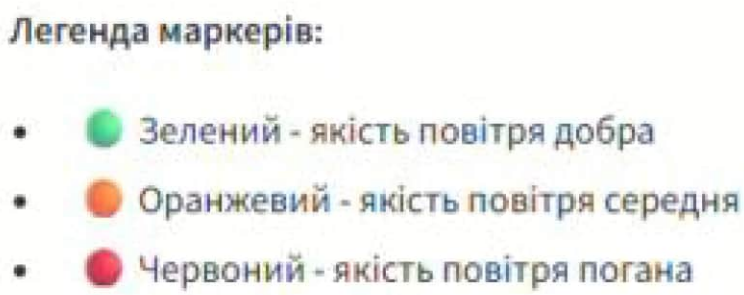


Рисунок 2.6 – Легенда маркерів

4) Інформація про джерела даних (рисунок 2.7) – текстовий блок, який показує кількість відображених точок і джерела даних (наприклад, SaveEcoBot, OpenAQ, Eco-City).

Рисунок 2.7 – Інформація про джерела даних

5) Фільтр джерел даних (рисунок 2.8) – випадаючий список (st.selectbox) для вибору джерела даних ("Всі джерела" або конкретне джерело). Це дозволяє користувачу обмежити відображення точок за певним джерелом.

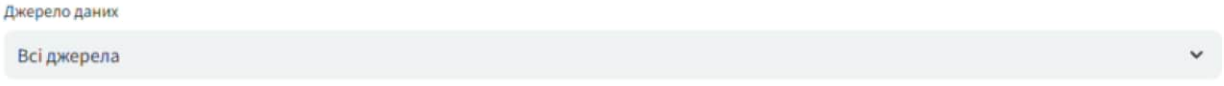


Рисунок 2.8 – Джерела даних

6) Вибір точки моніторингу (рисунок 2.9) – випадаючий список із переліком станцій, де кожна опція містить назву станції та її джерело даних. При виборі точки оновлюється права колонка з детальною інформацією.

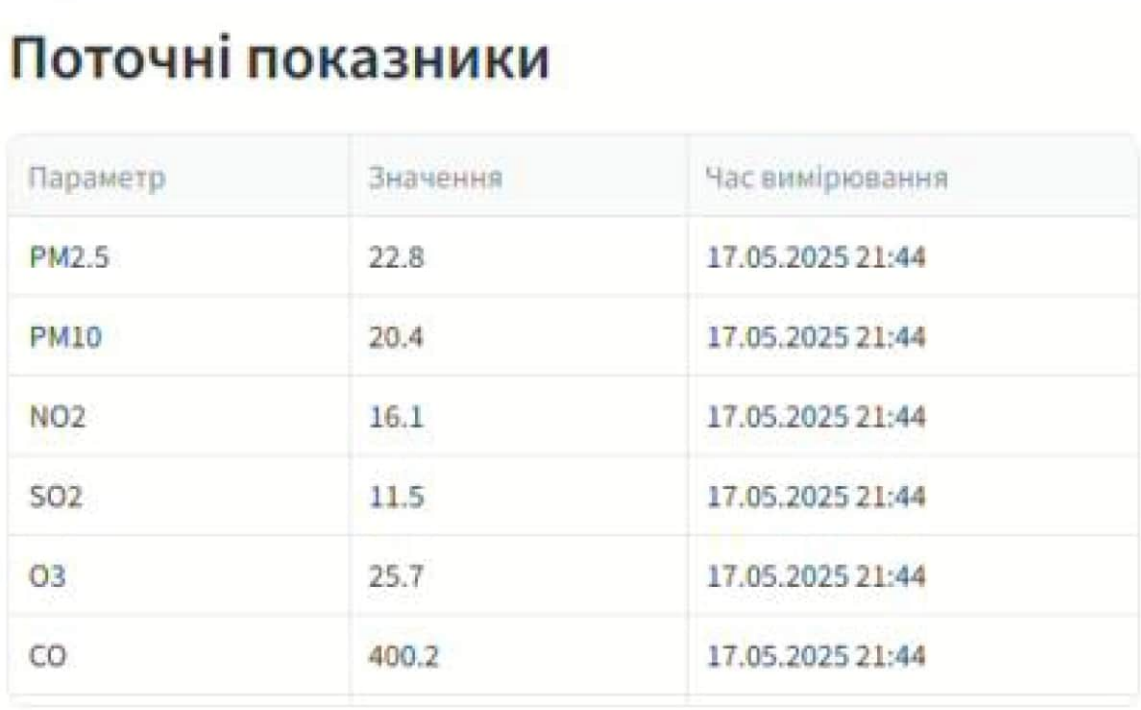
Рисунок 2.9 – Вибір точки моніторингу

7) Права колонка з даними точки:

– Назва точки, джерело та координати відображаються у текстовому форматі (рисунок 2.10).

Рисунок 2.10 – Дані моніторингу

– Таблиця поточних показників (рисунок 2.11) стилізована таблиця (st.dataframe), що містить параметри (наприклад, PM2.5, PM10, AQI), їх значення з форматуванням (з емодзі для AQI), одиниці виміру та час вимірювання.



Поточні показники

Параметр	Значення	Час вимірювання
PM2.5	22.8	17.05.2025 21:44
PM10	20.4	17.05.2025 21:44
NO2	16.1	17.05.2025 21:44
SO2	11.5	17.05.2025 21:44
O3	25.7	17.05.2025 21:44
CO	400.2	17.05.2025 21:44

Рисунок 2.11 – Таблиця поточних показників

– Графік показників (рисунок 2.12) – інтерактивний графік, створений за допомогою Plotly, що відображає динаміку вибраного параметра (вибір через st.selectbox). Графік включає кольорові зони (зелену, помаранчеву, червону) для позначення рівнів безпеки, а також лінію тренду.

Рисунок 2.12 – Графік показників

– Форма підписки на сповіщення (рисунок 2.13) – форма (st.form) містить вибір параметра, поле для введення порогового значення (st.number_input) та кнопку "Підписатися на сповіщення". Порогове значення за замовчуванням на 20% вище поточного.

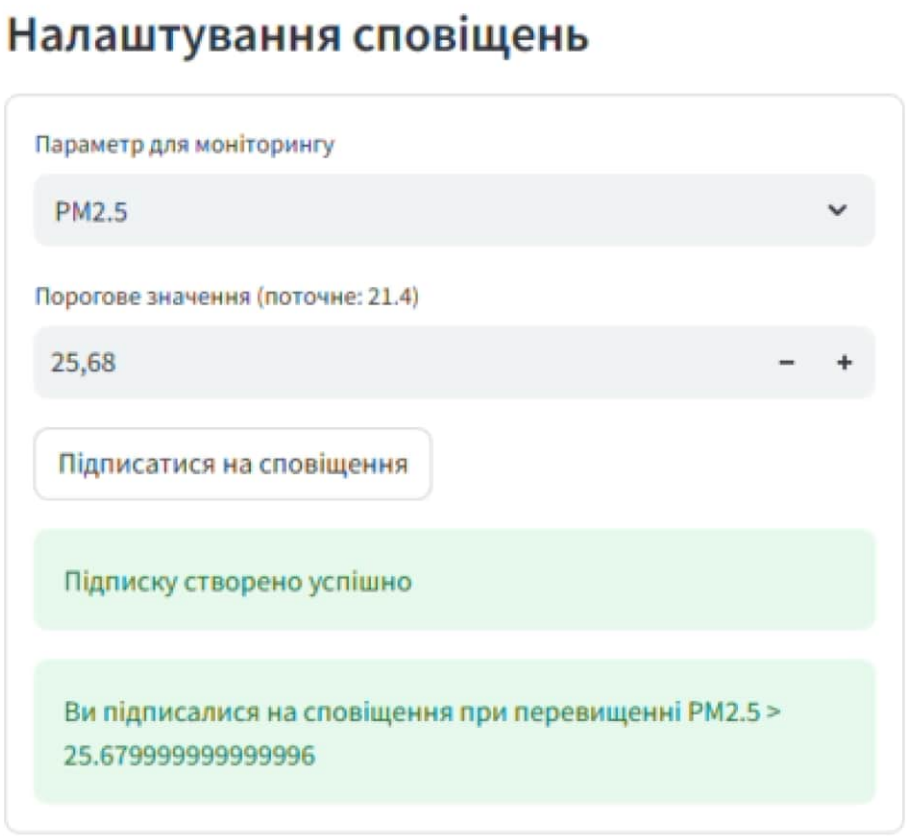


Рисунок 2.13 – Форма підписки на сповіщення

– Останні повідомлення (рисунок 2.14) - розділ із розкриваючимися блоками (st.expander), де відображаються до 5 останніх сповіщень із заголовком, текстом, часом і кнопкою "Перейти до точки" для швидкої навігації.

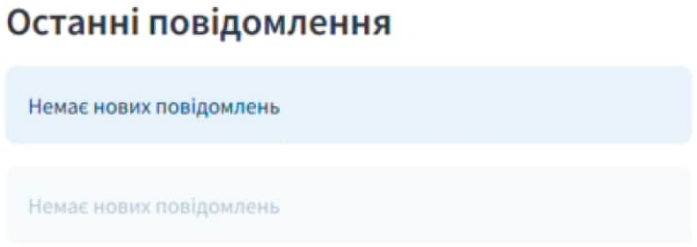


Рисунок 2.14 – Останні повідомлення

Особливості дизайну:

– Використання двоколонного макета (2:1) забезпечує баланс між картою та детальною інформацією.

– Інтерактивні елементи (випадаючі списки, кнопки) мають чітке візуальне виділення при наведенні.

– Графіки Plotly підтримують масштабування та відображення даних при наведенні, що полегшує аналіз.

– Таблиця показників адаптивна, її висота залежить від кількості параметрів.

– Сповіщення відображаються як тости (st.toast) для ненав’язливого інформування.

Вкладка "Аналіз даних" (рисунок 2.15) призначена для поглибленого аналізу даних моніторингу. Вона дозволяє користувачам фільтрувати дані за різними критеріями та візуалізувати Розділ включає кілька типів аналізу, таких як таблиця даних, розподіл значень, часові ряди, порівняння станцій, теплова карта та добовий аналіз.

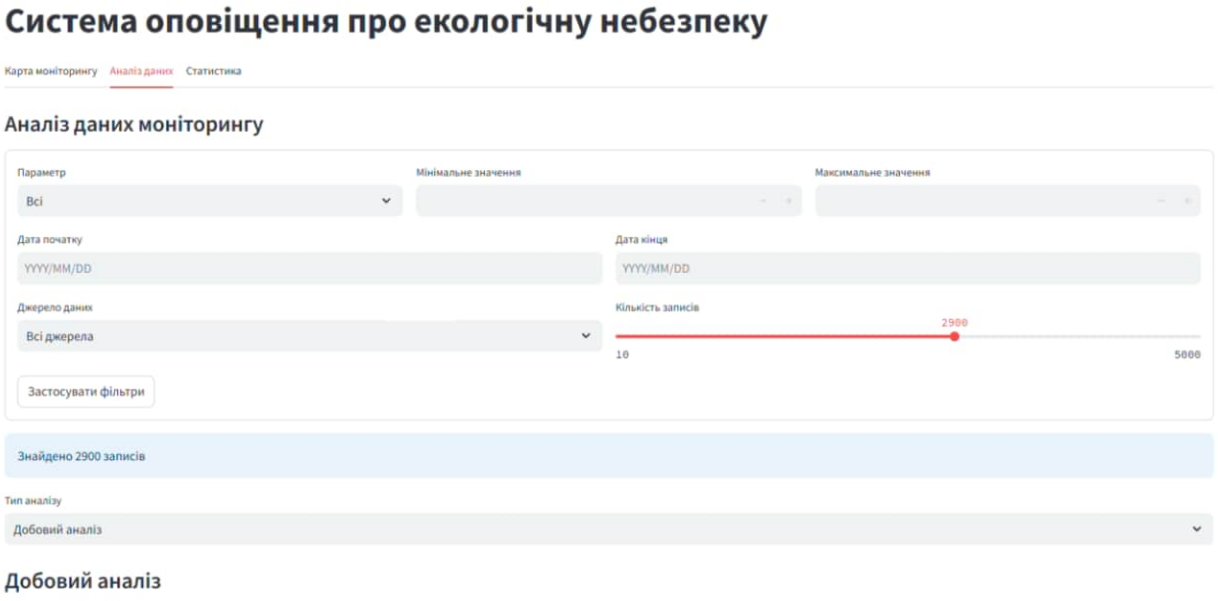


Рисунок 2.15 – Вкладка "Аналіз даних"

Компоненти інтерфейсу:

1) Форма фільтрації:

– Параметр (рисунок 2.16) – випадаючий список із переліком доступних параметрів (наприклад, PM2.5, PM10) або опція "Всі".

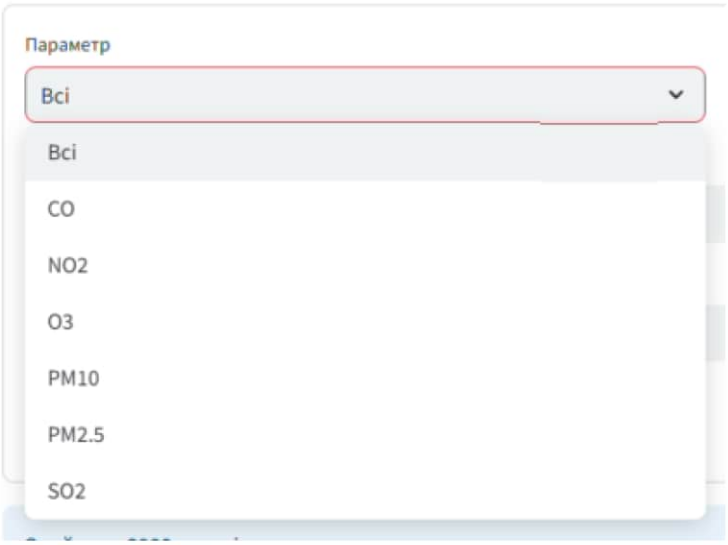


Рисунок 2.16 - Параметр

– Мінімальне та максимальне значення (рисунок 2.17) – поля введення чисел (st.number_input) для обмеження діапазону значень.

Рисунок 2.17 – Мінімальне та максимальне значення

– Дата початку та кінця (рисунок 2.18) – поля вибору дати (st.date_input) для фільтрації за часовим діапазоном.

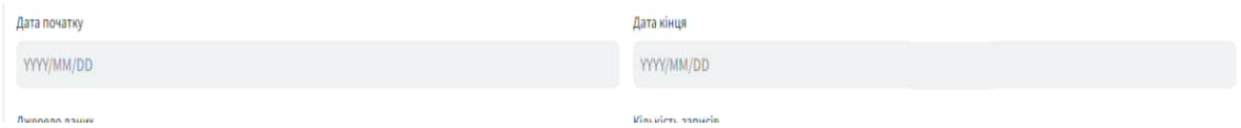


Рисунок 2.18 – Дата початку та кінця

– Джерело даних (рисунок 2.19) – випадаючий список із джерелами або "Всі джерела".

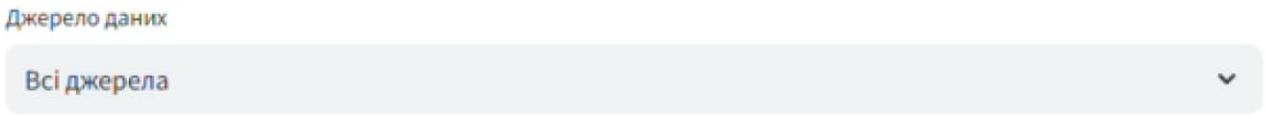


Рисунок 2.19 – Джерело даних

– Кількість записів (рисунок 2.20) – повзунок (st.slider) для вибору ліміту записів (10–5000).

Рисунок 2.20 – Кількість запитів

– Кнопка "Застосувати фільтри" (рисунок 2.21) – запускає запит до API для отримання відфільтрованих даних.

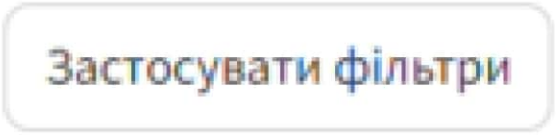


Рисунок 2.21 – Кнопка "Застосувати фільтри"

2) Інформація про дані (рисунок 2.22) - текстовий блок, що показує кількість знайдених записів.

Рисунок 2.22 – Інформація про дані

3) Тип аналізу (рисунок 2.23) - випадаючий список із варіантами аналізу:

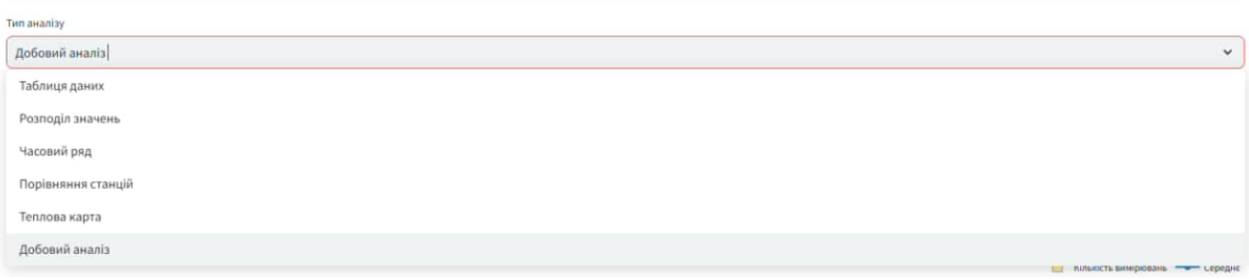
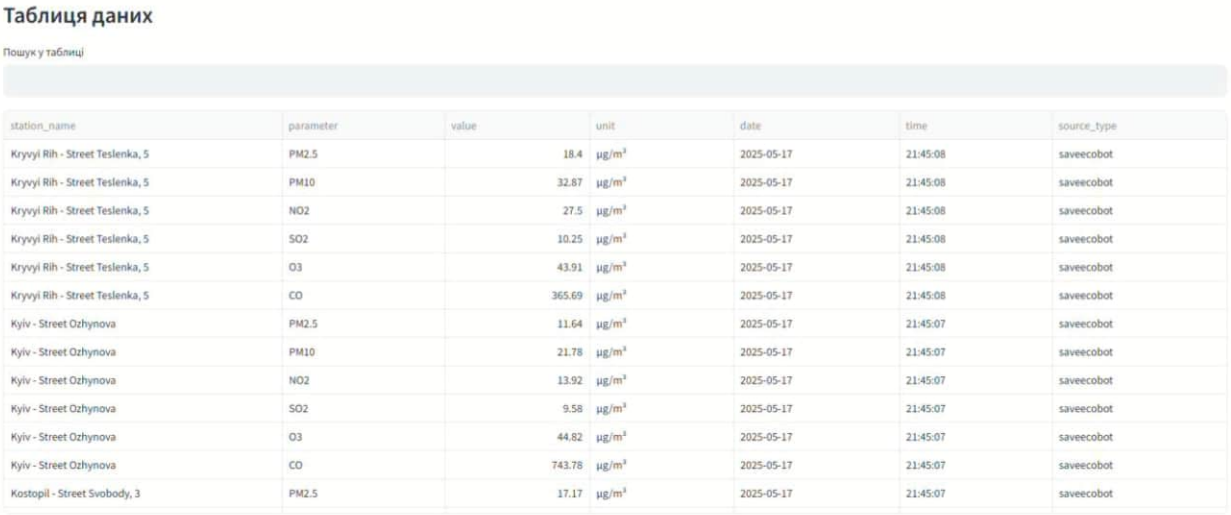


Рисунок 2.23 – Тип аналізу

– Таблиця даних (рисунок 2.24) відображає відфільтровані дані у вигляді таблиці з пошуком (st.text_input). Таблиця включає колонки: назва станції, параметр, значення, одиниця виміру, дата, час, джерело.



station_name	parameter	value	unit	date	time	source_type
Kryvyi Rih - Street Teslenka, 5	PM2.5	18.4	µg/m³	2025-05-17	21:45:08	saveecobot
Kryvyi Rih - Street Teslenka, 5	PM10	32.87	µg/m³	2025-05-17	21:45:08	saveecobot
Kryvyi Rih - Street Teslenka, 5	NO2	27.5	µg/m³	2025-05-17	21:45:08	saveecobot
Kryvyi Rih - Street Teslenka, 5	SO2	10.25	µg/m³	2025-05-17	21:45:08	saveecobot
Kryvyi Rih - Street Teslenka, 5	O3	43.91	µg/m³	2025-05-17	21:45:08	saveecobot
Kryvyi Rih - Street Teslenka, 5	CO	365.69	µg/m³	2025-05-17	21:45:08	saveecobot
Kyiv - Street Ozhynova	PM2.5	11.64	µg/m³	2025-05-17	21:45:07	saveecobot
Kyiv - Street Ozhynova	PM10	21.78	µg/m³	2025-05-17	21:45:07	saveecobot
Kyiv - Street Ozhynova	NO2	13.92	µg/m³	2025-05-17	21:45:07	saveecobot
Kyiv - Street Ozhynova	SO2	9.58	µg/m³	2025-05-17	21:45:07	saveecobot
Kyiv - Street Ozhynova	O3	44.82	µg/m³	2025-05-17	21:45:07	saveecobot
Kyiv - Street Ozhynova	CO	743.78	µg/m³	2025-05-17	21:45:07	saveecobot
Kostopil - Street Svobody, 3	PM2.5	17.17	µg/m³	2025-05-17	21:45:07	saveecobot

Рисунок 2.24 -Таблиця даних

– Розподіл значень (рисунок 2.25) – гістограма (Plotly), що показує розподіл значень для вибраного параметра з лініями критичних рівнів (наприклад, 12 і 35 для PM2.5). Додатково відображаються метрики: середнє, медіана, мінімум, максимум.

