НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету	**Завідувач кафедри комп'ютерних наук**
_____ **Ігор БОЛБОТ**	_____ **Белла ГОЛУБ**
(підпис)	(підпис)
«___» _____ 2026 р.	«___» _____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Система моніторингу екологічних параметрів транспортних потоків мегаполісів»

Спеціальність «122 Комп'ютерні науки»

Освітньо-професійна програма «Комп'ютерні науки»

Гарант освітньої програми

д.е.н., професор _____ **Роман РУДЕНСЬКИЙ**

(підпис)

Керівник бакалаврської кваліфікаційної роботи

к.т.н, доцент _____ **Ганна ВАЙГАН**

(підпис)

Виконав _____ **Анастасія КИПІЧ**

(підпис)

Київ-2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук

к.т.н., доцент _____ Белла ГОЛУБ
(підпис)

«___» _____ 2026 р.

ЗАВДАННЯ ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ ЗДОБУВАЧУ

Кипіч Анастасії Сергіївні
(прізвище, ім'я, по батькові)

Спеціальність «122 Комп'ютерні науки»
Освітньо-професійна програма «Комп'ютерні науки»

Тема бакалаврської кваліфікаційної роботи
«Система моніторингу екологічних параметрів транспортних потоків мегаполісів»
затверджена наказом від «06» січня 2026 р. № 46 «С»

Термін подання завершеної роботи на кафедру «_22__» травня 2026 р.

Вихідні дані до роботи: опис процесів екологічного моніторингу міського середовища та аналізу транспортного навантаження; вимоги до web-орієнтованої системи збору, візуалізації та прогнозування екологічних показників; структура екологічних і транспортних даних; технології Python, Flask, SQLite, HTML/CSS, JavaScript та Chart.js для реалізації інтерактивного web-застосунку EcoMonitor..

Перелік питань, що підлягають дослідженню: аналіз предметної області та постановка задачі; проєктування системи та інформаційного забезпечення; розроблення програмного забезпечення; тестування та впровадження системи

Перелік графічних документів (за необхідності).

Дата видачі завдання «__02__» ___лютого_____ 2025 р.

Керівник бакалаврської кваліфікаційної роботи _____ **Ганна ВАЙГАНГ**
(підпис)

Завдання взяв до виконання _____ **Анастасія КИПІЧ**
(підпис)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AQI – Air Quality Index – індекс якості повітря

API – Application Programming Interface – програмний інтерфейс застосунку

БД – база даних

ГДК – гранично допустима концентрація

ВООЗ – Всесвітня організація охорони здоров'я

CO – монооксид вуглецю (чадний газ)

CO_2 – діоксид вуглецю (вуглекислий газ)

CSS – Cascading Style Sheets – каскадні таблиці стилів

CSV – Comma-Separated Values – формат даних із роздільником-комою

CRUD – Create, Read, Update, Delete – базові операції з даними

DDL – Data Definition Language – мова визначення даних

ER – Entity-Relationship – модель "сутність–зв'язок"

FK – Foreign Key – зовнішній ключ

HTML – HyperText Markup Language – мова розмітки гіпертексту

HTTP – HyperText Transfer Protocol – протокол передачі гіпертексту

JS – JavaScript – мова програмування для веб-інтерфейсів

JSON – JavaScript Object Notation – формат обміну даними

OLS – Ordinary Least Squares – метод найменших квадратів

PDF – Portable Document Format – формат переносних документів

PK – Primary Key – первинний ключ

PM2.5 – Particulate Matter 2.5 – дрібнодисперсні частинки (діаметр $\leq 2{,}5$ мкм)

PM10 – Particulate Matter 10 – тверді частинки (діаметр ≤ 10 мкм)

REST – Representational State Transfer – архітектурний стиль веб-сервісів

SQL – Structured Query Language – мова структурованих запитів

SQLite – вбудована реляційна СУБД, що зберігає дані у єдиному файлі

СУБД – система управління базами даних

ВСТУП

Актуальність теми. Сучасні мегаполіси характеризуються високою концентрацією транспортних потоків, що супроводжується значним антропогенним навантаженням на навколишнє середовище. Автомобільний транспорт є одним із основних джерел забруднення атмосферного повітря, формування шумового навантаження та погіршення екологічного стану урбанізованих територій [1, 2]. За даними Всесвітньої організації охорони здоров'я, забруднення атмосферного повітря щороку спричиняє понад сім мільйонів передчасних смертей у світі, а транспортний сектор у багатьох містах формує значну частину шкідливих викидів у міському середовищі [3]. Особливої актуальності проблема набуває в умовах постійного зростання рівня автомобілізації, збільшення інтенсивності дорожнього руху та перевантаження транспортної інфраструктури великих міст.

Для України проблема екологічного впливу транспортних потоків також є надзвичайно актуальною. Кількість зареєстрованих транспортних засобів перевищує 10 мільйонів одиниць, при цьому темпи розвитку дорожньої інфраструктури не відповідають темпам зростання транспортного навантаження [4]. Наслідком цього є збільшення кількості транспортних заторів, підвищення концентрації шкідливих речовин у повітрі, погіршення акустичного стану міського середовища та зростання ризиків для здоров'я населення [5]. У щільно забудованих районах мегаполісів негативний вплив транспортних потоків посилюється через високу концентрацію населення, недостатню пропускну здатність дорожньої мережі та обмежені можливості оперативного екологічного контролю.