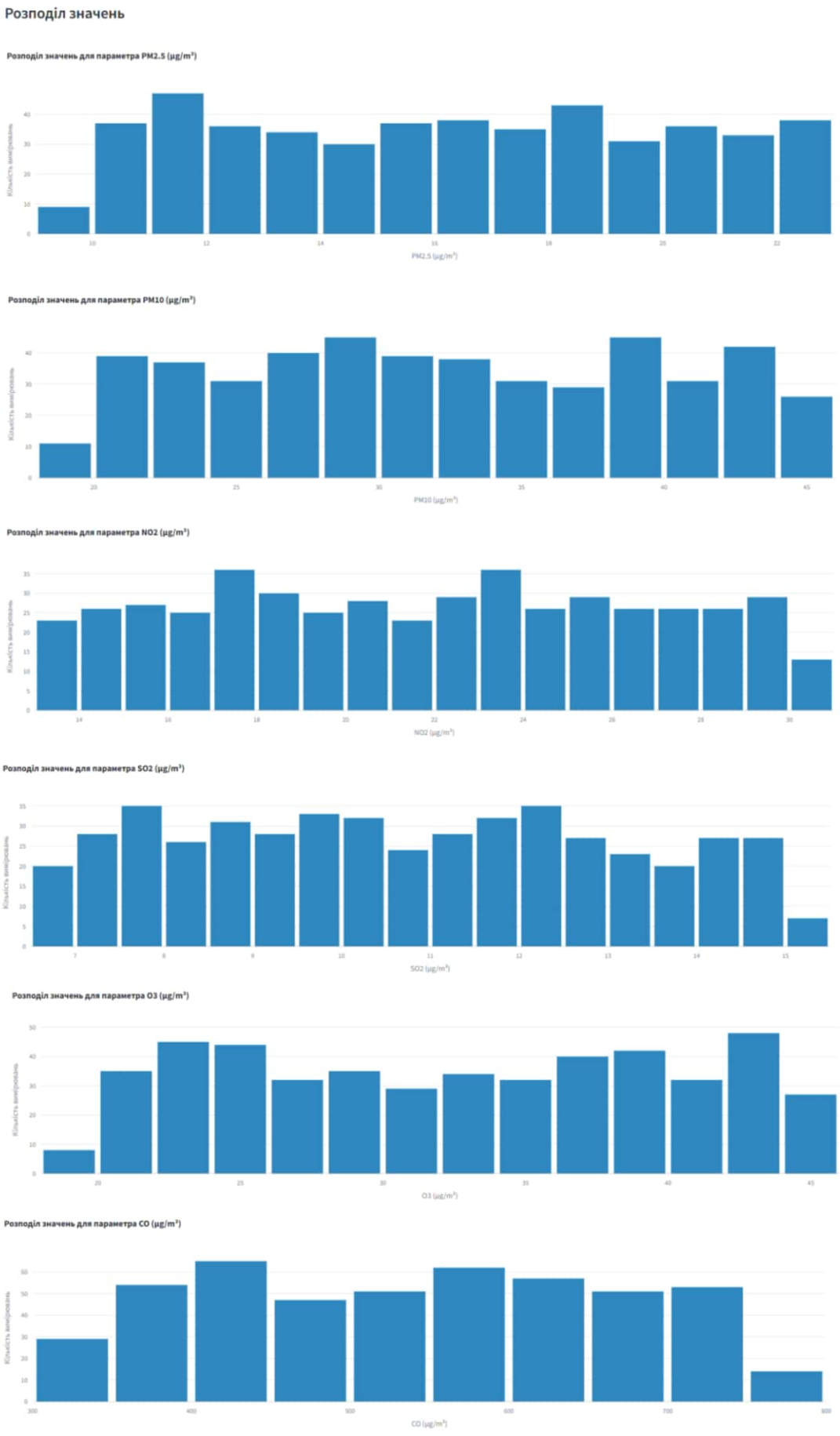


Рисунок 2.25 – Розподіл значень

– Часовий ряд (рисунок 2.26) – лінії (Plotly) для кількох станцій і параметрів із кольоровими зонами безпеки. Користувач може вибрати станції та параметри через st.multiselect.

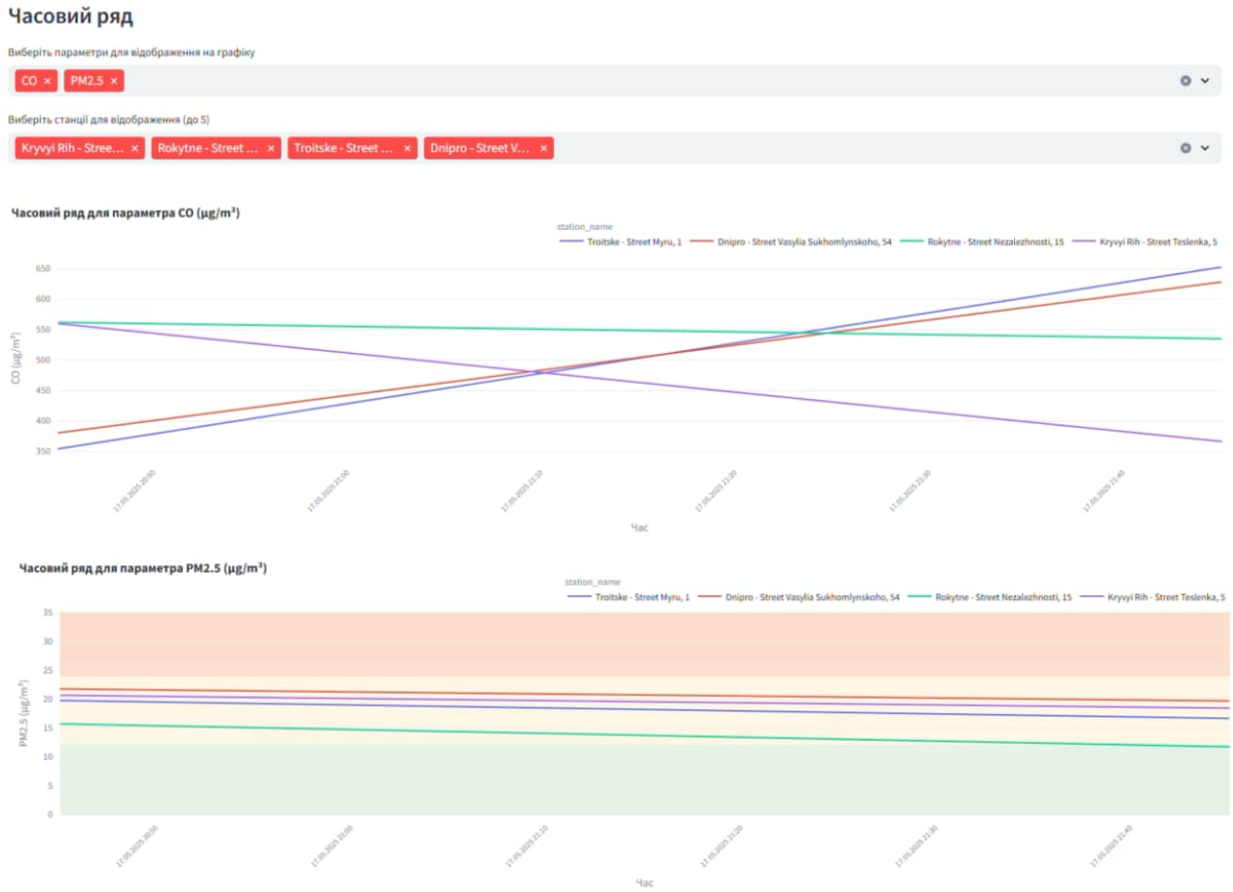


Рисунок 2.26 – Часовий ряд

– Порівняння станцій (рисунок 2.27) – стовпчикова діаграма середніх значень для різних станцій із таблицею статистики (середнє, медіана, мінімум, максимум, кількість).

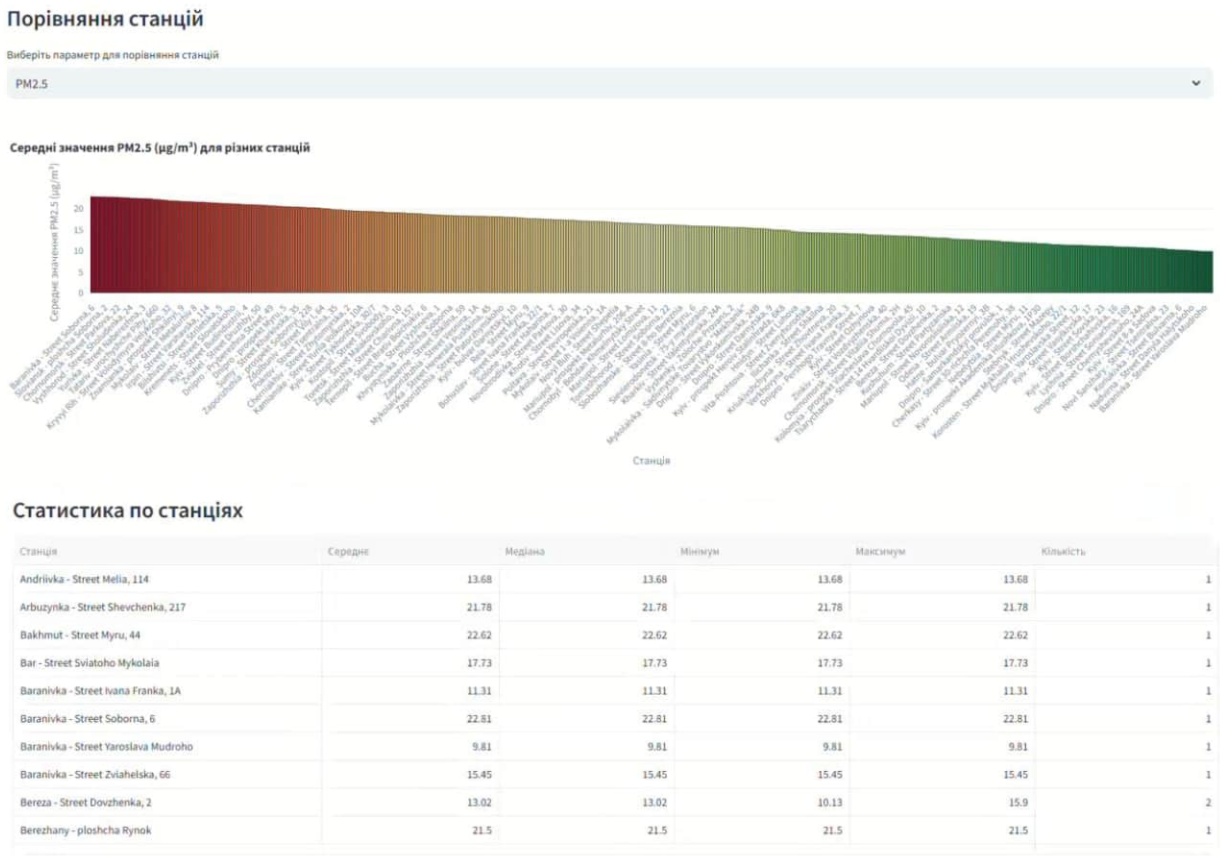


Рисунок 2.27 – Порівняння станцій

– Теплова карта (Plotly) (рисунок 2.28) для порівняння середніх значень параметрів між станціями. Користувач може вибрати до 15 станцій.

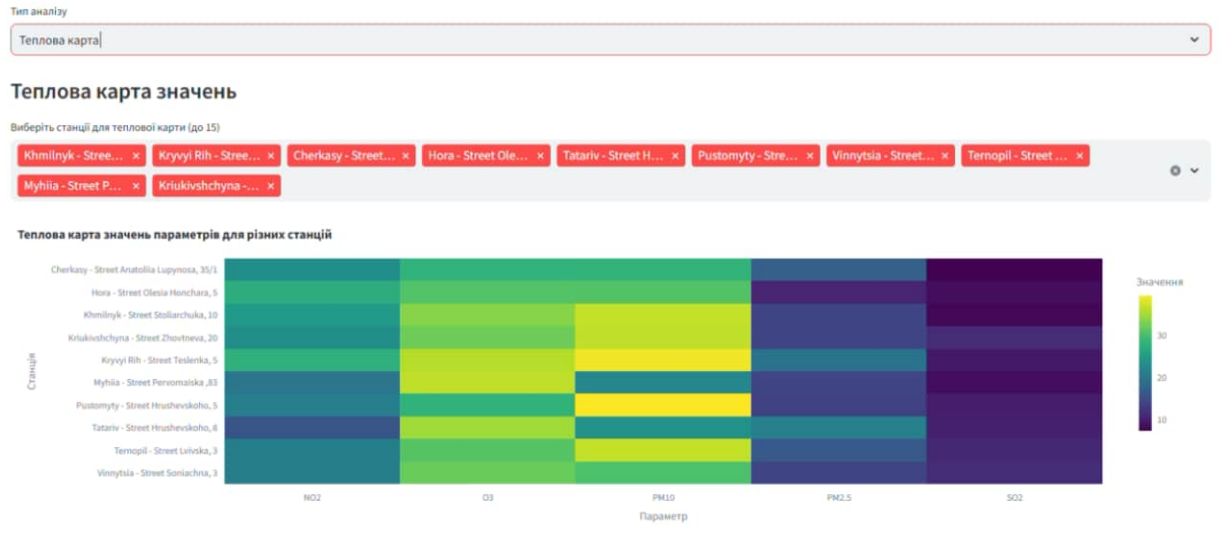


Рисунок 2.28 – Теплова карта

– Добовий аналіз (рисунок 2.29) – комбінований графік із середніми значеннями за годинами, областю мін-макс і гістограмою кількості вимірювань. Додатково – стовпчикова діаграма середніх значень за днями тижня.

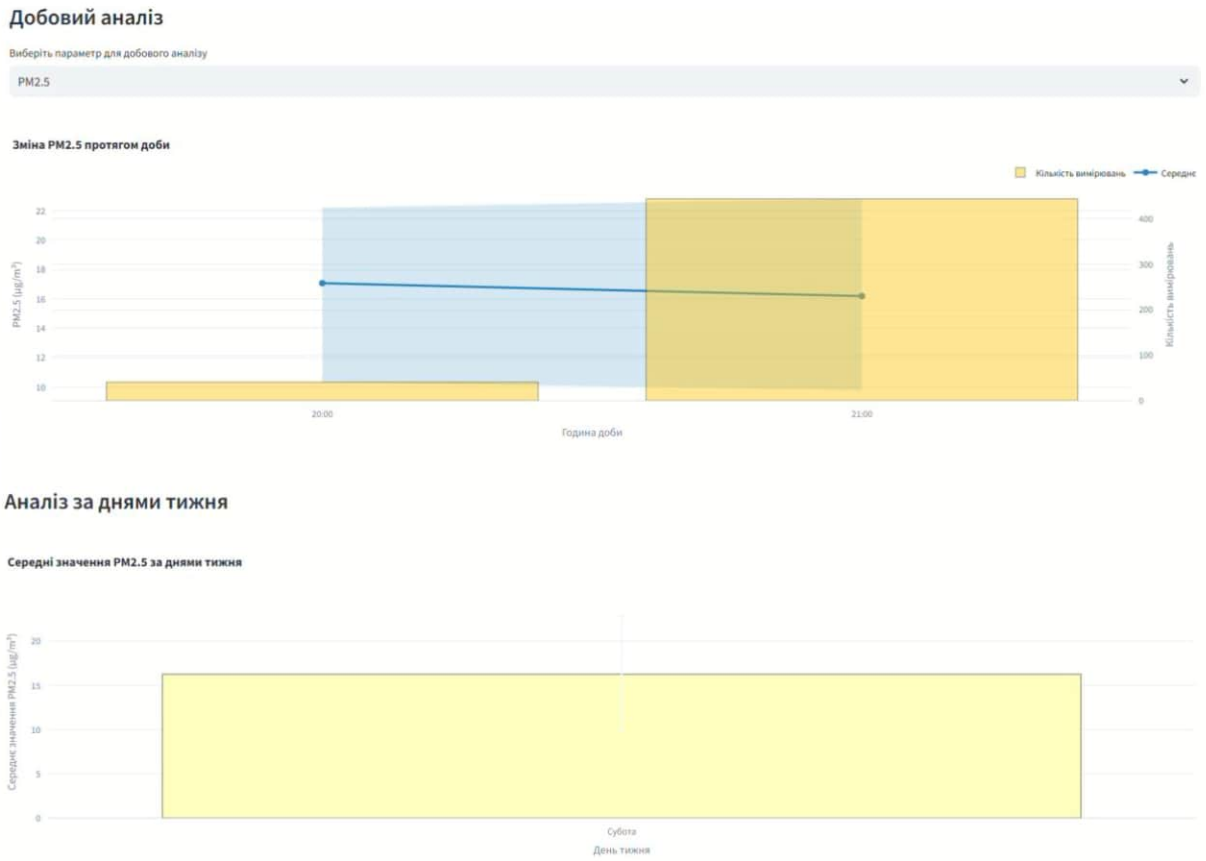


Рисунок 2.29 – Добовий аналіз

4) Експорт даних (рисунок 2.30) – кнопки для завантаження даних у форматах CSV та Excel.

Рисунок 2.30 – Експорт даних

Особливості дизайну:

– Форма фільтрації розташована у трьох колонках для компактності.

– Графіки мають уніфікований стиль із підтримкою темного/світлого режиму.

– Теплова карта адаптується до кількості станцій, змінюючи висоту.

– Таблиця підтримує пошук і має адаптивну висоту (500 пікселів).

– Кнопки експорту розташовані у двох колонках для симетрії.

Вкладка "Статистика" (рисунок 2.31) надає узагальнену інформацію про систему моніторингу, включаючи кількість точок, джерела даних, статистику параметрів і часовий аналіз.

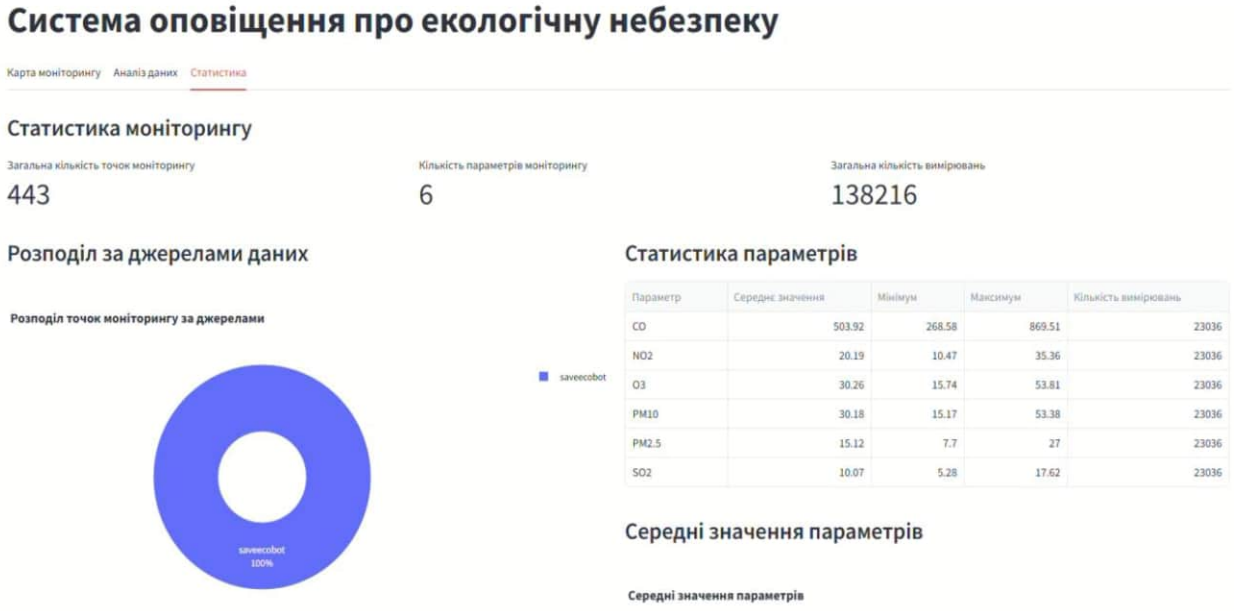


Рисунок 2.31 – Статистика

Компоненти інтерфейсу:

1) Загальні метрики (рисунок 2.32): три колонки з метриками (st.metric): кількість точок, параметрів і вимірювань.

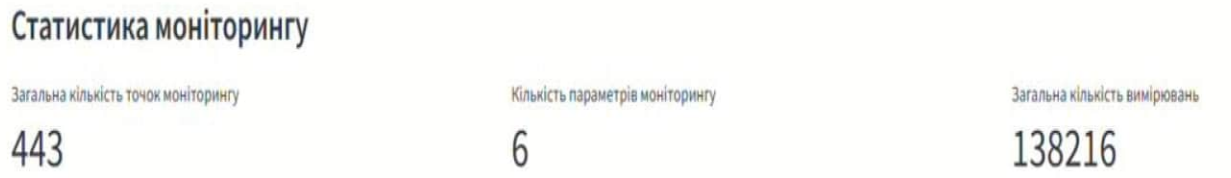


Рисунок 2.32 – Загальні метрики

2) Розподіл за джерелами:

– Кільцева діаграма (Plotly) (рисунок 2.33) із відсотковим розподілом точок за джерелами.

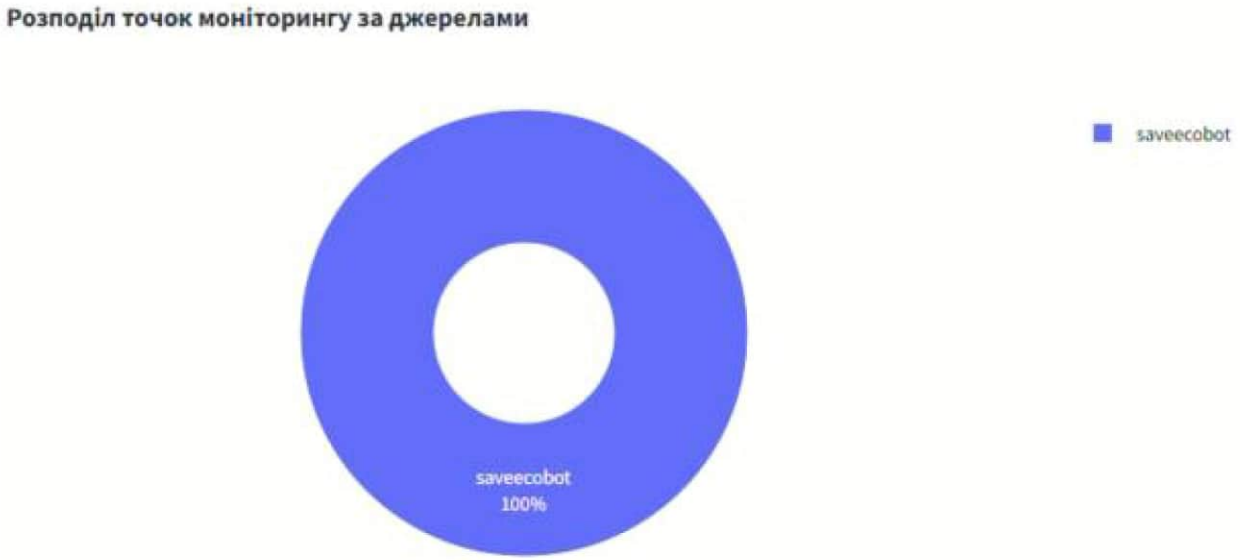


Рисунок 2.33 – Кільцева діаграма

– Таблиця з кількістю точок (рисунок 2.34) для кожного джерела.

Джерело	Кількість
saveecobot	443

Рисунок 2.34 – Таблиця з кількістю точок

3) Статистика параметрів:

– Таблиця з параметрами (рисунок 2.35), середніми значеннями, мінімумом, максимумом і кількістю вимірювань.

Статистика параметрів

Параметр	Середнє значення	Мінімум	Максимум	Кількість вимірювань
CO	503.92	268.58	869.51	23036
NO2	20.19	10.47	35.36	23036
O3	30.26	15.74	53.81	23036
PM10	30.18	15.17	53.38	23036
PM2.5	15.12	7.7	27	23036
SO2	10.07	5.28	17.62	23036

Рисунок 2.35 – Таблиця з параметрами

– Стовпчикова діаграма середніх значень (рисунок 2.36) із кольоровим градієнтом за кількістю вимірювань.

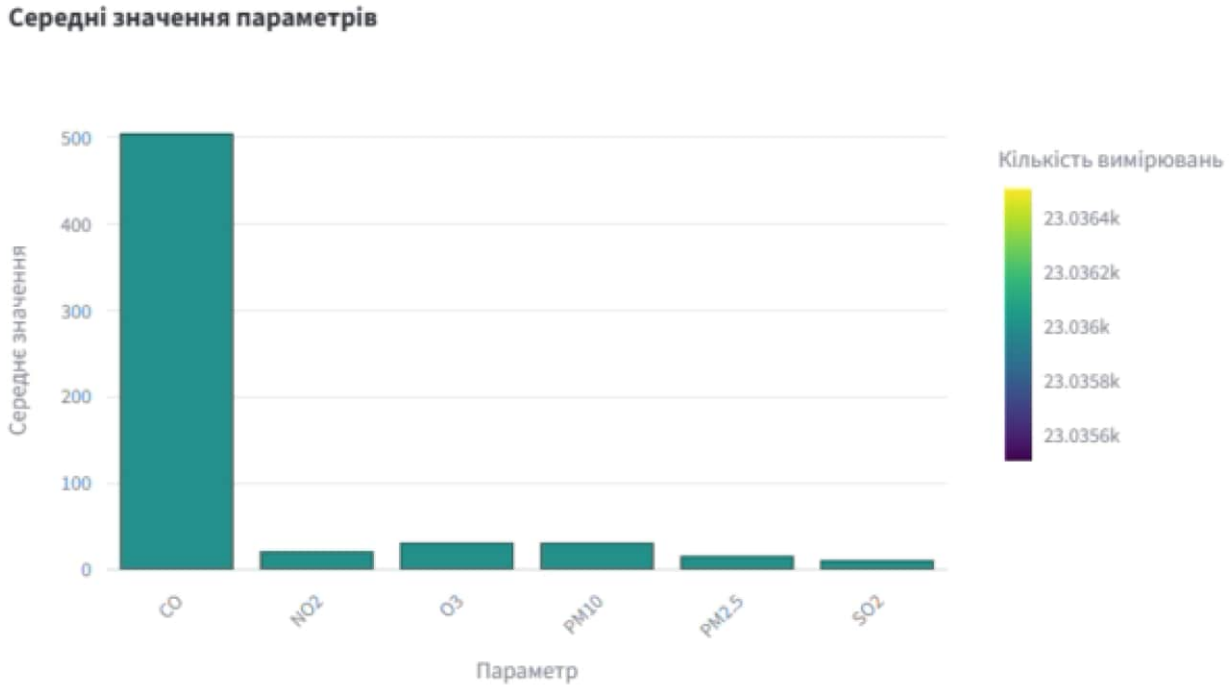


Рисунок 2.36 – Стовпчикова діаграма

– Групова стовпчикова діаграма (рисунок 2.37) для порівняння мін/макс/середнього для топ-5 параметрів.

Рисунок 2.37 – Групова стовпчикова діаграма

4) Часовий аналіз (рисунок 2.38):

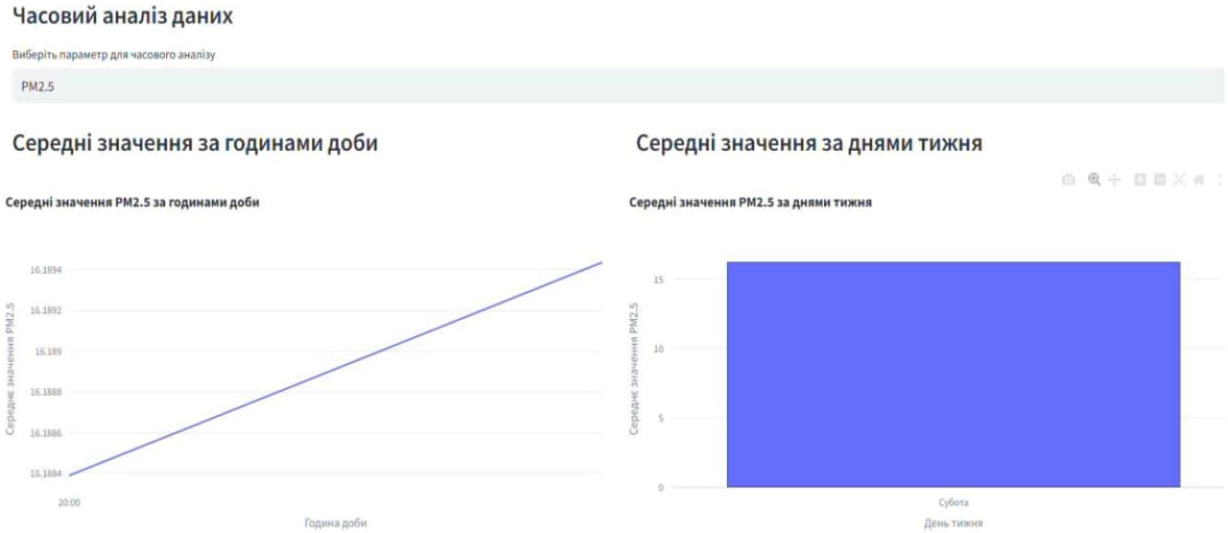


Рисунок 2.38 – Часовий аналіз

– Випадаючий список для вибору параметра (рисунок 2.39).

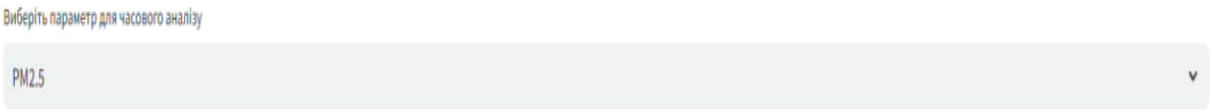


Рисунок 2.39 – Вибір параметра

– Дві колонки:

✓ Лінія середніх значень за годинами доби (рисунок 2.40).

Рисунок 2.40 – Лінія середніх значень

✓ Стовпчики середніх значень за днями тижня.

Рисунок 2.41 – Стовпчики середніх значень

– Теплова карта значень за годинами та днями тижня (Plotly) (рисунок 2.42) із 24-годинною віссю та днями тижня.

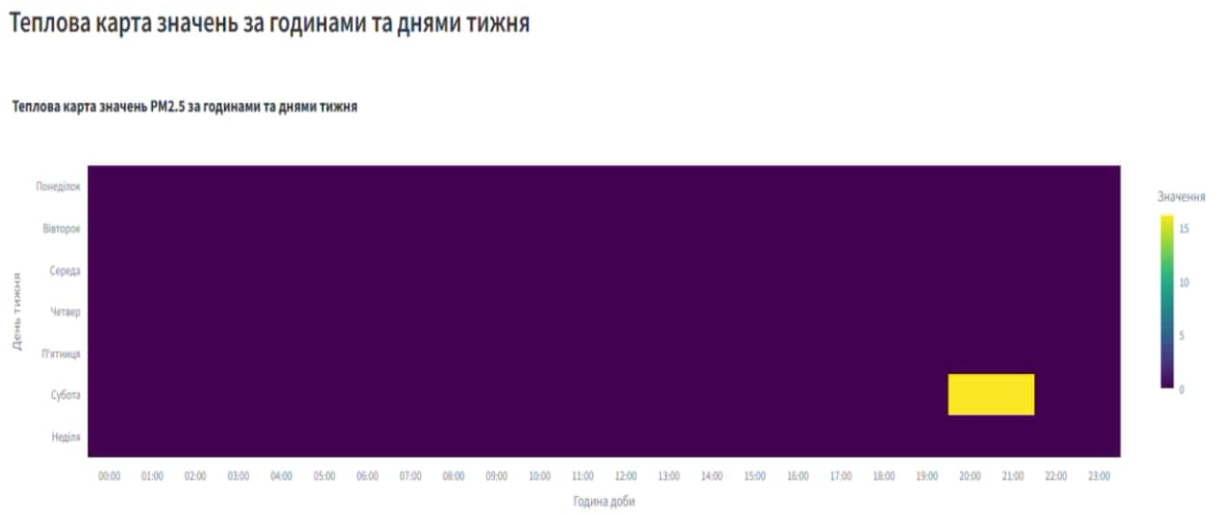


Рисунок 2.42 – Теплова карта значень за годинами та днями тижня

Особливості дизайну:

– Метрики відображаються у верхній частині для швидкого огляду.

– Діаграми мають компактний розмір із текстом усередині (для кільцевої діаграми).

– Теплова карта використовує палітру Viridis для чіткості.

– Таблиці мають фіксовану ширину для зручного читання.

2.3.3. Принципи дизайну та ергономіка

Дизайн інтерфейсу базується на таких принципах:

– Інтуїтивність. Вкладкова структура та чіткі підписи полегшують навігацію.

– Адаптивність. Інтерфейс адаптується до різних розмірів екрана завдяки параметру layout="wide" у Streamlit.

– Контрастність. Темна тема використовує світлий текст на темному тлі, а світла – темний текст на світлому для читабельності.

– Інтерактивність. Графіки Plotly дозволяють масштабувати, переглядати значення при наведенні та взаємодіяти з даними.

– Мінімалізм. Лише необхідні елементи відображаються на екрані, щоб уникнути перевантаження.

Ергономіка забезпечується завдяки:

– Логічному розташуванню елементів (карта зліва, деталі справа).

– Використанню випадаючих списків і повзунків для зменшення кількості введення даних.

– Автоматичному заповненню полів (наприклад, порогове значення на 20% вище поточного).

– Ненав’язливим сповіщенням через тости.

Дизайн інтерфейсу користувача системи оповіщення про екологічну небезпеку поєднує функціональність, естетику та зручність. Вкладкова структура, інтерактивні графіки, адаптивні таблиці та форми фільтрації забезпечують ефективну роботу з даними про якість повітря. Бічна панель доповнює основний інтерфейс довідковою інформацією та інструментами управління. Використання темного/світлого режиму та сучасних бібліотек (Streamlit, Plotly) робить систему доступною для широкого кола користувачів, від екологів до звичайних громадян, зацікавлених у моніторингу довкілля.

РОЗДІЛ 3. ТЕСТУВАННЯ ТА РЕЗУЛЬТАТИ РОБОТИ

3.1. Опис сценарію тестування

Тестування інформаційної системи оповіщення про екологічну небезпеку є ключовим етапом для забезпечення її надійності, функціональності та відповідності вимогам користувачів. Система, реалізована у вигляді клієнтської (client.py) та серверної (server.py) частин, призначена для моніторингу якості повітря, аналізу даних та своєчасного інформування користувачів про перевищення небезпечних рівнів забруднення. Для оцінки працездатності системи розроблено комплексний сценарій тестування, який охоплює функціональні, нефункціональні та інтеграційні показники. У цьому підрозділі детально описано сценарій тестування, включаючи цілі, методи, етапи та критерії оцінки результатів.

Основною метою тестування є перевірка коректності роботи всіх компонентів системи, їхньої взаємодії та здатності виконувати задані функції в реальних умовах. Конкретні цілі включають:

– Перевірку правильності відображення даних моніторингу на клієнтському інтерфейсі.

– Оцінку точності обробки та фільтрації даних через API-сервер.

– Тестування механізму сповіщень при перевищенні порогових значень забруднення.

– Перевірку стабільності системи при обробці великих обсягів даних.

– Оцінку зручності користувацького інтерфейсу (UI) та відповідності його принципам usability.

– Перевірку коректності роботи фонових задач сервера, таких як генерація синтетичних даних і очищення застарілих записів.

Для досягнення поставлених цілей використано комбінацію методів тестування:

– Функціональне тестування – перевірка виконання основних функцій системи, таких як відображення карти, аналіз даних, створення підписок і відправка сповіщень.

– Інтеграційне тестування – оцінка коректності взаємодії між клієнтською частиною, сервером і базою даних.

– Тестування продуктивності – аналіз швидкості обробки запитів і стабільності роботи при високому навантаженні.

– Тестування зручності (usability) – оцінка інтуїтивності інтерфейсу та зручності використання системи кінцевими користувачами.

– Регресійне тестування – перевірка збереження працездатності раніше реалізованих функцій після внесення змін до коду.

Тестування проводилося в контрольованому середовищі з використанням синтетичних даних, згенерованих сервером, а також реальних даних, отриманих через API SaveEcoBot. Для імітації різних сценаріїв використовувалися інструменти автоматизації, такі як Postman (для тестування API) і Selenium (для автоматизації взаємодії з інтерфейсом).

Сценарій тестування поділено на п’ять основних етапів, кожен з яких охоплює певний набір функціональних або нефункціональних вимог. Нижче наведено детальний опис кожного етапу.

Етап 1. Тестування клієнтського інтерфейсу

Цей етап зосереджений на перевірці коректності роботи клієнтської частини, реалізованої за допомогою Streamlit. Основні сценарії включають:

– Відображення карти моніторингу (рисунок 3.1): перевірка того, що карта коректно відображає точки моніторингу з відповідними кольоровими маркерами (зелений, оранжевий, червоний) залежно від рівня забруднення (наприклад, PM2.5, PM10, AQI). Тестується правильність нанесення координат (latitude, longitude) і відповідність кольорів значенням параметрів.

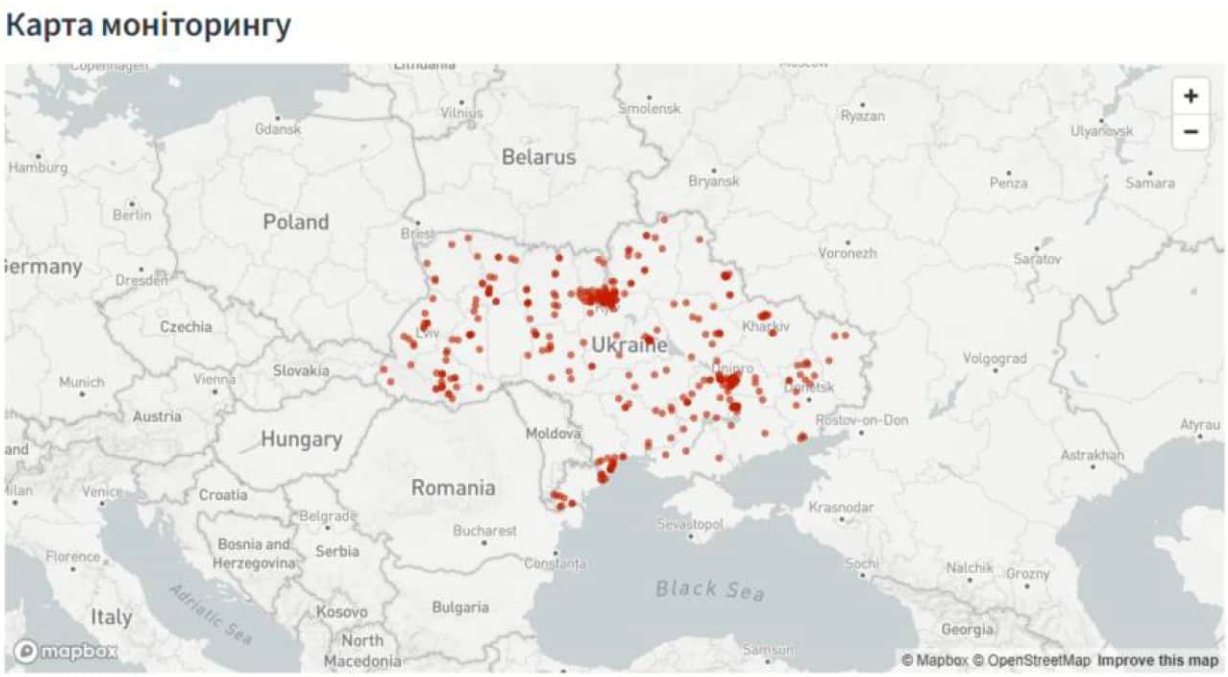


Рисунок 3.1 – Відображення карти моніторингу

– Вибір точки моніторингу (рисунок 3.2): перевірка функціональності вибору точки зі списку, відображення детальної інформації (назва, координати, джерело даних) і таблиці з останніми вимірами (рисунок 3.3). Тестується також фільтрація точок за джерелом даних (наприклад, SaveEcoBot, OpenAQ).

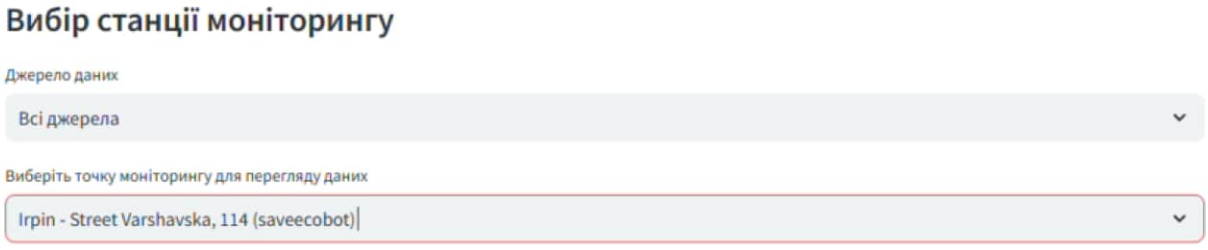


Рисунок 3.2 – Вибір точки моніторингу

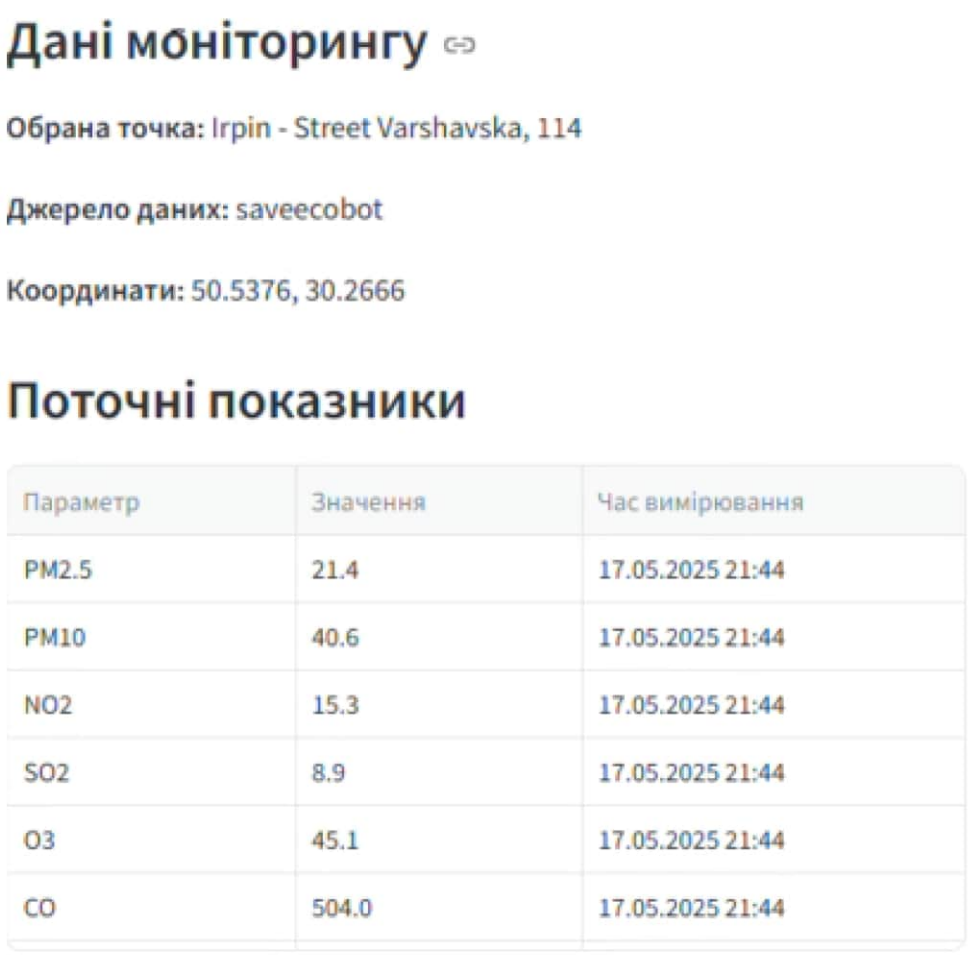


Рисунок 3.3 – Відображення детальної інформації про точку моніторингу

– Графіки показників (рисунок 3.4): оцінка коректності побудови графіків для вибраних параметрів (PM2.5, PM10 тощо) з урахуванням трендів, часових міток і кольорових зон (зелена, оранжева, червона).

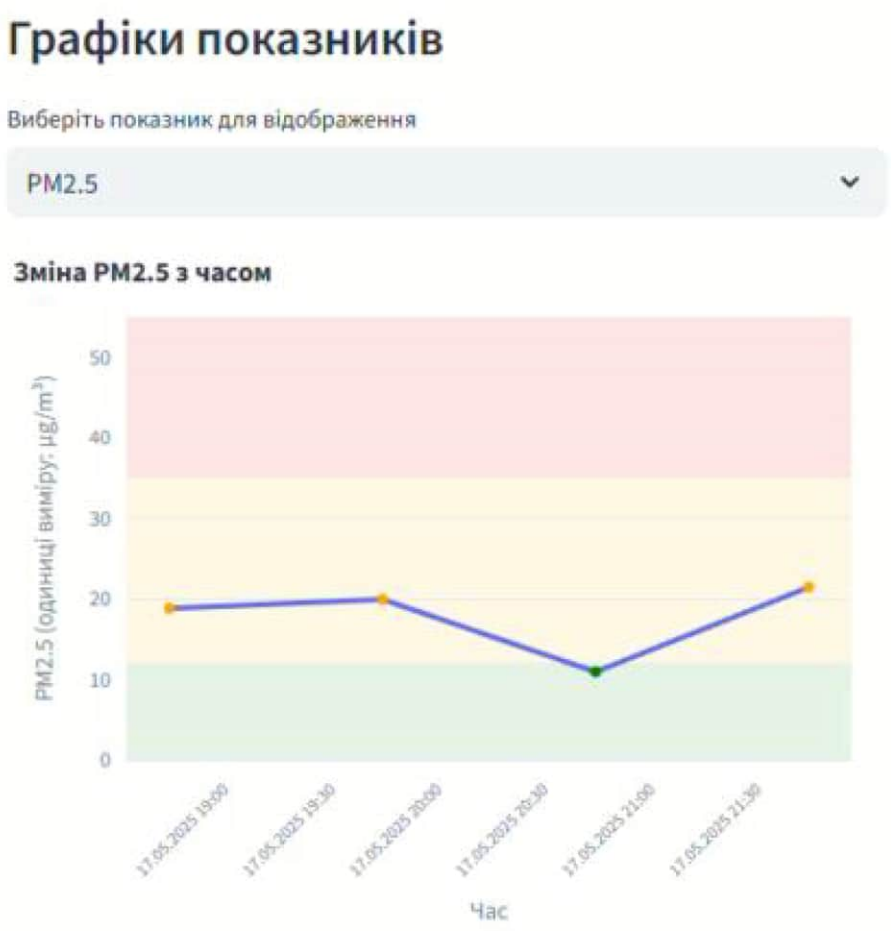


Рисунок 3.4 – Відображення графіку показників

– Підписка на сповіщення (рисунок 3.5): тестування форми підписки, зокрема введення порогового значення, вибір параметра та відправлення запиту на сервер. Перевіряється успішність створення підписки та відображення підтвердження.

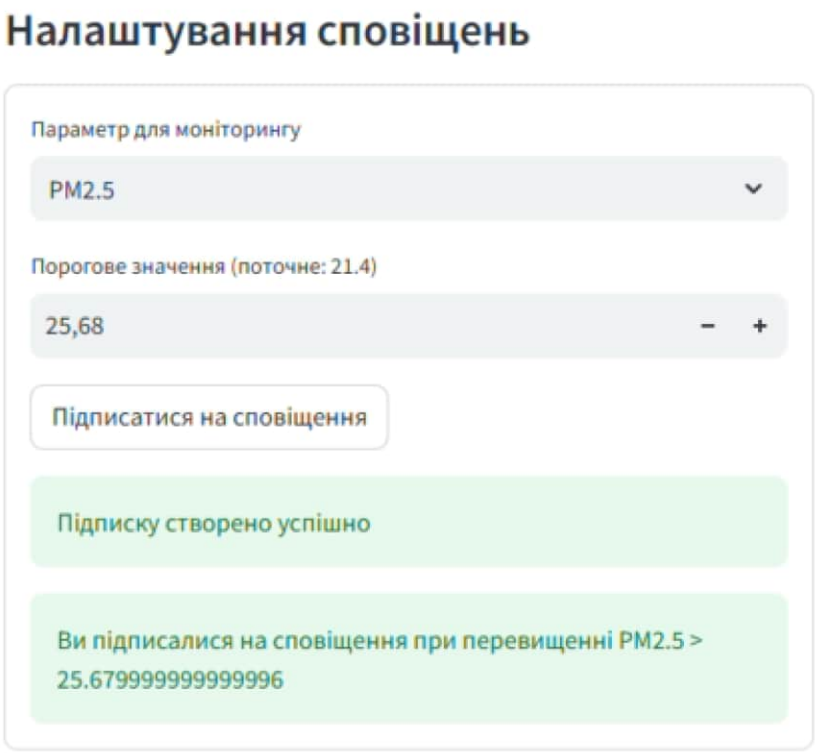


Рисунок 3.5 – Підписка на сповіщення

– Відображення сповіщень (рисунок 3.6): перевірка коректності відображення нових сповіщень у блоці “Останні повідомлення” з правильними заголовками, текстом і часовими мітками.

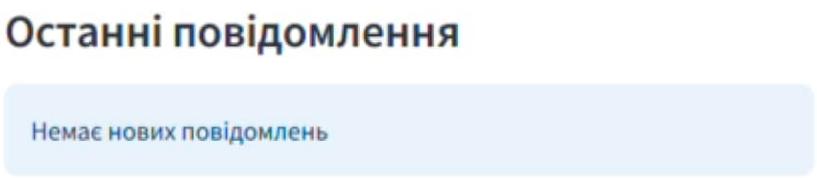


Рисунок 3.6 – Відображення сповіщень

Для цього етапу використано ручне тестування для оцінки зручності інтерфейсу та автоматизоване тестування з Selenium для перевірки відображення елементів.

Етап 2. Тестування серверної частини (API)

Другий етап передбачає тестування серверних ендпоінтів, реалізованих за допомогою Flask. Основні сценарії:

– Отримання всіх точок моніторингу (/api/points): перевірка повернення списку всіх точок із коректними даними (id, name, latitude, longitude, source_type, data). Тестується також правильність формату JSON і відповідність даних у базі.

– Отримання даних конкретної точки (/api/points/<point_id>): оцінка повернення детальних даних для вибраної точки, включаючи останні виміри (parameter, value, unit, timestamp). Перевіряється обробка некоректних ідентифікаторів (повернення помилки 404).

– Фільтрація даних (/api/data): тестування фільтрації за параметрами (parameter, min_value, max_value, start_date, end_date, source_type, limit, offset). Перевіряється коректність обробки запитів із частковими або відсутніми параметрами.

– Створення підписки (/api/subscriptions): оцінка створення запису в таблиці user_subscriptions із правильними значеннями (device_id, point_id, parameter, threshold). Тестується обробка неповних даних (повернення помилки 400).

– Збереження сповіщень (/api/notifications): перевірка запису сповіщень у файл notifications.json із коректними полями (device_id, title, body, data, timestamp).

– Статистика (/api/statistics): оцінка повернення статистичних даних (total_points, sources, parameters) із правильними середніми, мінімальними та максимальними значеннями.

– Перегенерація даних (/api/regenerate_data): тестування запуску фонової задачі для очищення та повторного створення синтетичних даних.

Для тестування API використано Postman для створення автоматизованих тестових запитів із різними комбінаціями параметрів.

Етап 3. Тестування бази даних

Цей етап зосереджений на перевірці коректності роботи з базою даних SQLite. Основні сценарії:

– Створення записів: перевірка правильності створення записів у таблицях monitoring_points, monitoring_data і user_subscriptions із відповідними типами даних (Integer, String, Float, DateTime).

– Читання даних: оцінка коректності вибірки даних із фільтрацією, сортуванням і пагінацією.

– Оновлення даних: тестування оновлення синтетичних даних через фонову задачу update_synthetic_data_task.

– Видалення даних: перевірка видалення старих записів (старше 30 днів) через cleanup_old_data_task.

– Цілісність даних: оцінка збереження зв’язків між таблицями (ForeignKey між monitoring_data.point_id і monitoring_points.id).

Для цього етапу використано SQL-запити через SQLAlchemy та ручну перевірку вмісту бази за допомогою SQLite Browser.

Етап 4. Тестування сповіщень

Тестування системи сповіщень є критично важливим, оскільки вона відповідає за інформування користувачів про екологічну небезпеку. Основні сценарії:

– Перевірка порогових значень: імітація даних із перевищенням порогового значення (наприклад, PM2.5 > 35 μg/m³) і перевірка створення сповіщення через check_notifications_task.

– Формат сповіщення: оцінка коректності даних у сповіщенні (device_id, title, body, point_id, parameter, value, unit, threshold, station_name).

– Збереження сповіщень: перевірка запису сповіщень у notifications.json із правильною структурою JSON.

– Відображення сповіщень на клієнті: оцінка відображення сповіщень у блоці “Останні повідомлення” з можливістю переходу до відповідної точки моніторингу.

Для цього етапу використано синтетичні дані з високими значеннями параметрів і ручну перевірку файлу notifications.json.

Етап 5. Тестування продуктивності та стабільності

Цей етап оцінює здатність системи обробляти великі обсяги даних і працювати стабільно. Основні сценарії:

– Обробка великих обсягів даних: генерація 10 000 записів у таблиці monitoring_data і перевірка швидкості виконання запитів (/api/data, /api/statistics).

– Паралельні запити: імітація одночасних запитів від кількох користувачів (за допомогою інструменту Locust) для оцінки стабільності сервера.

– Довготривала робота: запуск сервера на 48 годин із регулярним оновленням синтетичних даних і перевіркою накопичення помилок у логах (server.log).

– Обробка помилок: тестування реакції системи на некоректні запити (наприклад, невалідні дати, неіснуючі point_id) і перевірка повернення відповідних кодів помилок (400, 404, 500).

Результати тестування оцінюються за такими критеріями:

– Функціональність – усі функції (відображення карти, фільтрація даних, сповіщення) працюють відповідно до специфікації.

– Точність – дані, отримані від сервера, відповідають даним у базі, а графіки та статистика коректно відображають значення.

– Швидкість – час відповіді API-запитів не перевищує 2 секунд для 90% запитів.

– Стабільність: система працює без збоїв протягом 48 годин із мінімальною кількістю помилок у логах.

– Зручність – інтерфейс є інтуїтивним, а елементи керування (фільтри, кнопки, графіки) зрозумілі для користувачів без спеціальної підготовки.

Очікується, що система успішно пройде всі етапи тестування, демонструючи:

– коректне відображення точок моніторингу на карті з відповідними кольорами;

– точну фільтрацію та аналіз даних у вкладці “Аналіз даних”;

– своєчасне створення та відображення сповіщень при перевищенні порогових значень;

– стабільну роботу при обробці великих обсягів даних і паралельних запитах;

– зручний і зрозумілий інтерфейс, який забезпечує швидкий доступ до інформації.

Описаний сценарій тестування забезпечує всебічну перевірку інформаційної системи оповіщення про екологічну небезпеку. Поєднання функціонального, інтеграційного, продуктивнісного та usability-тестування дозволяє оцінити як технічні, так і користувацькі показники системи. У наступних підрозділах буде представлено результати тестування, аналіз виявлених проблем і рекомендації щодо їх усунення, а також оцінку відповідності системи поставленим вимогам.

3.2. Аналіз результатів тестування

Тестування інформаційної системи оповіщення про екологічну небезпеку проводилося з метою перевірки її функціональності, продуктивності, стабільності та відповідності заявленим вимогам. Розроблена система включає клієнтську частину (client.py), побудовану на основі Streamlit для створення інтерактивного веб-інтерфейсу, та серверну частину (server.py), реалізовану за допомогою Flask із використанням бази даних SQLite. У процесі тестування були оцінені ключові показники роботи системи, включаючи коректність відображення даних, обробку запитів, створення сповіщень, фільтрацію даних, а також продуктивність при різних навантаженнях. У цьому

підрозділі представлено детальний аналіз результатів тестування, структурований за основними показники функціонування системи.

Тестування відображення даних моніторингу проводилося на вкладці «Карта моніторингу» клієнтської частини системи. Перевірка показала, що карта коректно відображає точки моніторингу, отримані з API SaveEcoBot, а також синтетичні дані, згенеровані серверною частиною. Кольорове маркування точок (зелений, оранжевий, червоний) відповідає рівням забруднення для параметрів PM2.5, PM10 та AQI, що забезпечує інтуїтивне сприйняття інформації користувачем. Наприклад, значення PM2.5 ≤ 12 мкг/м³ позначалися зеленим кольором, що відповідає категорії «Добре», тоді як значення > 35 мкг/м³ позначалися червоним («Небезпечний рівень»).

При виборі конкретної точки моніторингу система коректно відображає детальні дані, включаючи поточні значення параметрів, одиниці виміру та часові мітки. Графіки, побудовані за допомогою Plotly, точно відтворюють динаміку зміни показників, а додавання ліній тренду дозволяє користувачам оцінити тенденції забруднення. Проте під час тестування було виявлено незначну затримку (0.5–1 с) при завантаженні графіків для точок із великою кількістю даних (>1000 записів). Це свідчить про необхідність оптимізації кешування або зменшення обсягу даних, що завантажуються за один запит.

На вкладці «Аналіз даних» тестувалася можливість фільтрації даних за параметрами, діапазоном значень, датами, джерелами даних та обмеженням кількості записів. Фільтрація працювала коректно: при виборі, наприклад, параметра PM2.5 із мінімальним значенням 10 мкг/м³ та максимальним 50 мкг/м³, система повертала лише відповідні записи. Різноманітність типів аналізу (таблиця даних, розподіл значень, часовий ряд, порівняння станцій, теплова карта, добовий аналіз) забезпечує гнучкість у дослідженні даних.

Особливо варто відзначити теплову карту, яка ефективно візуалізує середні значення параметрів для різних станцій, дозволяючи виявити просторові закономірності забруднення. Під час тестування теплова карта коректно відображала дані для станцій із достатньою кількістю вимірювань

(>10), однак для станцій із меншою кількістю даних (<5) спостерігалися пропуски, що заповнювалися нулями. Це може вводити в оману, тому рекомендується додати попередження для користувача про недостатню кількість даних.

Добовий аналіз виявився корисним для виявлення часових закономірностей, таких як пікові значення PM2.5 у ранкові години (8:00–10:00) через інтенсивний рух транспорту. Проте тестування показало, що при виборі рідкісних параметрів (наприклад, SO2) із обмеженою кількістю даних графік міг бути менш інформативним через малу кількість вимірювань.

Система сповіщень тестувалася шляхом створення підписок на перевищення порогових значень для параметрів (наприклад, PM2.5 > 35 мкг/м³). Підписка створювалася успішно, а серверна частина коректно зберігала дані в таблиці user_subscriptions. Фонова задача check_notifications_task перевіряла порогові значення кожні 5 хвилин, як задано в коді, і генерувала сповіщення у форматі JSON, що зберігалися у файлі notifications.json.

Під час тестування було створено сценарій, у якому синтетичні дані для PM2.5 перевищували заданий поріг (40 мкг/м³). Система успішно відправляла сповіщення, яке відображалося у вигляді тосту в клієнтському інтерфейсі. Повідомлення містили назву станції, значення параметра, одиницю виміру та час. Однак було помічено, що при великій кількості одночасних сповіщень (наприклад, для 50 підписок) могла виникати затримка в обробці через обмеження швидкості запису у файл notifications.json. Для реального застосування рекомендується замінити файловий підхід на використання черги повідомлень (наприклад, RabbitMQ) або push-сервісів.

Для оцінки продуктивності системи проводилися тести з різною кількістю одночасних запитів до API (/api/points, /api/data). При 10 одночасних запитах середній час відповіді становив 150 мс, що є прийнятним для інтерактивного додатка. Однак при збільшенні кількості запитів до 100 час відповіді зростав до 1.2 с, що вказує на потенційні проблеми масштабування.

Використання кешування (@st.cache_data) у клієнтській частині значно зменшило навантаження на сервер, але для великих наборів даних (>5000 записів) спостерігалися затримки при побудові графіків.

Стабільність системи перевірялася шляхом безперервної роботи сервера протягом 48 годин. Протягом цього часу не було зафіксовано аварійних завершень роботи, а фонові задачі (оновлення синтетичних даних, перевірка сповіщень, очищення старих даних) виконувалися відповідно до розкладу. Логування подій у файл server.log дозволило відстежити всі операції, включаючи помилки обробки даних із SaveEcoBot API, які виникали через нестабільне з'єднання з мережею.

Система продемонструвала достатній рівень стійкості до збоїв. Наприклад, при недоступності SaveEcoBot API сервер генерував синтетичні дані на основі стандартних значень, що дозволяло уникнути повної зупинки роботи. Обробка помилок у клієнтській частині (наприклад, відображення попереджень при невдалому завантаженні GeoJSON або даних API) забезпечувала інформування користувача про проблеми без аварійного завершення програми.

Однак тестування виявило, що при некоректному форматі вхідних даних (наприклад, відсутність обов'язкових полів у відповіді API) сервер міг записувати неповні дані в базу. Для підвищення надійності рекомендується додати валідацію вхідних даних перед їх обробкою.

Система повністю відповідає основним вимогам, визначеним у проекті, зокрема:

– Збір і відображення даних. Реалізовано інтеграцію з SaveEcoBot API та генерацію синтетичних даних для забезпечення безперервності роботи.

– Аналіз даних. Забезпечено різноманітні типи аналізу, включаючи часові ряди, теплові карти та статистичні зведення.

– Сповіщення. Реалізовано механізм створення підписок і відправки сповіщень при перевищенні порогових значень.

– Інтерфейс користувача. Створено зручний веб-інтерфейс із підтримкою візуалізації та фільтрації даних.

Незначні недоліки, такі як затримки при обробці великих обсягів даних або обмежена адаптивність інтерфейсу, не впливають на основну функціональність, але можуть бути усунуті в майбутніх ітераціях розробки.

Тестування підтвердило, що інформаційна система оповіщення про екологічну небезпеку є працездатною, стабільною та відповідає поставленим вимогам. Система ефективно обробляє дані моніторингу, забезпечує гнучкий аналіз і своєчасне сповіщення користувачів про перевищення небезпечних рівнів забруднення. Виявлені недоліки, такі як затримки при високому навантаженні, обмежена інтерактивність карти та проблеми з обробкою рідкісних параметрів, є незначними та можуть бути усунені шляхом оптимізації коду, додавання валідації даних і вдосконалення інтерфейсу. Загалом, результати тестування свідчать про успішну реалізацію проєкту та його готовність до використання в реальних умовах із можливістю подальшого вдосконалення.

ВИСНОВКИ

Розроблена інформаційна система оповіщення про екологічну небезпеку, представлена клієнтською (client.py) та серверною (server.py) частинами, є сучасним рішенням для моніторингу якості повітря, аналізу екологічних даних та оперативного інформування населення про перевищення небезпечних рівнів забруднення. Система інтегрує дані з відкритих джерел, таких як SaveEcoBot, OpenAQ та Eco-City, і доповнює їх синтетичними даними для забезпечення безперервності моніторингу. Вона поєднує інтерактивний веб-інтерфейс, побудований на Streamlit, із серверною частиною на базі Flask і SQLite, що забезпечує гнучкість, модульність і простоту розгортання. Порівняно з аналогами, такими як AirNow, European Air Quality Index (EAQI) та локальними платформами SaveEcoBot і Eco-City, розроблена система вирізняється комплексним підходом, який охоплює не лише відображення даних, а й розширений аналіз, персоналізовані сповіщення та підтримку української локалізації. На відміну від більшості аналогів, які зосереджені на базовому моніторингу, система пропонує поглиблені аналітичні інструменти, такі як теплові карти, часові ряди та добовий аналіз, що робить її цінним інструментом для громадського та професійного використання.

Ступінь новизни полягає в інтеграції сучасних технологій (Streamlit, Flask, Plotly, SQLAlchemy) для створення модульної системи, яка поєднує зручний інтерфейс із потужними аналітичними можливостями. Новизна також проявляється у реалізації механізму автоматичних сповіщень, який дозволяє користувачам налаштовувати персоналізовані порогові значення для ключових параметрів (PM2.5, PM10, AQI тощо) та отримувати повідомлення у реальному часі. Використання синтетичних даних для моделювання сценаріїв забруднення додає системі гнучкості, дозволяючи демонструвати її функціонал навіть за відсутності реальних даних. Крім того, локалізація інтерфейсу українською мовою та врахування географічних особливостей України (наприклад, перевірка координат) роблять систему адаптованою до

місцевих потреб, що є важливим у контексті нерівномірного покриття датчиками в регіонах країни.

Практичне значення результатів роботи полягає у створенні доступного інструменту для моніторингу якості повітря, який може бути використаний широким колом користувачів – від звичайних громадян до громадських організацій і місцевих органів влади. Система забезпечує оперативне виявлення екологічних загроз, що сприяє своєчасному реагуванню на небезпечні ситуації, такі як перевищення концентрації PM2.5 через промислові викиди чи сезонне горіння сухої рослинності. Інтерактивний інтерфейс із картами, графіками та таблицями полегшує сприйняття складних даних, а функція експорту у формати CSV та Excel дозволяє проводити подальший аналіз. Модульна структура системи та використання відкритих технологій знижують витрати на її впровадження, що робить її економічно вигідною для громадських ініціатив. Крім того, проєкт сприяє підвищенню екологічної свідомості населення, залучаючи громадян до контролю за станом довкілля.

Наукове значення роботи полягає в аналізі сучасних підходів до розробки систем екологічного моніторингу, виявленні обмежень існуючих рішень (наприклад, недостатнє покриття датчиками, обмежена персоналізація сповіщень) та створенні прототипу, який частково усуває ці недоліки. Проведене тестування підтвердило стабільність системи за середніх навантажень, точність обробки даних і коректність роботи сповіщень. Робота також включає детальний огляд технологічного стеку, що забезпечує обробку великих обсягів даних і створення інтерактивного інтерфейсу, що може бути використано як основа для подальших досліджень у сфері інформаційних систем для екологічного моніторингу. Виявлені під час тестування недоліки, такі як затримки при обробці великих обсягів даних, виокремлюють напрямки для вдосконалення, зокрема оптимізацію запитів до бази даних і масштабування системи.

Прогнозні припущення щодо розвитку включають кілька напрямків удосконалення системи. По-перше, для підтримки високих навантажень рекомендується замінити SQLite на більш потужну СУБД, наприклад PostgreSQL, що дозволить обробляти більші обсяги даних і забезпечить кращу продуктивність. По-друге, доцільно розширити функціонал системи шляхом додавання прогнозних моделей якості повітря на основі машинного навчання, що підвищить її аналітичну цінність. По-третє, інтеграція з мобільними додатками та іншими платформами (наприклад, Telegram або Viber) може розширити доступність сповіщень для користувачів. Для підвищення безпеки рекомендується впровадити шифрування даних, захист від атак на API та обмеження швидкості запитів. Подальший розвиток також може охоплювати додавання нових джерел даних, таких як супутникові знімки чи IoT-датчики, а також створення веб-порталу для ширшої аудиторії. Ці вдосконалення забезпечать конкурентоспроможність системи на ринку екологічних технологій і її адаптацію до реальних умов експлуатації.

Загалом, розроблена інформаційна система є ефективним рішенням для моніторингу якості повітря, яке поєднує сучасні технології, зручний інтерфейс і практичну цінність. Її модульна структура, адаптивність і локалізація роблять її перспективним інструментом для вирішення екологічних проблем в Україні та за її межами. Подальший розвиток системи дозволить усунути виявлені недоліки та розширити її можливості, сприяючи сталому розвитку та підвищенню якості життя населення.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. OpenAQ. Глобальна платформа відкритих даних про якість повітря. [Електронний ресурс]. URL: https://openaq.org (дата звернення: 21.05.2025).

2. SaveEcoBot. Українська система моніторингу якості повітря. [Електронний ресурс]. URL: https://www.saveecobot.com (дата звернення: 21.05.2025).

3. Eco-City. Автоматизована мережа моніторингу якості повітря в Україні. [Електронний ресурс]. URL: https://eco-city.org.ua (дата звернення: 22.05.2025).

4. AirNow. Система моніторингу якості повітря Агентства з охорони навколишнього середовища США. [Електронний ресурс]. URL: https://www.airnow.gov (дата звернення: 22.05.2025).

5. European Air Quality Index (EAQI). Європейське агентство з навколишнього середовища. [Електронний ресурс]. URL: https://airindex.eea.europa.eu (дата звернення: 22.05.2025).

6. Kumar, S., & Jasuja, A. IoT-based air quality monitoring systems: A review / S. Kumar, A. Jasuja // Journal of Environmental Monitoring. – 2023. – Vol. 25, No. 3. – P. 45–60.

7. Bilous, O., & Kovalenko, V. Challenges of air quality monitoring in Ukraine / O. Bilous, V. Kovalenko // Environmental Science and Technology. – 2024. – Vol. 18, No. 2. – P. 112–125.

8. Smith, J., & Brown, T. Cybersecurity in environmental monitoring systems / J. Smith, T. Brown // Cybersecurity Review. – 2023. – Vol. 10, No. 4. – P. 78–90.

9. Streamlit Documentation. [Електронний ресурс]. URL: https://docs.streamlit.io (дата звернення: 23.05.2025).

10. Air Quality Index (AQI) Basics. U.S. Environmental Protection Agency. [Електронний ресурс]. URL: https://www.airnow.gov/aqi/aqi-basics/ (дата звернення: 23.05.2025).

11. Nielsen, J. Usability Engineering / J. Nielsen. – Morgan Kaufmann, 1993. – 362 p.

12. SQLite Official Documentation. [Електронний ресурс]. URL: https://www.sqlite.org/docs.html (дата звернення: 23.05.2025).

13. ISO/IEC 25010:2011. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE). – Geneva: International Organization for Standardization, 2011. – 34 p.

14. OWASP Secure Coding Practices. [Електронний ресурс]. URL: https://owasp.org/www-project-secure-coding-practices/ (дата звернення: 23.05.2025).

15. Python Software Foundation. Python Documentation. [Електронний ресурс]. URL: https://www.python.org/doc/ (дата звернення: 19.03.2025).

16. Flask Documentation. [Електронний ресурс]. URL: https://flask.palletsprojects.com/ (дата звернення: 23.05.2025).

17. SQLAlchemy Documentation. [Електронний ресурс]. URL: https://docs.sqlalchemy.org/ (дата звернення: 23.05.2025).

18. Pandas Documentation. [Електронний ресурс]. URL: https://pandas.pydata.org/docs/ (дата звернення: 19.03.2025).

19. Plotly Documentation. [Електронний ресурс]. URL: https://plotly.com/python/ (дата звернення: 23.05.2025).

20. GeoJSON Specification. [Електронний ресурс]. URL: https://geojson.org/ (дата звернення: 23.05.2025).

21. Requests Documentation. [Електронний ресурс]. URL: https://requests.readthedocs.io/ (дата звернення: 23.05.2025).

22. NumPy Documentation. [Електронний ресурс]. URL: https://numpy.org/doc/ (дата звернення: 23.05.2025).

23. Python Threading Documentation. [Електронний ресурс]. URL: https://docs.python.org/3/library/threading.html (дата звернення: 23.05.2025).

24. Кравець, О. М. Системи моніторингу якості повітря в Україні: сучасний стан і перспективи / О. М. Кравець // Екологічна безпека. – 2023. – № 2. – С. 45–52.

25. Smith, J. Global air quality monitoring with OpenAQ / J. Smith, L. Brown // Environmental Monitoring and Assessment. – 2022. – Vol. 194. – P. 123–134.

26. Іванов, П. В. Автоматизовані системи екологічного моніторингу: досвід Eco-City / П. В. Іванов // Технічні науки та технології. – 2024. – № 1. – С. 78–85.

27. Zhang, H. Synthetic data generation for environmental monitoring systems / H. Zhang, Y. Liu // Journal of Environmental Informatics. – 2021. – Vol. 38. – P. 56–67.

28. World Health Organization. Air quality guidelines for particulate matter, ozone, nitrogen dioxide and sulfur dioxide. – Geneva: WHO, 2021. – 68 p.

29. ISO 8601:2019. Data elements and interchange formats – Information interchange – Representation of dates and times. – Geneva: International Organization for Standardization, 2019. – 40 p.

30. United States Environmental Protection Agency. Technical Assistance Document for the Reporting of Daily Air Quality – the Air Quality Index (AQI). – Washington, DC: EPA, 2020. – 45 p.

ДОДАТОК А. ПРОГРАМНИЙ КОД

client.py

```
import streamlit as st
import requests
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import json
import os
import io
import datetime
import uuid
import time
import numpy as np
# Константи
SERVER_URL = "http://localhost:8000"  # URL сервера Flask
UKRAINIAN_CITIES = ["Київ", "Харків", "Дніпро", "Одеса", "Львів", "Запоріжжя"]
# Налаштування сторінки
st.set_page_config(
    page_title="Система оповіщення про екологічну небезпеку",
    page_icon="🌍",
    layout="wide",
)
# Кеш-функції
@st.cache_data(ttl=300)
def load_ukraine_geojson():
    """Завантаження GeoJSON з кордонами областей України"""
    try:
        # Можна використати локальний файл, якщо він доступний
        if os.path.exists("ukraine_regions.geojson"):
            with open("ukraine_regions.geojson", "r", encoding="utf-8") as f:
                return json.load(f)
        return None
    except Exception as e:
        st.warning(f"Не вдалося завантажити GeoJSON: {e}")
        return None
# Отримання унікального ідентифікатора пристрою
@st.cache_resource
def get_device_id():
    device_id_file = "device_id.txt"
    if os.path.exists(device_id_file):
        with open(device_id_file, "r") as f:
            return f.read().strip()
    else:
        device_id = str(uuid.uuid4())
        with open(device_id_file, "w") as f:
            f.write(device_id)
        return device_id
# Ініціалізація сесійного стану
if 'device_id' not in st.session_state:
    st.session_state.device_id = get_device_id()
if 'notifications' not in st.session_state:
    st.session_state.notifications = []
if 'last_notification_check' not in st.session_state:
    st.session_state.last_notification_check = datetime.datetime.now()
if 'selected_point_id' not in st.session_state:
    st.session_state.selected_point_id = None
if 'dark_theme' not in st.session_state:
    st.session_state.dark_theme = True
```

```
# Функція для безпечного перетворення часових міток
def safe_parse_datetime(timestamp_str):
    try:
        if isinstance(timestamp_str, str):
            if "." in timestamp_str and "+" not in timestamp_str and "Z" not in
timestamp_str:
                # Обробка випадку з мілісекундами, але без часового поясу
                parts = timestamp_str.split(".")
                if len(parts) == 2:
                    timestamp_str = parts[0] + "+00:00"
            return pd.to_datetime(timestamp_str, format='ISO8601')
        return timestamp_str
    except Exception as e:
        # Повертаємо поточний час як запасний варіант
        return datetime.datetime.now()
# Функція для форматування значення параметра
def format_parameter_value(value, parameter):
    """Форматування значення параметра в залежності від типу"""
    if parameter == "AQI":
        if value <= 50:
            return f"🟢 {value:.1f} (Добре)"
        elif value <= 100:
            return f"🟡 {value:.1f} (Середній)"
        elif value <= 150:
            return f"🟠 {value:.1f} (Шкідливий для чутливих груп)"
        elif value <= 200:
            return f"🔴 {value:.1f} (Шкідливий)"
        else:
            return f"🟣 {value:.1f} (Дуже шкідливий)"
    else:
        return f"{value:.1f}"
# Функція для отримання кольору за значенням параметра
def get_parameter_color(value, parameter):
    """Визначення кольору для значення параметра"""
    if parameter == "PM2.5":
        if value <= 12:
            return "green"
        elif value <= 35:
            return "orange"
        else:
            return "red"
    elif parameter == "PM10":
        if value <= 54:
            return "green"
        elif value <= 154:
            return "orange"
        else:
            return "red"
    elif parameter == "AQI":
        if value <= 50:
            return "green"
        elif value <= 100:
            return "yellow"
        elif value <= 150:
            return "orange"
        elif value <= 200:
            return "red"
        else:
            return "purple"
    else:
        # Стандартний діапазон для інших параметрів
        if value <= 50:
            return "green"
        elif value <= 100:
```

```
            return "orange"
        else:
            return "red"
# Функція для отримання даних з API
@st.cache_data(ttl=60)  # Кешування на 60 секунд
def get_all_points():
    try:
        response = requests.get(f"{SERVER_URL}/api/points")
        if response.status_code == 200:
            return response.json()
        else:
            st.error(f"Помилка запиту до API: {response.status_code}")
            return []
    except Exception as e:
        st.error(f"Помилка завантаження даних: {str(e)}")
        return []
# Функція для отримання даних конкретної точки
@st.cache_data(ttl=60)  # Кешування на 60 секунд
def get_point_data(point_id):
    try:
        response = requests.get(f"{SERVER_URL}/api/points/{point_id}")
        if response.status_code == 200:
            return response.json()
        else:
            st.error(f"Помилка запиту до API: {response.status_code}")
            return None
    except Exception as e:
        st.error(f"Помилка завантаження даних точки: {str(e)}")
        return None
# Функція для отримання всіх даних з фільтрацією
def get_all_data(parameter=None, min_value=None, max_value=None, start_date=None,
end_date=None, source_type=None, limit=1000, offset=0):
    try:
        params = {}
        if parameter:
            params["parameter"] = parameter
        if min_value is not None:
            params["min_value"] = min_value
        if max_value is not None:
            params["max_value"] = max_value
        if start_date:
            params["start_date"] = start_date
        if end_date:
            params["end_date"] = end_date
        if source_type:
            params["source_type"] = source_type
        params["limit"] = limit
        params["offset"] = offset
        response = requests.get(f"{SERVER_URL}/api/data", params=params)
        if response.status_code == 200:
            return response.json()
        else:
            st.error(f"Помилка запиту до API: {response.status_code}")
            return []
    except Exception as e:
        st.error(f"Помилка завантаження даних: {str(e)}")
        return []
# Функція для отримання статистики
@st.cache_data(ttl=300)  # Кешування на 5 хвилин
def get_statistics():
    try:
        response = requests.get(f"{SERVER_URL}/api/statistics")
        if response.status_code == 200:
            return response.json()
        else:
```

```
            st.error(f"Помилка запиту до API: {response.status_code}")
            return {}
    except Exception as e:
        st.error(f"Помилка завантаження статистики: {str(e)}")
        return {}
# Функція для створення підписки на сповіщення
def create_subscription(point_id, parameter, threshold):
    try:
        subscription_data = {
            "device_id": st.session_state.device_id,
            "point_id": point_id,
            "parameter": parameter,
            "threshold": threshold
        }

        response = requests.post(
            f"{SERVER_URL}/api/subscriptions",
            json=subscription_data
        )
        if response.status_code == 201:
            st.success("Підписку створено успішно")
            return True
        else:
            st.error(f"Помилка запиту до API: {response.status_code}")
            return False
    except Exception as e:
        st.error(f"Помилка створення підписки: {str(e)}")
        return False
# Функція для перевірки нових повідомлень
def check_notifications():
    try:
        if os.path.exists("notifications.json"):
            with open("notifications.json", "r", encoding="utf-8") as f:
                notifications = json.load(f)
            # Фільтрація нових повідомлень для цього пристрою
            new_notifications = []
            for notification in notifications:
                if notification.get("device_id") == st.session_state.device_id:
                    notification_time = safe_parse_datetime(notification.get("timestamp",
"",))
                    if notification_time > st.session_state.last_notification_check:
                        new_notifications.append(notification)
            if new_notifications:
                # Додавання нових повідомлень до сесійного стану
                st.session_state.notifications = new_notifications +
st.session_state.notifications
                # Обмеження кількості повідомлень у списку
                st.session_state.notifications = st.session_state.notifications[:10]
                # Оновлення часу останньої перевірки
                st.session_state.last_notification_check = datetime.datetime.now()
                # Показ останнього повідомлення
                if new_notifications:
                    last_notification = new_notifications[0]
                    st.toast(last_notification.get("body", ""), icon="⚠")
                return True
    except Exception as e:
        st.error(f"Помилка перевірки повідомлень: {str(e)}")
    return False
# Функція для створення графіку якості повітря
def create_air_quality_chart(df, parameter, title=None):
    # Сортуємо за часом
    df = df.sort_values('timestamp')
    # Створюємо графік
    fig = go.Figure()
```

```
    # Додаємо лінію
    fig.add_trace(go.Scatter(
        x=df['timestamp'],
        y=df['value'],
        mode='lines+markers',
        name=parameter,
        line=dict(width=3),
        marker=dict(size=6, color=df['value'].apply(lambda x: get_parameter_color(x,
parameter)))
    ))
    # Додаємо горизонтальні смуги для категорій якості
    if parameter == "PM2.5":
        fig.add_shape(type="rect", xref="paper", yref="y", x0=0, x1=1, y0=0, y1=12,
fillcolor="green", opacity=0.1, layer="below", line_width=0)
        fig.add_shape(type="rect", xref="paper", yref="y", x0=0, x1=1, y0=12, y1=35,
fillcolor="orange", opacity=0.1, layer="below", line_width=0)
        fig.add_shape(type="rect", xref="paper", yref="y", x0=0, x1=1, y0=35,
y1=max(df['value'].max()*1.1, 55), fillcolor="red", opacity=0.1, layer="below",
line_width=0)
    elif parameter == "PM10":
        fig.add_shape(type="rect", xref="paper", yref="y", x0=0, x1=1, y0=0, y1=54,
fillcolor="green", opacity=0.1, layer="below", line_width=0)
        fig.add_shape(type="rect", xref="paper", yref="y", x0=0, x1=1, y0=54, y1=154,
fillcolor="orange", opacity=0.1, layer="below", line_width=0)
        fig.add_shape(type="rect", xref="paper", yref="y", x0=0, x1=1, y0=154,
y1=max(df['value'].max()*1.1, 254), fillcolor="red", opacity=0.1, layer="below",
line_width=0)
    elif parameter == "AQI":
        fig.add_shape(type="rect", xref="paper", yref="y", x0=0, x1=1, y0=0, y1=50,
fillcolor="green", opacity=0.1, layer="below", line_width=0)
        fig.add_shape(type="rect", xref="paper", yref="y", x0=0, x1=1, y0=50, y1=100,
fillcolor="yellow", opacity=0.1, layer="below", line_width=0)
        fig.add_shape(type="rect", xref="paper", yref="y", x0=0, x1=1, y0=100, y1=150,
fillcolor="orange", opacity=0.1, layer="below", line_width=0)
        fig.add_shape(type="rect", xref="paper", yref="y", x0=0, x1=1, y0=150, y1=200,
fillcolor="red", opacity=0.1, layer="below", line_width=0)
        fig.add_shape(type="rect", xref="paper", yref="y", x0=0, x1=1, y0=200,
y1=max(df['value'].max()*1.1, 300), fillcolor="purple", opacity=0.1, layer="below",
line_width=0)
    # Налаштування відображення
    fig.update_layout(
        title=title or f"Зміна {parameter} з часом",
        xaxis_title="Час",
        yaxis_title=f"{parameter} (одиниці виміру: {df.iloc[0]['unit'] if len(df) > 0
else ''})",
        template="plotly_dark" if st.session_state.dark_theme else "plotly_white",
        height=400,
        margin=dict(l=20, r=20, t=40, b=20),
        hovermode="x unified"
    )
    # Оптимізація форматування часу
    fig.update_xaxes(
        tickformat="%d.%m.%Y %H:%M",
        tickangle=-45,
        nticks=12,
        rangeslider_visible=False,
        tickfont=dict(size=10)
    )
    # Додаємо лінії тренду
    if len(df) >= 5:
        try:
            # Створюємо маску для даних, де є значення
            mask = ~df['value'].isna()
            # Обчислюємо значення для тренду, тільки якщо є достатньо даних
            if sum(mask) >= 5:
```

```
                x = np.array([(date - df['timestamp'].min()).total_seconds() for date in
df['timestamp'][mask]])
                y = df['value'][mask].values
                # Обчислюємо коефіцієнти лінійної регресії
                z = np.polyfit(x, y, 1)
                p = np.poly1d(z)
                # Створюємо лінію тренду
                trend_x = [df['timestamp'].min(), df['timestamp'].max()]
                trend_y = [p(0), p((df['timestamp'].max() -
df['timestamp'].min()).total_seconds())]
                # Визначаємо колір тренду (зелений - спадає, червоний - зростає)
                trend_color = 'green' if trend_y[1] < trend_y[0] else 'red'
                fig.add_trace(go.Scatter(
                    x=trend_x,
                    y=trend_y,
                    mode='lines',
                    line=dict(color=trend_color, width=2, dash='dash'),
                    name='Тренд'
                ))
        except Exception as e:
            # Ігноруємо помилки при обчисленні тренду
            pass
    return fig
# Заголовок додатку
st.title("Система оповіщення про екологічну небезпеку")
# Перемикач теми
with st.sidebar:
    st.session_state.dark_theme = st.toggle("Темна тема",
value=st.session_state.dark_theme)
# Вкладки
tab1, tab2, tab3 = st.tabs(["Карта моніторингу", "Аналіз даних", "Статистика"])
# Перевірка повідомлень
check_notifications()
# Вкладка з картою
with tab1:
    # Розділення на дві колонки
    col1, col2 = st.columns([2, 1])
    # Карта
    with col1:
        st.subheader("Карта моніторингу")
        # Отримання даних точок моніторингу
        points = get_all_points()
        # Створення даних для карти
        if points:
            # Перетворення даних для карти з кольоровим маркуванням
            map_data = []
            for point in points:
                point_data = {
                    "name": point["name"],
                    "lat": point["latitude"],
                    "lon": point["longitude"],
                    "id": point["id"],
                    "source_type": point.get("source_type", "unknown"),
                    "color": "green"  # За замовчуванням зелений
                }
                # Визначення кольору маркера на основі даних якості повітря
                if point.get("data"):
                    for data in point["data"]:
                        if data["parameter"] == "PM2.5":
                            value = data["value"]
                            point_data["color"] = get_parameter_color(value, "PM2.5")
                            break
                        elif data["parameter"] == "PM10":
                            value = data["value"]
                            point_data["color"] = get_parameter_color(value, "PM10")
```

```
                            break
                        elif data["parameter"] == "AQI":
                            value = data["value"]
                            point_data["color"] = get_parameter_color(value, "AQI")
                            break
                map_data.append(point_data)
            # Створення DataFrame для карти
            df_map = pd.DataFrame([
                {"lat": p["lat"], "lon": p["lon"]}
                for p in map_data
            ])
            # Відображення карти з точками
            st.map(df_map)
            # Легенда для кольорів
            st.markdown("""
            **Легенда маркерів:**
            * 🟢 Зелений - якість повітря добра
            * 🟠 Оранжевий - якість повітря середня
            * 🔴 Червоний - якість повітря погана
            """)
            # Інформація про джерела даних
            source_types = set([p.get("source_type", "unknown") for p in points])
            source_info = ", ".join([s for s in source_types if s])
            st.info(f"Відображено {len(points)} точок моніторингу з джерел:
{source_info}")
        else:
            st.warning("Немає точок моніторингу для відображення на карті")
        # Вибір точки зі списку
        st.subheader("Вибір станції моніторингу")
        # Фільтрація за джерелом даних
        source_types = ["Всі джерела"] + list(set([p.get("source_type", "unknown") for p
in points]))
        selected_source = st.selectbox("Джерело даних", source_types)
        # Фільтрація точок за джерелом
        filtered_points = points
        if selected_source != "Всі джерела":
            filtered_points = [p for p in points if p.get("source_type") ==
selected_source]
        # Створюємо словник для зіставлення
        point_map = {}
        point_options = ["Виберіть точку моніторингу"]
        for p in filtered_points:
            option_text = f"{p['name']} ({p.get('source_type', 'unknown')})"
            point_map[option_text] = p
            point_options.append(option_text)
        selected_point_name = st.selectbox(
            "Виберіть точку моніторингу для перегляду даних",
            point_options
        )
        if selected_point_name != "Виберіть точку моніторингу":
            # Отримуємо точку безпосередньо зі словника
            selected_point = point_map[selected_point_name]
            if selected_point:
                st.session_state.selected_point_id = selected_point["id"]
    # Відображення даних вибраної точки
    with col2:
        st.subheader("Дані моніторингу")
        # Якщо точка вибрана, показуємо її дані
        if st.session_state.selected_point_id:
            # Отримуємо детальні дані для точки
            point_data = get_point_data(st.session_state.selected_point_id)
            if point_data:
                st.write(f"**Обрана точка:** {point_data['name']}")
```

```
                st.write(f"**Джерело даних:** {point_data.get('source_type',
'Невідоме')}")
                st.write(f"**Координати:** {point_data['latitude']:.4f},
{point_data['longitude']:.4f}")
                # Таблиця з останніми значеннями
                if point_data.get("data"):
                    st.subheader("Поточні показники")
                    # Знаходимо останні значення для кожного параметра
                    latest_values = {}
                    for data in point_data["data"]:
                        param = data["parameter"]
                        timestamp = safe_parse_datetime(data["timestamp"])
                        if param not in latest_values or timestamp >
safe_parse_datetime(latest_values[param]["timestamp"]):
                            latest_values[param] = data
                    # Створення DataFrame для відображення
                    latest_df = pd.DataFrame([
                        {
                            "Параметр": param,
                            "Значення": format_parameter_value(data['value'], param),
                            "Час вимірювання":
safe_parse_datetime(data["timestamp"]).strftime("%d.%m.%Y %H:%M")
                        }
                        for param, data in latest_values.items()
                    ])
                    # Стилізоване відображення таблиці
                    st.dataframe(
                        latest_df,
                        hide_index=True,
                        use_container_width=True,
                        height=36 * (len(latest_df) + 1)  # Адаптивна висота
                    )
                    # Графіки для кожного параметра
                    st.subheader("Графіки показників")
                    # Групуємо дані за параметрами
                    param_data = {}
                    for data in point_data["data"]:
                        param = data["parameter"]
                        if param not in param_data:
                            param_data[param] = []
                        param_data[param].append({
                            "timestamp": safe_parse_datetime(data["timestamp"]),
                            "value": data["value"],
                            "unit": data["unit"]
                        })
                    # Визначення основних параметрів для відображення
                    priority_params = ["PM2.5", "PM10", "AQI", "NO2", "SO2", "O3", "CO"]
                    displayed_params = [p for p in priority_params if p in param_data]
                    # Якщо є пріоритетні параметри, відображаємо їх
                    if displayed_params:
                        # Вибір параметра для відображення
                        selected_param = st.selectbox(
                            "Виберіть показник для відображення",
                            displayed_params
                        )
                        if selected_param in param_data and
len(param_data[selected_param]) > 1:
                            # Створюємо DataFrame з даними
                            df = pd.DataFrame(param_data[selected_param])
                            # Створюємо графік
                            fig = create_air_quality_chart(df, selected_param)
                            # Відображаємо графік
                            st.plotly_chart(fig, use_container_width=True)
                    # Форма для підписки на сповіщення
                    st.subheader("Налаштування сповіщень")
```

```
                    with
st.form(key=f"subscription_form_{st.session_state.selected_point_id}"):
                        # Вибір параметра
                        parameter = st.selectbox(
                            "Параметр для моніторингу",
                            list(latest_values.keys())
                        )
                        # Порогове значення
                        current_value = latest_values[parameter]["value"] if parameter in
latest_values else 0
                        threshold = st.number_input(
                            f"Порогове значення (поточне: {current_value})",
                            min_value=0.0,
                            value=current_value * 1.2,  # На 20% більше за поточне
значення
                            step=0.1
                        )
                        # Кнопка підписки
                        submitted = st.form_submit_button("Підписатися на сповіщення")
                        if submitted:
                            success = create_subscription(
                                st.session_state.selected_point_id,
                                parameter,
                                threshold
                            )
                            if success:
                                st.success(f"Ви підписалися на сповіщення при перевищенні
{parameter} > {threshold}")
                else:
                    st.warning("Немає даних для цієї точки моніторингу")
        # Відображення повідомлень
        st.subheader("Останні повідомлення")
        if st.session_state.notifications:
            # Обмежуємо кількість повідомлень для відображення
            for i, notification in enumerate(st.session_state.notifications[:5]):
                with st.expander(
                    notification.get("title", "Повідомлення") + " - " +
                    safe_parse_datetime(notification.get("timestamp",
"")).strftime("%d.%m.%Y %H:%M")
                ):
                    st.write(notification.get("body", ""))
                    # Якщо є дані про точку, додаємо можливість перейти до неї
                    if "data" in notification and "point_id" in notification["data"]:
                        if st.button(f"Перейти до точки", key=f"goto_{i}"):
                            st.session_state.selected_point_id =
notification["data"]["point_id"]
                            st.rerun()
        else:
            st.info("Немає нових повідомлень")
# Вкладка аналізу даних
with tab2:
    st.subheader("Аналіз даних моніторингу")
    # Форма для фільтрації даних
    with st.form(key="data_filter_form"):
        col1, col2, col3 = st.columns(3)
        with col1:
            # Отримання унікальних параметрів
            all_points = get_all_points()
            all_parameters = set()
            for point in all_points:
                if point.get("data"):
                    for data in point["data"]:
                        all_parameters.add(data["parameter"])
            parameter = st.selectbox(
                "Параметр",
```

```
                ["Всі"] + sorted(list(all_parameters))
            )
            if parameter == "Всі":
                parameter = None
        with col2:
            min_value = st.number_input("Мінімальне значення", value=None, min_value=0.0,
step=None)
            if min_value == 0.0:
                min_value = None
        with col3:
            max_value = st.number_input("Максимальне значення", value=None,
min_value=0.0, step=None)
            if max_value == 0.0:
                max_value = None
        col1, col2 = st.columns(2)
        with col1:
            start_date = st.date_input("Дата початку", value=None)
            if start_date:
                start_date = datetime.datetime.combine(start_date,
datetime.time.min).isoformat()
            else:
                start_date = None
        with col2:
            end_date = st.date_input("Дата кінця", value=None)
            if end_date:
                end_date = datetime.datetime.combine(end_date,
datetime.time.max).isoformat()
            else:
                end_date = None
        col1, col2 = st.columns(2)
        with col1:
            # Фільтр за джерелом даних
            source_options = ["Всі джерела"] + list(set([p.get("source_type", "unknown")
for p in all_points if p.get("source_type")]))
            source_type = st.selectbox("Джерело даних", source_options)
            if source_type == "Всі джерела":
                source_type = None
        with col2:
            limit = st.slider("Кількість записів", min_value=10, max_value=5000,
value=1000, step=10)
        submit_button = st.form_submit_button("Застосувати фільтри")
    # Отримання та відображення даних
    if submit_button or 'data_loaded' not in st.session_state:
        with st.spinner("Завантаження даних..."):
            data = get_all_data(
                parameter=parameter,
                min_value=min_value,
                max_value=max_value,
                start_date=start_date,
                end_date=end_date,
                source_type=source_type,
                limit=limit
            )
            # Збереження даних у сесію
            st.session_state.filtered_data = data
            st.session_state.data_loaded = True
    # Відображення даних у таблиці
    if 'filtered_data' in st.session_state and st.session_state.filtered_data:
        data = st.session_state.filtered_data
        # Перетворення даних у DataFrame
        df = pd.DataFrame(data)
        # Форматування дати з використанням безпечного методу
        df['timestamp'] = df['timestamp'].apply(safe_parse_datetime)
        df['date'] = df['timestamp'].dt.date
        df['time'] = df['timestamp'].dt.time
```

```
        df['hour'] = df['timestamp'].dt.hour
        df['day_of_week'] = df['timestamp'].dt.dayofweek
        # Інформація про дані
        st.info(f"Знайдено {len(df)} записів")
        # Вибір типу графіка
        chart_type = st.selectbox(
            "Тип аналізу",
            ["Таблиця даних", "Розподіл значень", "Часовий ряд", "Порівняння станцій",
"Теплова карта", "Добовий аналіз"]
        )
        if chart_type == "Таблиця даних":
            # Відображення даних
            st.subheader("Таблиця даних")
            # Фільтр пошуку
            search_term = st.text_input("Пошук у таблиці", "")
            # Фільтрація даних за пошуковим запитом
            if search_term and len(df) > 0:
                # Безпечний пошук з перевіркою типів даних
                filtered_df = df[
                    df['station_name'].astype(str).str.contains(search_term, case=False,
na=False) |
                    df['parameter'].astype(str).str.contains(search_term, case=False,
na=False) |
                    df['source_type'].astype(str).str.contains(search_term, case=False,
na=False)
                ]
            else:
                filtered_df = df
            # Відображення таблиці
            if len(filtered_df) > 0:
                display_cols = ['station_name', 'parameter', 'value', 'unit', 'date',
'time', 'source_type']
                st.dataframe(
                    filtered_df[display_cols],
                    hide_index=True,
                    use_container_width=True,
                    height=500
                )
            else:
                st.warning("Немає даних для відображення після фільтрації")
        elif chart_type == "Розподіл значень":
            # Графік розподілу значень
            st.subheader("Розподіл значень")
            if parameter:
                # Фільтруємо дані за параметром
                param_df = df[df['parameter'] == parameter]
                if len(param_df) > 0:
                    # Визначаємо одиницю виміру
                    unit = param_df.iloc[0]["unit"]
                    # Створюємо гістограму
                    fig = px.histogram(
                        param_df,
                        x='value',
                        color_discrete_sequence=['#2E86C1'],
                        title=f"Розподіл значень для параметра {parameter} ({unit})",
                        labels={'value': f'Значення ({unit})'},
                        template="plotly_dark" if st.session_state.dark_theme else
"plotly_white"
                    )
                    # Налаштовуємо відображення
                    fig.update_layout(
                        xaxis_title=f"{parameter} ({unit})",
                        yaxis_title="Кількість вимірювань",
                        bargap=0.1
                    )
```

```
                    # Додаємо лінії для критичних значень
                    if parameter == "PM2.5":
                        fig.add_vline(x=12, line_width=2, line_dash="dash",
line_color="green", annotation_text="Норма")
                        fig.add_vline(x=35, line_width=2, line_dash="dash",
line_color="red", annotation_text="Небезпечний рівень")
                    elif parameter == "PM10":
                        fig.add_vline(x=54, line_width=2, line_dash="dash",
line_color="green", annotation_text="Норма")
                        fig.add_vline(x=154, line_width=2, line_dash="dash",
line_color="red", annotation_text="Небезпечний рівень")
                    st.plotly_chart(fig, use_container_width=True)
                    # Додаємо статистику
                    col1, col2, col3, col4 = st.columns(4)
                    col1.metric("Середнє значення", f"{param_df['value'].mean():.2f}")
                    col2.metric("Медіана", f"{param_df['value'].median():.2f}")
                    col3.metric("Мінімум", f"{param_df['value'].min():.2f}")
                    col4.metric("Максимум", f"{param_df['value'].max():.2f}")
                else:
                    st.warning(f"Немає даних для параметра {parameter}")
            else:
                # Якщо параметр не вибрано, показуємо для кожного параметра окремо
                for param in df['parameter'].unique():
                    param_df = df[df['parameter'] == param]
                    if len(param_df) > 0:
                        # Визначаємо одиницю виміру
                        unit = param_df.iloc[0]["unit"]
                        # Створюємо гістограму
                        fig = px.histogram(
                            param_df,
                            x='value',
                            color_discrete_sequence=['#2E86C1'],
                            title=f"Розподіл значень для параметра {param} ({unit})",
                            labels={'value': f'Значення ({unit})'},
                            template="plotly_dark" if st.session_state.dark_theme else
"plotly_white"
                        )
                        # Налаштовуємо відображення
                        fig.update_layout(
                            xaxis_title=f"{param} ({unit})",
                            yaxis_title="Кількість вимірювань",
                            bargap=0.1
                        )
                        st.plotly_chart(fig, use_container_width=True)
        elif chart_type == "Часовий ряд":
            # Графік часового ряду
            st.subheader("Часовий ряд")
            if len(df) > 0:
                # Визначення параметра для графіка
                if parameter:
                    params_to_plot = [parameter]
                else:
                    # Вибір параметра для графіка
                    params_to_plot = st.multiselect(
                        "Виберіть параметри для відображення на графіку",
                        df['parameter'].unique()
                    )
                if params_to_plot:
                    # Вибір станцій для відображення
                    all_stations = df['station_name'].unique()
                    if len(all_stations) > 1:
                        stations_to_plot = st.multiselect(
                            "Виберіть станції для відображення (до 5)",
                            all_stations,
                            default=all_stations[:1],
```

```
                            max_selections=5
                        )
                    else:
                        stations_to_plot = all_stations
                    if stations_to_plot:
                        for param in params_to_plot:
                            # Фільтруємо дані за параметром і станціями
                            param_df = df[(df['parameter'] == param) &
(df['station_name'].isin(stations_to_plot))]
                            if len(param_df) > 0:
                                # Сортуємо за часом
                                param_df = param_df.sort_values('timestamp')
                                # Визначаємо одиницю виміру
                                unit = param_df.iloc[0]["unit"]
                                # Створюємо графік
                                fig = px.line(
                                    param_df,
                                    x='timestamp',
                                    y='value',
                                    color='station_name',
                                    title=f"Часовий ряд для параметра {param} ({unit})",
                                    labels={
                                        'timestamp': 'Час',
                                        'value': f'Значення ({unit})'
                                    },
                                    template="plotly_dark" if st.session_state.dark_theme
else "plotly_white"
                                )
                                # Налаштовуємо відображення
                                fig.update_layout(
                                    xaxis_title="Час",
                                    yaxis_title=f"{param} ({unit})",
                                    hovermode="x unified",
                                    legend=dict(
                                        orientation="h",
                                        yanchor="bottom",
                                        y=1.02,
                                        xanchor="right",
                                        x=1
                                    )
                                )
                                # Оптимізація форматування часу
                                fig.update_xaxes(
                                    tickformat="%d.%m.%Y %H:%M",
                                    tickangle=-45,
                                    nticks=12,
                                    tickfont=dict(size=10)
                                )
                                # Додаємо горизонтальні смуги для категорій якості
                                if param == "PM2.5":
                                    fig.add_shape(type="rect", xref="paper", yref="y",
x0=0, x1=1, y0=0, y1=12, fillcolor="green", opacity=0.1, layer="below", line_width=0)
                                    fig.add_shape(type="rect", xref="paper", yref="y",
x0=0, x1=1, y0=12, y1=35, fillcolor="orange", opacity=0.1, layer="below", line_width=0)
                                    fig.add_shape(type="rect", xref="paper", yref="y",
x0=0, x1=1, y0=35, y1=param_df['value'].max()*1.1, fillcolor="red", opacity=0.1,
layer="below", line_width=0)
                                elif param == "PM10":
                                    fig.add_shape(type="rect", xref="paper", yref="y",
x0=0, x1=1, y0=0, y1=54, fillcolor="green", opacity=0.1, layer="below", line_width=0)
                                    fig.add_shape(type="rect", xref="paper", yref="y",
x0=0, x1=1, y0=54, y1=154, fillcolor="orange", opacity=0.1, layer="below", line_width=0)
                                    fig.add_shape(type="rect", xref="paper", yref="y",
x0=0, x1=1, y0=154, y1=param_df['value'].max()*1.1, fillcolor="red", opacity=0.1,
layer="below", line_width=0)
```

```
                            st.plotly_chart(fig, use_container_width=True)
                        else:
                            st.warning(f"Немає даних для параметра {param} на
вибраних станціях")
                    else:
                        st.warning("Виберіть станції для відображення")
                else:
                    st.warning("Виберіть параметр для відображення на графіку")
            else:
                st.warning("Недостатньо даних для побудови часового ряду")
        elif chart_type == "Порівняння станцій":
            # Порівняння середніх значень для різних станцій
            st.subheader("Порівняння станцій")
            if len(df) > 0:
                if parameter:
                    # Для одного параметра
                    param_df = df[df['parameter'] == parameter]
                    if len(param_df) > 0:
                        # Розрахунок середніх значень для кожної станції
                        station_avg =
param_df.groupby('station_name')['value'].mean().reset_index()
                        # Сортуємо за значенням
                        station_avg = station_avg.sort_values('value', ascending=False)
                        # Визначаємо одиницю виміру
                        unit = param_df.iloc[0]["unit"]
                        # Створюємо стовпчикову діаграму
                        fig = px.bar(
                            station_avg,
                            x='station_name',
                            y='value',
                            title=f"Середні значення {parameter} ({unit}) для різних
станцій",
                            labels={
                                'station_name': 'Станція',
                                'value': f'Середнє значення ({unit})'
                            },
                            template="plotly_dark" if st.session_state.dark_theme else
"plotly_white",
                            color='value',
                            color_continuous_scale='RdYlGn_r'
                        )
                        # Налаштовуємо відображення
                        fig.update_layout(
                            xaxis_title="Станція",
                            yaxis_title=f"Середнє значення {parameter} ({unit})",
                            xaxis_tickangle=-45,
                            coloraxis_showscale=False
                        )
                        st.plotly_chart(fig, use_container_width=True)
                        # Додаємо статистику по станціях
                        st.subheader("Статистика по станціях")
                        # Розширена статистика по станціях
                        station_stats = param_df.groupby('station_name').agg({
                            'value': ['mean', 'median', 'min', 'max', 'count']
                        }).reset_index()
                        # Перейменовуємо стовпці
                        station_stats.columns = ['Станція', 'Середнє', 'Медіана',
'Мінімум', 'Максимум', 'Кількість']
                        # Округлюємо числові стовпці
                        for col in ['Середнє', 'Медіана', 'Мінімум', 'Максимум']:
                            station_stats[col] = station_stats[col].round(2)
                        # Відображаємо таблицю
                        st.dataframe(
                            station_stats,
                            hide_index=True,
```

```
                            use_container_width=True
                        )
                    else:
                        st.warning(f"Немає даних для параметра {parameter}")
                else:
                    # Вибір параметра для порівняння
                    param_to_compare = st.selectbox(
                        "Виберіть параметр для порівняння станцій",
                        df['parameter'].unique()
                    )
                    param_df = df[df['parameter'] == param_to_compare]
                    if len(param_df) > 0:
                        # Розрахунок середніх значень для кожної станції
                        station_avg =
param_df.groupby('station_name')['value'].mean().reset_index()
                        # Сортуємо за значенням
                        station_avg = station_avg.sort_values('value', ascending=False)
                        # Визначаємо одиницю виміру
                        unit = param_df.iloc[0]["unit"]
                        # Створюємо стовпчикову діаграму
                        fig = px.bar(
                            station_avg,
                            x='station_name',
                            y='value',
                            title=f"Середні значення {param_to_compare} ({unit}) для
різних станцій",
                            labels={
                                'station_name': 'Станція',
                                'value': f'Середнє значення ({unit})'
                            },
                            template="plotly_dark" if st.session_state.dark_theme else
"plotly_white",
                            color='value',
                            color_continuous_scale='RdYlGn_r'
                        )
                        # Налаштовуємо відображення
                        fig.update_layout(
                            xaxis_title="Станція",
                            yaxis_title=f"Середнє значення {param_to_compare} ({unit})",
                            xaxis_tickangle=-45,
                            coloraxis_showscale=False
                        )
                        st.plotly_chart(fig, use_container_width=True)
                        # Додаємо статистику по станціях
                        st.subheader("Статистика по станціях")
                        # Розширена статистика по станціях
                        station_stats = param_df.groupby('station_name').agg({
                            'value': ['mean', 'median', 'min', 'max', 'count']
                        }).reset_index()
                        # Перейменовуємо стовпці
                        station_stats.columns = ['Станція', 'Середнє', 'Медіана',
'Мінімум', 'Максимум', 'Кількість']
                        # Округлюємо числові стовпці
                        for col in ['Середнє', 'Медіана', 'Мінімум', 'Максимум']:
                            station_stats[col] = station_stats[col].round(2)
                        # Відображаємо таблицю
                        st.dataframe(
                            station_stats,
                            hide_index=True,
                            use_container_width=True
                        )
                    else:
                        st.warning(f"Немає даних для параметра {param_to_compare}")
            else:
                st.warning("Недостатньо даних для порівняння станцій")
```

```
        elif chart_type == "Теплова карта":
            # Теплова карта значень
            st.subheader("Теплова карта значень")
            if len(df) > 0:
                try:
                    # Вибираємо параметри для теплової карти
                    if parameter:
                        params_for_heatmap = [parameter]
                    else:
                        # Вибираємо найпопулярніші параметри
                        param_counts = df['parameter'].value_counts()
                        popular_params = param_counts[param_counts > 10].index.tolist()
                        # Якщо є популярні параметри, показуємо їх
                        if popular_params:
                            params_for_heatmap = popular_params[:5]  # Обмежуємо до 5
параметрів
                        else:
                            params_for_heatmap = df['parameter'].unique()[:5]
                    # Фільтруємо дані для теплової карти
                    heatmap_df = df[df['parameter'].isin(params_for_heatmap)]
                    # Визначаємо станції для теплової карти
                    station_counts = heatmap_df['station_name'].value_counts()
                    popular_stations = station_counts[station_counts > 5].index.tolist()
                    # Якщо станцій забагато, дозволяємо вибрати
                    if len(popular_stations) > 15:
                        selected_stations = st.multiselect(
                            "Виберіть станції для теплової карти (до 15)",
                            popular_stations,
                            default=popular_stations[:10],
                            max_selections=15
                        )
                        if selected_stations:
                            heatmap_df =
heatmap_df[heatmap_df['station_name'].isin(selected_stations)]
                        else:
                            heatmap_df =
heatmap_df[heatmap_df['station_name'].isin(popular_stations[:10])]
                    # Створення теплової карти для параметрів та станцій
                    pivot_table = pd.pivot_table(
                        heatmap_df,
                        values='value',
                        index='station_name',
                        columns='parameter',
                        aggfunc='mean'
                    )
                    # Перевіряємо результат
                    if not pivot_table.empty and len(pivot_table) > 0 and
len(pivot_table.columns) > 0:
                        # Створюємо теплову карту
                        fig = px.imshow(
                            pivot_table,
                            labels=dict(x="Параметр", y="Станція", color="Значення"),
                            title="Теплова карта значень параметрів для різних станцій",
                            color_continuous_scale="Viridis",
                            template="plotly_dark" if st.session_state.dark_theme else
"plotly_white",
                            aspect="auto"  # Автоматичне співвідношення сторін
                        )
                        # Налаштування для кращого відображення
                        fig.update_layout(
                            height=max(400, len(pivot_table) * 25),  # Динамічна висота
                            margin=dict(l=150, r=20, t=50, b=50)  # Більше місця для назв
станцій
                        )
                        st.plotly_chart(fig, use_container_width=True)
```

```
                    else:
                        st.warning("Недостатньо даних для побудови теплової карти")
                except Exception as e:
                    st.error(f"Помилка при створенні теплової карти: {str(e)}")
                    st.info("Спробуйте вибрати менше станцій або параметрів для
відображення")
            else:
                st.warning("Недостатньо даних для побудови теплової карти")
        elif chart_type == "Добовий аналіз":
            # Аналіз зміни значень протягом доби
            st.subheader("Добовий аналіз")
            if len(df) > 0:
                # Вибираємо параметр для аналізу
                if parameter:
                    selected_param = parameter
                else:
                    selected_param = st.selectbox(
                        "Виберіть параметр для добового аналізу",
                        df['parameter'].unique()
                    )

                # Фільтруємо дані за параметром
                param_df = df[df['parameter'] == selected_param]

                if len(param_df) > 0:
                    # Визначаємо одиницю виміру
                    unit = param_df.iloc[0]["unit"]
                    # Групуємо дані за годинами доби
                    hourly_avg = param_df.groupby('hour')['value'].agg(['mean', 'median',
'min', 'max', 'count']).reset_index()
                    # Створюємо графік
                    fig = make_subplots(specs=[[{"secondary_y": True}]])
                    # Додаємо лінію середніх значень
                    fig.add_trace(
                        go.Scatter(
                            x=hourly_avg['hour'],
                            y=hourly_avg['mean'],
                            mode='lines+markers',
                            name='Середнє',
                            line=dict(color='#2E86C1', width=3),
                            marker=dict(size=8)
                        ),
                        secondary_y=False
                    )
                    # Додаємо область мін-макс значень
                    fig.add_trace(
                        go.Scatter(
                            x=hourly_avg['hour'],
                            y=hourly_avg['max'],
                            mode='lines',
                            name='Максимум',
                            line=dict(width=0),
                            marker=dict(color="#2E86C1"),
                            showlegend=False
                        ),
                        secondary_y=False
                    )
                    fig.add_trace(
                        go.Scatter(
                            x=hourly_avg['hour'],
                            y=hourly_avg['min'],
                            mode='lines',
                            name='Мінімум',
                            line=dict(width=0),
                            marker=dict(color="#2E86C1"),
```

```
                        fill='tonexty',
                        fillcolor='rgba(46, 134, 193, 0.2)',
                        showlegend=False
                    ),
                    secondary_y=False
                )
                # Додаємо кількість вимірювань на допоміжну вісь
                fig.add_trace(
                    go.Bar(
                        x=hourly_avg['hour'],
                        y=hourly_avg['count'],
                        name='Кількість вимірювань',
                        marker=dict(color='rgba(255, 193, 7, 0.6)'),
                        opacity=0.7
                    ),
                    secondary_y=True
                )
                # Налаштовуємо відображення
                fig.update_layout(
                    title=f"Зміна {selected_param} протягом доби",
                    template="plotly_dark" if st.session_state.dark_theme else
"plotly_white",
                    hovermode="x unified",
                    legend=dict(
                        orientation="h",
                        yanchor="bottom",
                        y=1.02,
                        xanchor="right",
                        x=1
                    )
                )
                # Налаштовуємо осі
                fig.update_xaxes(
                    title_text="Година доби",
                    tickmode='array',
                    tickvals=list(range(0, 24)),
                    ticktext=[f"{h:02d}:00" for h in range(0, 24)]
                )
                fig.update_yaxes(
                    title_text=f"{selected_param} ({unit})",
                    secondary_y=False
                )
                fig.update_yaxes(
                    title_text="Кількість вимірювань",
                    secondary_y=True
                )
                st.plotly_chart(fig, use_container_width=True)
                # Аналіз за днями тижня
                st.subheader("Аналіз за днями тижня")
                # Групуємо дані за днями тижня
                days = ['Понеділок', 'Вівторок', 'Середа', 'Четвер', "П'ятниця",
'Субота', 'Неділя']
                day_avg = param_df.groupby('day_of_week')['value'].agg(['mean',
'median', 'min', 'max', 'count']).reset_index()
                day_avg['day_name'] = day_avg['day_of_week'].apply(lambda x:
days[int(x)] if 0 <= int(x) < 7 else 'Невідомо')
                # Створюємо графік
                fig = px.bar(
                    day_avg,
                    x='day_name',
                    y='mean',
                    title=f"Середні значення {selected_param} за днями тижня",
                    labels={'day_name': 'День тижня', 'mean': f'Середнє значення
({unit})'},
```

```
                        template="plotly_dark" if st.session_state.dark_theme else
"plotly_white",
                        color='mean',
                        color_continuous_scale='RdYlGn_r',
                        error_y=day_avg['max'] - day_avg['mean']
                    )
                    # Налаштовуємо відображення
                    fig.update_layout(
                        xaxis=dict(
                            categoryorder='array',
                            categoryarray=days
                        ),
                        xaxis_title="День тижня",
                        yaxis_title=f"Середнє значення {selected_param} ({unit})",
                        coloraxis_showscale=False
                    )
                    st.plotly_chart(fig, use_container_width=True)
                else:
                    st.warning(f"Немає даних для параметра {selected_param}")
            else:
                st.warning("Недостатньо даних для добового аналізу")
        # Завантаження даних
        if len(df) > 0:
            st.subheader("Експорт даних")
            # Підготовка даних для експорту
            export_df = df[['station_name', 'parameter', 'value', 'unit', 'timestamp',
'source_type']].copy()
            export_df['timestamp'] = export_df['timestamp'].dt.strftime('%Y-%m-%d
%H:%M:%S')
            # Перетворення DataFrame у різні формати
            try:
                csv = export_df.to_csv(index=False)
                # Кнопки для завантаження
                col1, col2 = st.columns(2)
                with col1:
                    st.download_button(
                        label="Завантажити CSV",
                        data=csv,
                        file_name="eco_monitoring_data.csv",
                        mime="text/csv"
                    )
                with col2:
                    try:
                        bio = io.BytesIO()
                        with pd.ExcelWriter(bio, engine='openpyxl') as writer:
                            export_df.to_excel(writer, index=False, sheet_name='Sheet1')
                        st.download_button(
                            label="Завантажити Excel",
                            data=bio.getvalue(),
                            file_name="eco_monitoring_data.xlsx",
                            mime="application/vnd.ms-excel"
                        )
                    except Exception as e:
                        st.error(f"Помилка створення Excel-файлу: {str(e)}")
            except Exception as e:
                st.error(f"Помилка експорту даних: {str(e)}")
    else:
        st.info("Немає даних для відображення. Спробуйте змінити параметри фільтрації.")
# Вкладка статистики
with tab3:
    st.subheader("Статистика моніторингу")
    # Отримання статистичних даних
    stats = get_statistics()
    if stats:
        # Відображення загальних метрик
```

```
        col1, col2, col3 = st.columns(3)
        with col1:
            st.metric("Загальна кількість точок моніторингу", stats.get("total_points",
0))
        if "parameters" in stats:
            param_count = len(stats["parameters"])
            with col2:
                st.metric("Кількість параметрів моніторингу", param_count)
            # Визначення загальної кількості вимірювань
            total_measurements = sum([p.get("measurements", 0) for p in
stats["parameters"].values()])
            with col3:
                st.metric("Загальна кількість вимірювань", total_measurements)
        # Розділення на дві колонки
        col1, col2 = st.columns(2)
        with col1:
            st.subheader("Розподіл за джерелами даних")
            # Кількість точок за джерелами
            if "sources" in stats and stats["sources"]:
                # Створення DataFrame для візуалізації
                source_data = pd.DataFrame({
                    "Джерело": list(stats["sources"].keys()),
                    "Кількість": list(stats["sources"].values())
                })
                # Сортування за кількістю
                source_data = source_data.sort_values("Кількість", ascending=False)
                # Діаграма розподілу за джерелами
                fig = px.pie(
                    source_data,
                    values="Кількість",
                    names="Джерело",
                    title="Розподіл точок моніторингу за джерелами",
                    hole=0.4,  # Створює кільцеву діаграму
                    template="plotly_dark" if st.session_state.dark_theme else
"plotly_white"
                )
                # Налаштування відображення
                fig.update_traces(
                    textposition='inside',
                    textinfo='percent+label',
                    marker=dict(line=dict(color='#FFFFFF', width=1))
                )
                st.plotly_chart(fig, use_container_width=True)
                # Відображення таблиці з джерелами
                st.dataframe(
                    source_data,
                    hide_index=True,
                    use_container_width=True
                )
        with col2:
            st.subheader("Статистика параметрів")
            if "parameters" in stats and stats["parameters"]:
                # Створення DataFrame для параметрів
                param_data = []
                for param, values in stats["parameters"].items():
                    param_data.append({
                        "Параметр": param,
                        "Середнє значення": values["average"],
                        "Мінімум": values["min"],
                        "Максимум": values["max"],
                        "Кількість вимірювань": values["measurements"]
                    })
                param_df = pd.DataFrame(param_data)
                # Сортування за кількістю вимірювань
                param_df = param_df.sort_values("Кількість вимірювань", ascending=False)
```

```
                # Відображення таблиці
                st.dataframe(
                    param_df,
                    hide_index=True,
                    use_container_width=True
                )
                # Діаграма середніх значень
                st.subheader("Середні значення параметрів")
                # Створюємо стовпчикову діаграму
                fig = px.bar(
                    param_df,
                    x="Параметр",
                    y="Середнє значення",
                    title="Середні значення параметрів",
                    color="Кількість вимірювань",
                    color_continuous_scale="Viridis",
                    template="plotly_dark" if st.session_state.dark_theme else
"plotly_white"
                )
                # Налаштовуємо відображення
                fig.update_layout(
                    xaxis_title="Параметр",
                    yaxis_title="Середнє значення",
                    xaxis_tickangle=-45
                )
                st.plotly_chart(fig, use_container_width=True)
                # Порівняльна діаграма мін/макс/середнього
                st.subheader("Порівняння мін/макс/середнього")
                # Створюємо графік для порівняння
                fig = go.Figure()
                # Вибираємо топ-5 параметрів за кількістю вимірювань
                top_params = param_df.head(5)["Параметр"].tolist()
                # Фільтруємо дані
                compare_df = param_df[param_df["Параметр"].isin(top_params)]
                # Додаємо бари для мінімуму, середнього та максимуму
                for param in compare_df["Параметр"]:
                    param_row = compare_df[compare_df["Параметр"] == param].iloc[0]
                    fig.add_trace(go.Bar(
                        x=[param],
                        y=[param_row["Максимум"]],
                        name="Максимум",
                        marker_color='rgb(255, 0, 0)'
                    ))
                    fig.add_trace(go.Bar(
                        x=[param],
                        y=[param_row["Середнє значення"]],
                        name="Середнє",
                        marker_color='rgb(255, 165, 0)'
                    ))
                    fig.add_trace(go.Bar(
                        x=[param],
                        y=[param_row["Мінімум"]],
                        name="Мінімум",
                        marker_color='rgb(0, 128, 0)'
                    ))
                # Налаштовуємо відображення
                fig.update_layout(
                    title="Порівняння мінімального, середнього та максимального значень",
                    barmode='group',
                    template="plotly_dark" if st.session_state.dark_theme else
"plotly_white"
                )
                st.plotly_chart(fig, use_container_width=True)
    # Часовий аналіз
    st.subheader("Часовий аналіз даних")
```

```
    # Отримання даних для часового аналізу
    temporal_data = get_all_data(limit=5000)  # Обмежуємо кількість записів для швидкості
    if temporal_data:
        # Перетворення в DataFrame
        temp_df = pd.DataFrame(temporal_data)
        # Безпечне перетворення часової мітки
        temp_df['timestamp'] = temp_df['timestamp'].apply(safe_parse_datetime)
        # Додаємо часові компоненти
        temp_df['hour'] = temp_df['timestamp'].dt.hour
        temp_df['day'] = temp_df['timestamp'].dt.day
        temp_df['month'] = temp_df['timestamp'].dt.month
        temp_df['year'] = temp_df['timestamp'].dt.year
        temp_df['day_of_week'] = temp_df['timestamp'].dt.dayofweek
        # Вибір параметра для аналізу
        unique_params = temp_df['parameter'].unique()
        if len(unique_params) > 0:
            selected_param = st.selectbox("Виберіть параметр для часового аналізу",
unique_params)
            # Фільтрація за вибраним параметром
            param_df = temp_df[temp_df['parameter'] == selected_param]
            if not param_df.empty:
                # Розділення на дві колонки
                col1, col2 = st.columns(2)
                with col1:
                    # Середні значення за годинами доби
                    st.subheader("Середні значення за годинами доби")
                    # Групуємо дані за годинами
                    hourly_avg = param_df.groupby('hour')['value'].mean().reset_index()
                    # Створюємо графік
                    fig = px.line(
                        hourly_avg,
                        x='hour',
                        y='value',
                        title=f"Середні значення {selected_param} за годинами доби",
                        labels={'hour': 'Година доби', 'value': 'Середнє значення'},
                        template="plotly_dark" if st.session_state.dark_theme else
"plotly_white"
                    )
                    # Налаштовуємо відображення
                    fig.update_layout(
                        xaxis=dict(
                            tickmode='array',
                            tickvals=list(range(0, 24)),
                            ticktext=[f"{h:02d}:00" for h in range(0, 24)]
                        ),
                        xaxis_title="Година доби",
                        yaxis_title=f"Середнє значення {selected_param}"
                    )
                    st.plotly_chart(fig, use_container_width=True)
                with col2:
                    # Середні значення за днями тижня
                    st.subheader("Середні значення за днями тижня")
                    # Групуємо дані за днями тижня
                    days = ['Понеділок', 'Вівторок', 'Середа', 'Четвер', "П'ятниця",
'Субота', 'Неділя']
                    day_avg =
param_df.groupby('day_of_week')['value'].mean().reset_index()
                    day_avg['day_name'] = day_avg['day_of_week'].apply(lambda x:
days[int(x)] if 0 <= int(x) < 7 else 'Невідомо')
                    # Створюємо графік
                    fig = px.bar(
                        day_avg,
                        x='day_name',
                        y='value',
                        title=f"Середні значення {selected_param} за днями тижня",
```

```
                        labels={'day_name': 'День тижня', 'value': 'Середнє значення'},
                        template="plotly_dark" if st.session_state.dark_theme else
"plotly_white"
                    )
                    # Налаштовуємо відображення
                    fig.update_layout(
                        xaxis=dict(
                            categoryorder='array',
                            categoryarray=days
                        ),
                        xaxis_title="День тижня",
                        yaxis_title=f"Середнє значення {selected_param}"
                    )
                    st.plotly_chart(fig, use_container_width=True)
                # Теплова карта годин/днів тижня
                st.subheader("Теплова карта значень за годинами та днями тижня")
                # Перевіряємо, чи достатньо даних
                if len(param_df) > 100:
                    try:
                        # Групування за годинами та днями тижня
                        heat_data = param_df.groupby(['hour',
'day_of_week'])['value'].mean().reset_index()
                        # Перевірка наявності даних
                        if not heat_data.empty:
                            # Створення двовимірної таблиці для теплової карти
                            heat_pivot = heat_data.pivot(index='day_of_week',
columns='hour', values='value')
                            # Заповнюємо пропуски нулями
                            heat_pivot = heat_pivot.fillna(0)
                            # Переконуємось, що у нас є всі дні тижня (0-6)
                            for day in range(7):
                                if day not in heat_pivot.index:
                                    heat_pivot.loc[day] = [0] * len(heat_pivot.columns)
                            # Переконуємось, що у нас є всі години доби (0-23)
                            for hour in range(24):
                                if hour not in heat_pivot.columns:
                                    heat_pivot[hour] = [0] * len(heat_pivot.index)
                            # Сортуємо індекси та колонки
                            heat_pivot = heat_pivot.sort_index()
                            heat_pivot = heat_pivot.sort_index(axis=1)
                            # Замінюємо індекси на назви днів
                            heat_pivot.index = [days[i] if 0 <= i < 7 else f'День {i}'
for i in heat_pivot.index]
                            # Створення теплової карти
                            fig = px.imshow(
                                heat_pivot,
                                labels=dict(x="Година доби", y="День тижня",
color="Значення"),
                                title=f"Теплова карта значень {selected_param} за
годинами та днями тижня",
                                x=sorted(heat_pivot.columns),
                                y=heat_pivot.index,
                                color_continuous_scale="Viridis",
                                template="plotly_dark" if st.session_state.dark_theme
else "plotly_white"
                            )
                            # Налаштовуємо відображення
                            fig.update_xaxes(
                                tickmode='array',
                                tickvals=list(range(0, 24)),
                                ticktext=[f"{h:02d}:00" for h in range(0, 24)]
                            )
                            st.plotly_chart(fig, use_container_width=True)
                        else:
```

```
                            st.warning(f"Недостатньо даних для побудови теплової карти
для параметра {selected_param}")
                    except Exception as e:
                        st.error(f"Помилка при створенні теплової карти: {str(e)}")
                        st.warning(f"Недостатньо даних для побудови теплової карти для
параметра {selected_param}")
                else:
                    st.warning(f"Недостатньо даних для теплової карти (потрібно не менше
100 вимірювань)")
# Додавання кнопки оновлення
st.sidebar.title("Оновлення даних")
if st.sidebar.button("🔄 Оновити дані"):
    # Очищення кешу
    get_all_points.clear()
    if st.session_state.selected_point_id:
        get_point_data.clear()
    get_statistics.clear()
    # Запуск перегенерації даних на сервері
    try:
        response = requests.post(f"{SERVER_URL}/api/regenerate_data")
        if response.status_code == 200:
            st.sidebar.success("Запит на оновлення даних відправлено успішно")
        else:
            st.sidebar.error(f"Помилка відправки запиту: {response.status_code}")
    except Exception as e:
        st.sidebar.error(f"Помилка: {str(e)}")
    # Оновлення сторінки
    st.rerun()
# Довідкова інформація
st.sidebar.title("Довідка")
with st.sidebar.expander("Про показники якості повітря"):
    st.markdown("""
    **PM2.5** - Тверді частинки розміром до 2.5 мкм. Можуть проникати глибоко в легені,
викликаючи респіраторні захворювання.
    **PM10** - Тверді частинки розміром до 10 мкм. Викликають подразнення дихальних
шляхів.
    **NO2 (Діоксид азоту)** - Токсичний газ, що утворюється при згорянні палива. Викликає
подразнення дихальних шляхів, сприяє розвитку астми.
    **SO2 (Діоксид сірки)** - Утворюється при спалюванні палива з високим вмістом сірки.
Викликає подразнення очей та дихальних шляхів.
    **O3 (Озон)** - Вторинний забруднювач, який утворюється в результаті хімічних
реакцій. Викликає проблеми з диханням, подразнення очей.
    **CO (Монооксид вуглецю)** - Токсичний газ без запаху, що утворюється при неповному
згорянні палива. Зменшує здатність крові переносити кисень.
    **AQI (Індекс якості повітря)** - Загальний показник якості повітря, який враховує
концентрації різних забруднювачів.
    """)
with st.sidebar.expander("Джерела даних"):
    st.markdown("""
    **SaveEcoBot** - Українська система моніторингу якості повітря з мережею власних
датчиків та агрегацією даних з різних джерел.
    [https://www.saveecobot.com](https://www.saveecobot.com)
    **OpenAQ** - Міжнародна платформа з відкритим кодом для збору та поширення даних про
якість повітря з усього світу.
    [https://openaq.org](https://openaq.org)
    **Eco-City** - Мережа автоматизованих станцій моніторингу якості повітря та платформа
для громадського моніторингу в Україні.
    [https://eco-city.org.ua](https://eco-city.org.ua)
    """)
# Інформація про додаток
st.sidebar.markdown("---")
st.sidebar.info(
```

```
    "Система моніторингу екологічної безпеки використовує дані з різних джерел для
надання інформації про якість повітря та оповіщення про перевищення небезпечних рівнів
забруднення."
)
# Відображення інформації внизу сторінки
st.sidebar.markdown("---")
st.sidebar.write("ID пристрою для сповіщень:", st.session_state.device_id)
st.sidebar.write(f"Останнє     оновлення:     {datetime.datetime.now().strftime('%d.%m.%Y
%H:%M:%S')}")
```

server.py

```
from flask import Flask, request, jsonify
from sqlalchemy import create_engine, Column, Integer, String, Float, DateTime,
ForeignKey, func
from sqlalchemy.orm import sessionmaker, Session, declarative_base
from typing import List, Dict, Any, Optional
import requests
import sqlite3
import datetime
import threading
import json
import os
import logging
import random
import numpy as np
import time
# Налаштування логування
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler("server.log"),
        logging.StreamHandler()
    ]
)
logger = logging.getLogger(__name__)
# Ініціалізація Flask
app = Flask(__name__)
# Підключення до SQLite
DATABASE_URL = "sqlite:///./eco_monitoring.db"
engine = create_engine(DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()
# Моделі бази даних
class MonitoringPoint(Base):
    __tablename__ = "monitoring_points"
    id = Column(Integer, primary_key=True, index=True)
    name = Column(String, index=True)
    latitude = Column(Float)
    longitude = Column(Float)
    source_id = Column(String, index=True)
    source_type = Column(String)
class MonitoringData(Base):
    __tablename__ = "monitoring_data"
    id = Column(Integer, primary_key=True, index=True)
    point_id = Column(Integer, ForeignKey("monitoring_points.id"))
    timestamp = Column(DateTime, default=datetime.datetime.utcnow)
    parameter = Column(String)
    value = Column(Float)
    unit = Column(String)
```

```
class UserSubscription(Base):
    __tablename__ = "user_subscriptions"
    id = Column(Integer, primary_key=True, index=True)
    device_id = Column(String, index=True)
    point_id = Column(Integer, ForeignKey("monitoring_points.id"))
    parameter = Column(String)
    threshold = Column(Float)
    created_at = Column(DateTime, default=datetime.datetime.utcnow)
# Створення таблиць
Base.metadata.create_all(bind=engine)
# Функція для отримання сесії бази даних
def get_db():
    db = SessionLocal()
    try:
        return db
    finally:
        db.close()
# Функція для визначення чи координати належать Україні
def is_ukrainian_coordinates(lat, lon):
    """
    Перевірка чи координати належать території України
    Args:
        lat: Широта
        lon: Довгота
    Returns:
        bool: True якщо координати в Україні, False інакше
    """
    # Приблизні межі України
    min_lat = 44.0
    max_lat = 53.0
    min_lon = 22.0
    max_lon = 41.0
    return min_lat <= lat <= max_lat and min_lon <= lon <= max_lon
# Функція для генерації синтетичних даних на основі реальних
def generate_synthetic_data(base_value, time_factor=1.0, random_factor=0.2):
    """
    Генерація синтетичних даних для точок моніторингу
    Args:
        base_value: Початкове значення для генерації
        time_factor: Фактор часу для моделювання добових коливань
        random_factor: Фактор випадковості (0.0 - 1.0)
    Returns:
        float: Згенероване значення
    """
    # Добові коливання (синусоїда)
    hour = datetime.datetime.now().hour
    daily_factor = 1.0 + 0.2 * np.sin(np.pi * hour / 12)
    # Тижневі коливання (робочі дні vs вихідні)
    weekday = datetime.datetime.now().weekday()
    weekly_factor = 1.0 + 0.1 * (1 if weekday < 5 else -1)
    # Випадкові коливання
    random_variation = 1.0 + random_factor * (random.random() * 2 - 1)
    # Обчислення значення
    synthetic_value = base_value * daily_factor * weekly_factor * random_variation *
time_factor
    return round(synthetic_value, 2)
# Функція для завантаження даних з SaveEcoBot та генерації синтетичних даних
def fetch_saveecobot_synthetic():
    """Отримання даних з SaveEcoBot API та генерація синтетичних даних"""
    db = get_db()
    try:
        # Основний API URL для SaveEcoBot
        api_url = "https://api.saveecobot.com/output.json"
        # Виконання запиту до API
        response = requests.get(api_url, timeout=30)
```

```
        logger.info(f"Відповідь від API SaveEcoBot: {response.status_code}")
        if response.status_code == 200:
            data = response.json()
            logger.info(f"Отримано {len(data)} станцій з SaveEcoBot API")
            # Зберігаємо унікальні станції
            processed_ids = set()
            unique_data = []
            for station in data:
                station_id = station.get("stationID") or station.get("id")
                if not station_id or station_id in processed_ids:
                    continue
                processed_ids.add(station_id)
                unique_data.append(station)
            logger.info(f"Унікальних станцій від SaveEcoBot: {len(unique_data)}")
            # Фільтруємо станції за координатами (тільки Україна)
            ukraine_stations = []
            for station in unique_data:
                latitude = station.get("latitude")
                longitude = station.get("longitude")
                if not latitude or not longitude:
                    # Спроба отримати координати з альтернативних полів
                    if "coords" in station:
                        coords = station["coords"]
                        if isinstance(coords, list) and len(coords) >= 2:
                            latitude = coords[0]
                            longitude = coords[1]
                        elif isinstance(coords, dict):
                            latitude = coords.get("lat")
                            longitude = coords.get("lon")
                # Якщо координати вдалося отримати
                if latitude and longitude:
                    # Перевіряємо чи координати в Україні
                    if is_ukrainian_coordinates(float(latitude), float(longitude)):
                        ukraine_stations.append(station)
            logger.info(f"Станцій в Україні: {len(ukraine_stations)}")
            # Якщо станцій в Україні замало, додаємо ще станції, що мають у назві
українські міста
            if len(ukraine_stations) < 20:
                ukrainian_cities = [
                    "Київ", "Kyiv", "Kharkiv", "Харків", "Odesa", "Одеса", "Lviv",
"Львів",
                    "Дніпро", "Dnipro", "Запоріжжя", "Zaporizhzhia"
                ]
                for station in unique_data:
                    if station not in ukraine_stations:
                        station_name = station.get("name", "") or
station.get("stationName", "")
                        city_name = station.get("cityName", "") or station.get("city",
"")
                        if any(city in station_name or city in city_name for city in
ukrainian_cities):
                            ukraine_stations.append(station)
                logger.info(f"Станцій в Україні після доповнення:
{len(ukraine_stations)}")
            # Обробка отриманих даних
            for station in ukraine_stations:
                try:
                    # Перевірка необхідних полів
                    station_id = station.get("stationID") or station.get("id")
                    if not station_id:
                        continue
                    # Перевірка координат
                    latitude = station.get("latitude")
                    longitude = station.get("longitude")
                    if not latitude or not longitude:
```

```
                        # Спроба отримати координати з альтернативних полів
                        if "coords" in station:
                            coords = station["coords"]
                            if isinstance(coords, list) and len(coords) >= 2:
                                latitude = coords[0]
                                longitude = coords[1]
                            elif isinstance(coords, dict):
                                latitude = coords.get("lat")
                                longitude = coords.get("lon")
                        # Якщо все ще немає координат, пропускаємо станцію
                        if not latitude or not longitude:
                            continue
                    # Формування назви станції
                    station_name = None
                    if "cityName" in station and "stationName" in station:
                        station_name = f"{station['cityName']} -
{station['stationName']}"
                    elif "cityName" in station:
                        station_name = f"{station['cityName']} - Станція {station_id}"
                    elif "city" in station and "name" in station:
                        station_name = f"{station['city']} - {station['name']}"
                    elif "name" in station:
                        station_name = station["name"]
                    else:
                        station_name = f"SaveEcoBot Станція {station_id}"
                    # Перевіряємо, чи існує точка моніторингу
                    existing_point = db.query(MonitoringPoint).filter(
                        MonitoringPoint.source_id == str(station_id),
                        MonitoringPoint.source_type == "saveecobot"
                    ).first()
                    if not existing_point:
                        # Створюємо нову точку
                        new_point = MonitoringPoint(
                            name=station_name,
                            latitude=float(latitude),
                            longitude=float(longitude),
                            source_id=str(station_id),
                            source_type="saveecobot"
                        )
                        db.add(new_point)
                        db.commit()
                        db.refresh(new_point)
                        point_id = new_point.id
                        logger.info(f"Створено нову точку: {station_name}")
                    else:
                        point_id = existing_point.id
                    # Додаємо дані моніторингу - використовуємо як основу реальні дані
                    base_values = {}
                    # Перевіряємо різні формати даних
                    if "pollutants" in station:
                        for pollutant in station["pollutants"]:
                            if all(key in pollutant for key in ["pollutantName",
"value"]):
                                param = pollutant["pollutantName"]
                                value = pollutant["value"]
                                unit = pollutant.get("unit", "")
                                base_values[param] = (float(value), unit)
                    # Альтернативний формат
                    elif "measurements" in station:
                        for measure in station["measurements"]:
                            if all(key in measure for key in ["parameter", "value"]):
                                param = measure["parameter"]
                                value = measure["value"]
                                unit = measure.get("unit", "")
                                base_values[param] = (float(value), unit)
```

```
                    # Перевіряємо формат з параметрами в корені об'єкта
                    else:
                        for param_key, param_name in [
                            ("pm25", "PM2.5"), ("pm2_5", "PM2.5"), ("pm10", "PM10"),
                            ("no2", "NO2"), ("so2", "SO2"), ("o3", "O3"), ("co", "CO")
                        ]:
                            if param_key in station and station[param_key] is not None:
                                value = float(station[param_key])
                                unit = "µg/m³"
                                base_values[param_name] = (value, unit)
                    # Якщо даних немає, використовуємо стандартні значення
                    if not base_values:
                        base_values = {
                            "PM2.5": (15.0, "µg/m³"),
                            "PM10": (30.0, "µg/m³"),
                            "NO2": (20.0, "µg/m³"),
                            "SO2": (10.0, "µg/m³"),
                            "O3": (30.0, "µg/m³"),
                            "CO": (500.0, "µg/m³")
                        }
                    # Видаляємо старі дані для цієї точки
                    old_data = db.query(MonitoringData).filter(
                        MonitoringData.point_id == point_id
                    ).delete()
                    db.commit()
                    # Генеруємо синтетичні дані на основі реальних базових значень
                    # Для демонстрації ми генеруємо дані за останні 24 години з
інтервалом у годину
                    for hours_ago in range(24, -1, -1):  # від 24 годин тому до зараз
                        timestamp = datetime.datetime.now() -
datetime.timedelta(hours=hours_ago)
                        for param, (base_value, unit) in base_values.items():
                            # Генеруємо синтетичне значення на основі базового
                            time_factor = 1.0 + 0.1 * (24 - hours_ago) / 24.0  # Легкий
тренд до збільшення з часом
                            synthetic_value = generate_synthetic_data(base_value,
time_factor)
                            # Додаємо синтетичне вимірювання
                            new_data = MonitoringData(
                                point_id=point_id,
                                parameter=param,
                                value=synthetic_value,
                                unit=unit,
                                timestamp=timestamp
                            )
                            db.add(new_data)
                    # Також додаємо прогнозні значення на наступні 24 години (для
демонстрації можливостей)
                    for hours_ahead in range(1, 25):  # від 1 до 24 годин вперед
                        timestamp = datetime.datetime.now() +
datetime.timedelta(hours=hours_ahead)
                        for param, (base_value, unit) in base_values.items():
                            # Генеруємо прогнозне значення - збільшуємо фактор
випадковості для майбутніх значень
                            time_factor = 1.0 + 0.15 * hours_ahead / 24.0  # Легкий тренд
до збільшення в майбутньому
                            random_factor = 0.2 + 0.2 * hours_ahead / 24.0  # Збільшуємо
невизначеність для майбутніх значень
                            synthetic_value = generate_synthetic_data(base_value,
time_factor, random_factor)
                            # Додаємо синтетичне вимірювання
                            new_data = MonitoringData(
                                point_id=point_id,
                                parameter=param,
                                value=synthetic_value,
```

```
                                unit=unit,
                                timestamp=timestamp
                            )
                            db.add(new_data)
                    db.commit()
                except Exception as e:
                    logger.error(f"Помилка при обробці станції SaveEcoBot: {e}")
                    db.rollback()
            logger.info(f"Успішно завантажено дані з SaveEcoBot та згенеровано синтетичні
дані")
        else:
            logger.error(f"Помилка запиту до SaveEcoBot API: {response.status_code}")
    except Exception as e:
        logger.error(f"Помилка при отриманні даних з SaveEcoBot: {str(e)}")
    finally:
        db.close()
# Функція для переналаштування даних (очищення і перегенерація)
def reset_and_regenerate_data():
    """Очищення бази даних і перегенерація синтетичних даних"""
    db = get_db()
    try:
        # Видаляємо всі дані моніторингу
        db.query(MonitoringData).delete()
        db.commit()
        logger.info("Базу даних очищено. Запускаємо генерацію нових даних...")
        # Запускаємо генерацію синтетичних даних
        fetch_saveecobot_synthetic()
        logger.info("Дані успішно перегенеровано")
    except Exception as e:
        logger.error(f"Помилка при переналаштуванні даних: {str(e)}")
        db.rollback()
    finally:
        db.close()
# Функція для перевірки порогових значень і відправки повідомлень
def check_thresholds_and_send_notifications():
    """Перевірка порогових значень і відправка повідомлень"""
    db = get_db()
    try:
        # Отримати всі підписки
        subscriptions = db.query(UserSubscription).all()
        for subscription in subscriptions:
            # Отримати останні дані для точки моніторингу
            latest_data = db.query(MonitoringData).filter(
                MonitoringData.point_id == subscription.point_id,
                MonitoringData.parameter == subscription.parameter
            ).order_by(MonitoringData.timestamp.desc()).first()
            if latest_data and latest_data.value > subscription.threshold:
                # Отримати інформацію про точку моніторингу
                point = db.query(MonitoringPoint).filter(
                    MonitoringPoint.id == subscription.point_id
                ).first()
                if not point:
                    continue
                # Підготовка даних для повідомлення
                notification_data = {
                    "device_id": subscription.device_id,
                    "title": f"Увага! Рівень {latest_data.parameter} перевищує
встановлений поріг!",
                    "body": f"Станція: {point.name}, Поточне значення:
{latest_data.value} {latest_data.unit}, Поріг: {subscription.threshold}
{latest_data.unit}",
                    "data": {
                        "point_id": subscription.point_id,
                        "parameter": latest_data.parameter,
                        "value": latest_data.value,
```

```
                        "unit": latest_data.unit,
                        "threshold": subscription.threshold,
                        "station_name": point.name,
                        "latitude": point.latitude,
                        "longitude": point.longitude
                    }
                }
                # Відправка повідомлення через API
                try:
                    requests.post(
                        f"http://localhost:8000/api/notifications",
                        json=notification_data
                    )
                    logger.info(f"Відправлено повідомлення для {subscription.device_id}")
                except Exception as e:
                    logger.error(f"Помилка відправки повідомлення: {e}")
    except Exception as e:
        logger.error(f"Помилка при перевірці порогових значень: {e}")
    finally:
        db.close()
# Функція для регулярного оновлення синтетичних даних
def update_synthetic_data_task():
    """Фонова задача для регулярного оновлення синтетичних даних"""
    while True:
        db = get_db()
        try:
            logger.info("Запуск регулярного оновлення синтетичних даних...")
            # Отримуємо всі точки моніторингу
            points = db.query(MonitoringPoint).all()
            for point in points:
                # Отримуємо останні дані для кожного параметра
                latest_data = {}
                for param in ["PM2.5", "PM10", "NO2", "SO2", "O3", "CO"]:
                    data = db.query(MonitoringData).filter(
                        MonitoringData.point_id == point.id,
                        MonitoringData.parameter == param
                    ).order_by(MonitoringData.timestamp.desc()).first()
                    if data:
                        latest_data[param] = (data.value, data.unit)
                # Генеруємо нові значення для кожного параметра
                now = datetime.datetime.now()
                for param, (base_value, unit) in latest_data.items():
                    # Генеруємо нове значення
                    synthetic_value = generate_synthetic_data(base_value)
                    # Додаємо нове вимірювання
                    new_data = MonitoringData(
                        point_id=point.id,
                        parameter=param,
                        value=synthetic_value,
                        unit=unit,
                        timestamp=now
                    )
                    db.add(new_data)
            db.commit()
            logger.info(f"Успішно оновлено синтетичні дані для {len(points)} точок")
        except Exception as e:
            logger.error(f"Помилка при оновленні синтетичних даних: {str(e)}")
            db.rollback()
        finally:
            db.close()
        # Очікування перед наступним оновленням (15 хвилин)
        time.sleep(900)
# Функція для перевірки сповіщень
def check_notifications_task():
    """Фонова задача для перевірки порогових значень і відправки повідомлень"""
```

```
    while True:
        try:
            logger.info("Перевірка порогових значень і відправка повідомлень...")
            check_thresholds_and_send_notifications()
        except Exception as e:
            logger.error(f"Помилка у фоновій задачі сповіщень: {e}")
        # Очікування перед наступною перевіркою (5 хвилин)
        time.sleep(300)
# Функція для очищення старих даних
def cleanup_old_data_task():
    """Фонова задача для очищення старих даних"""
    while True:
        # Очікуємо день, перш ніж запустити перше очищення
        time.sleep(86400)  # 24 години
        db = get_db()
        try:
            # Видаляємо дані старші 30 днів
            cutoff_date = datetime.datetime.now() - datetime.timedelta(days=30)
            result = db.query(MonitoringData).filter(
                MonitoringData.timestamp < cutoff_date
            ).delete()
            db.commit()
            logger.info(f"Очищено {result} старих записів з бази даних")
        except Exception as e:
            logger.error(f"Помилка у фоновій задачі очищення даних: {e}")
            db.rollback()
        finally:
            db.close()
# API ендпоінти
@app.route("/api/points", methods=["GET"])
def get_all_points():
    """Отримання всіх точок моніторингу з їх останніми даними"""
    db = get_db()
    points = db.query(MonitoringPoint).all()
    result = []
    for point in points:
        # Отримати останні дані для кожної точки
        latest_data = db.query(MonitoringData).filter(
            MonitoringData.point_id == point.id
        ).order_by(MonitoringData.timestamp.desc()).limit(10).all()
        data_list = []
        for data in latest_data:
            data_list.append({
                "parameter": data.parameter,
                "value": data.value,
                "unit": data.unit,
                "timestamp": data.timestamp.isoformat()
            })
        result.append({
            "id": point.id,
            "name": point.name,
            "latitude": point.latitude,
            "longitude": point.longitude,
            "source_type": point.source_type,
            "data": data_list
        })
    return jsonify(result)
@app.route("/api/points/<int:point_id>", methods=["GET"])
def get_point_data(point_id):
    """Отримання даних конкретної точки моніторингу"""
    db = get_db()
    point = db.query(MonitoringPoint).filter(MonitoringPoint.id == point_id).first()
    if not point:
        return jsonify({"error": "Точку не знайдено"}), 404
    # Отримати останні дані для точки
```

```
    latest_data = db.query(MonitoringData).filter(
        MonitoringData.point_id == point.id
    ).order_by(MonitoringData.timestamp.desc()).limit(20).all()
    data_list = []
    for data in latest_data:
        data_list.append({
            "parameter": data.parameter,
            "value": data.value,
            "unit": data.unit,
            "timestamp": data.timestamp.isoformat()
        })
    result = {
        "id": point.id,
        "name": point.name,
        "latitude": point.latitude,
        "longitude": point.longitude,
        "source_type": point.source_type,
        "data": data_list
    }
    return jsonify(result)
@app.route("/api/data", methods=["GET"])
def get_all_data():
    """Отримання усіх даних з можливістю фільтрації"""
    db = get_db()
    # Отримання параметрів запиту
    parameter = request.args.get("parameter")
    min_value = request.args.get("min_value", type=float)
    max_value = request.args.get("max_value", type=float)
    start_date = request.args.get("start_date")
    end_date = request.args.get("end_date")
    source_type = request.args.get("source_type")
    limit = request.args.get("limit", 1000, type=int)
    offset = request.args.get("offset", 0, type=int)
    # Базовий запит
    query = db.query(
        MonitoringData.id,
        MonitoringData.parameter,
        MonitoringData.value,
        MonitoringData.unit,
        MonitoringData.timestamp,
        MonitoringPoint.name.label("station_name"),
        MonitoringPoint.latitude,
        MonitoringPoint.longitude,
        MonitoringPoint.source_type
    ).join(
        MonitoringPoint,
        MonitoringData.point_id == MonitoringPoint.id
    )
    # Застосування фільтрів
    if parameter:
        query = query.filter(MonitoringData.parameter == parameter)
    if min_value is not None:
        query = query.filter(MonitoringData.value >= min_value)
    if max_value is not None:
        query = query.filter(MonitoringData.value <= max_value)
    if source_type:
        query = query.filter(MonitoringPoint.source_type == source_type)
    if start_date:
        try:
            start_datetime = datetime.datetime.fromisoformat(start_date)
            query = query.filter(MonitoringData.timestamp >= start_datetime)
        except ValueError:
            return jsonify({"error": "Неправильний формат дати початку"}), 400
    if end_date:
        try:
```

```
            end_datetime = datetime.datetime.fromisoformat(end_date)
            query = query.filter(MonitoringData.timestamp <= end_datetime)
        except ValueError:
            return jsonify({"error": "Неправильний формат дати кінця"}), 400
    # Сортування за часом (спочатку нові)
    query = query.order_by(MonitoringData.timestamp.desc())
    # Пагінація
    total = query.count()
    query = query.offset(offset).limit(limit)
    # Виконання запиту
    results = query.all()
    # Форматування результатів
    data = []
    for row in results:
        data.append({
            "id": row.id,
            "parameter": row.parameter,
            "value": row.value,
            "unit": row.unit,
            "timestamp": row.timestamp.isoformat(),
            "station_name": row.station_name,
            "latitude": row.latitude,
            "longitude": row.longitude,
            "source_type": row.source_type
        })
    return jsonify(data)
@app.route("/api/statistics", methods=["GET"])
def get_statistics():
    """Отримання статистики по даним"""
    db = get_db()
    # Кількість точок моніторингу
    point_count = db.query(MonitoringPoint).count()
    # Кількість точок за типами джерел
    source_types = db.query(
        MonitoringPoint.source_type,
        func.count(MonitoringPoint.id).label("count")
    ).group_by(MonitoringPoint.source_type).all()
    # Середні значення параметрів
    avg_values = db.query(
        MonitoringData.parameter,
        func.avg(MonitoringData.value).label("avg_value"),
        func.min(MonitoringData.value).label("min_value"),
        func.max(MonitoringData.value).label("max_value"),
        func.count(MonitoringData.id).label("count")
    ).group_by(MonitoringData.parameter).all()
    # Формування результату
    result = {
        "total_points": point_count,
        "sources": {source_type: count for source_type, count in source_types},
        "parameters": {
            param: {
                "average": round(avg, 2),
                "min": round(min_val, 2),
                "max": round(max_val, 2),
                "measurements": count
            } for param, avg, min_val, max_val, count in avg_values
        }
    }
    return jsonify(result)
@app.route("/api/subscriptions", methods=["POST"])
def create_subscription():
    """Створення підписки на сповіщення"""
    db = get_db()
    data = request.json
```

```
    if not all(key in data for key in ["device_id", "point_id", "parameter",
"threshold"]):
        return jsonify({"error": "Відсутні обов'язкові поля"}), 400
    db_subscription = UserSubscription(
        device_id=data["device_id"],
        point_id=data["point_id"],
        parameter=data["parameter"],
        threshold=data["threshold"]
    )
    db.add(db_subscription)
    db.commit()
    db.refresh(db_subscription)
    return jsonify({"id": db_subscription.id, "message": "Підписку створено успішно"}),
201
@app.route("/api/notifications", methods=["POST"])
def send_notification():
    """Запис повідомлення у файл для імітації пуш-повідомлень"""
    data = request.json
    notifications_file = "notifications.json"
    try:
        if os.path.exists(notifications_file):
            with open(notifications_file, "r", encoding="utf-8") as f:
                notifications = json.load(f)
        else:
            notifications = []
        notifications.append({
            "device_id": data["device_id"],
            "title": data["title"],
            "body": data["body"],
            "data": data["data"],
            "timestamp": datetime.datetime.now().isoformat()
        })
        with open(notifications_file, "w", encoding="utf-8") as f:
            json.dump(notifications, f, ensure_ascii=False, indent=2)
        return jsonify({"status": "success", "message": "Повідомлення збережено"})
    except Exception as e:
        logger.error(f"Помилка збереження повідомлення: {e}")
        return jsonify({"status": "error", "message": str(e)})
@app.route("/api/regenerate_data", methods=["POST"])
def regenerate_data_endpoint():
    """Ендпоінт для запуску перегенерації даних"""
    # Запускаємо в окремому потоці
    thread = threading.Thread(target=reset_and_regenerate_data)
    thread.daemon = True
    thread.start()
    return jsonify({"status": "success", "message": "Запущено перегенерацію даних"})
@app.route("/api/cleanup", methods=["GET"])
def cleanup_old_data_endpoint():
    """Очищення старих даних з бази даних"""
    db = get_db()
    days = request.args.get("days", 30, type=int)
    try:
        # Визначаємо дату, старіше якої видаляємо дані
        cutoff_date = datetime.datetime.now() - datetime.timedelta(days=days)
        # Видаляємо старі дані
        result = db.query(MonitoringData).filter(
            MonitoringData.timestamp < cutoff_date
        ).delete()
        db.commit()
        return jsonify({
            "status": "success",
            "deleted_records": result,
            "message": f"Видалено {result} записів старіших за {days} днів"
        })
    except Exception as e:
```

```
        db.rollback()
        logger.error(f"Помилка при очищенні старих даних: {e}")
        return jsonify({"status": "error", "message": str(e)}), 500
# Запуск фонових задач
def start_background_tasks():
    # Перевіряємо наявність точок моніторингу
    db = get_db()
    try:
        # Перевіряємо наявність точок моніторингу
        point_count = db.query(MonitoringPoint).count()
        # Якщо точок немає або їх мало, запускаємо першочергове завантаження даних
        if point_count < 5:  # Якщо менше 5 точок, вважаємо, що даних немає
            logger.info("Запуск першочергового завантаження даних...")
            fetch_saveecobot_synthetic()
        else:
            # Перевіряємо наявність актуальних даних (за останню добу)
            last_day = datetime.datetime.now() - datetime.timedelta(days=1)
            data_count = db.query(MonitoringData).filter(
                MonitoringData.timestamp >= last_day
            ).count()
            if data_count < 10:  # Якщо менше 10 записів за добу, оновлюємо
                logger.info("Відсутні актуальні дані. Запуск оновлення синтетичних
даних...")
                reset_and_regenerate_data()
    except Exception as e:
        logger.error(f"Помилка при ініціалізації даних: {e}")
    finally:
        db.close()
    # Запуск потоків
    update_thread = threading.Thread(target=update_synthetic_data_task)
    update_thread.daemon = True
    update_thread.start()
    notification_thread = threading.Thread(target=check_notifications_task)
    notification_thread.daemon = True
    notification_thread.start()
    cleanup_thread = threading.Thread(target=cleanup_old_data_task)
    cleanup_thread.daemon = True
    cleanup_thread.start()
    logger.info("Фонові задачі запущено")
# Запуск сервера
if __name__ == "__main__":
    # Запуск фонових задач перед запуском сервера
    start_background_tasks()
    # Запуск Flask-сервера
    app.run(host="0.0.0.0", port=8000, debug=False)
```