Важливим напрямом вирішення зазначених проблем є впровадження сучасних інформаційних систем моніторингу екологічних показників. На сьогодні існує низка платформ, призначених для збору та візуалізації даних про стан атмосферного повітря, зокрема OpenAQ, AQICN, Sensor.Community та SaveEcoBot [6–9]. Такі системи забезпечують доступ до даних екологічного

моніторингу, відображення показників якості повітря та формування базових аналітичних звітів. Проте більшість наявних рішень орієнтовані переважно на екологічні параметри та не враховують характеристики транспортних потоків як одного з ключових чинників формування забруднення [10].

Відсутність інтеграції екологічних показників із параметрами транспортного руху суттєво обмежує можливості комплексного аналізу міського середовища. Зокрема, ускладнюється визначення кореляційних залежностей між інтенсивністю транспортного потоку та рівнем забруднення атмосферного повітря, прогнозування екологічної ситуації на основі транспортних даних, а також формування аналітичних звітів для підтримки прийняття управлінських рішень [11, 12]. У зв’язку з цим виникає необхідність створення спеціалізованої інформаційної системи, що поєднуватиме засоби екологічного моніторингу, аналізу транспортних потоків та прогнозування змін екологічних показників у реальному часі.

Мета роботи – проєктування та розробка веб-орієнтованої системи моніторингу екологічних параметрів транспортних потоків мегаполісів, яка забезпечує збір, зберігання, обробку, візуалізацію та інтелектуальний аналіз даних про якість атмосферного повітря, рівень акустичного забруднення та характеристики транспортного потоку.

Для досягнення поставленої мети необхідно вирішити такі задачі:
– провести аналіз предметної області та визначити основні екологічні параметри транспортних потоків;
– виконати аналіз існуючих систем моніторингу екологічних показників;
– сформувати функціональні та нефункціональні вимоги до системи;
– спроєктувати архітектуру програмного забезпечення та структуру інформаційного забезпечення;
– реалізувати модулі збору, обробки, збереження та візуалізації даних;
– реалізувати засоби аналізу та прогнозування екологічних показників;
– провести тестування системи та оцінити результати її функціонування.

Об’єкт дослідження – процеси моніторингу та аналізу екологічних параметрів, пов’язаних із транспортними потоками у великих містах.

Предмет дослідження – методи та програмні засоби побудови інформаційних систем для автоматизованого збору, обробки, візуалізації та прогнозування екологічних показників, зумовлених транспортною активністю.

Методи та технології розробки. Для реалізації програмного забезпечення використано мову програмування Python 3 та мікрофреймворк Flask, що забезпечує побудову серверної частини веб-застосунку [13]. Для зберігання даних застосовано реляційну систему керування базами даних SQLite 3 [14]. Візуалізація екологічних показників та результатів аналізу реалізована за допомогою бібліотеки Chart.js [15], а картографічне відображення станцій моніторингу та транспортних об’єктів – із використанням бібліотеки Leaflet.js [16]. Для прогнозування рівня забруднення атмосферного повітря використано метод множинної лінійної регресії (OLS) [17]. Кореляційний аналіз здійснюється на основі обчислення коефіцієнта кореляції Пірсона [18]. Моделювання предметної області та проєктування структури системи виконано з використанням нотації UML [19].

Практичне значення отриманих результатів полягає у можливості використання розробленої системи для моніторингу екологічного стану транспортних зон мегаполісів, аналізу впливу транспортних потоків на якість атмосферного повітря та підтримки прийняття рішень щодо оптимізації транспортної інфраструктури і зменшення екологічного навантаження на міське середовище.

Структура та обсяг роботи. Бакалаврська кваліфікаційна робота складається зі вступу, чотирьох розділів, загальних висновків, списку використаних джерел та додатків.

У першому розділі проведено аналіз предметної області, досліджено особливості транспортних потоків мегаполісів як джерела екологічного впливу, систематизовано екологічні параметри, що підлягають моніторингу, виконано аналіз існуючих платформ екологічного моніторингу та сформульовано

постановку задачі з визначенням функціональних і нефункціональних вимог до системи.

У другому розділі виконано проєктування системи, розроблено трирівневу архітектуру програмного забезпечення, побудовано UML-діаграми варіантів використання, діяльності, класів, послідовності та розгортання, а також спроєктовано структуру реляційної бази даних, нормалізованої до третьої нормальної форми.

У третьому розділі описано процес розроблення програмного забезпечення, обґрунтовано вибір технологій та засобів реалізації, розроблено модулі збору, обробки та збереження даних, а також реалізовано веб-інтерфейс системи моніторингу з інтерактивними графіками, картою станцій та засобами формування аналітичних звітів.

У четвертому розділі проведено тестування системи, проаналізовано результати її функціонування, оцінено коректність роботи основних модулів та сформульовано рекомендації щодо впровадження і подальшої експлуатації розробленого програмного забезпечення.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Особливості транспортних потоків мегаполісів як джерела екологічного впливу

Транспортна система мегаполісу являє собою складний динамічний об'єкт, що характеризується високою інтенсивністю руху, нерівномірністю навантаження у просторі та часі, а також значним негативним впливом на довкілля. Саме автомобільний транспорт визнано одним із провідних чинників забруднення атмосферного повітря у великих містах. За даними Всесвітньої організації охорони здоров'я, забруднення повітря щорічно спричиняє понад сім мільйонів передчасних смертей у світі, при цьому значна частка забруднювачів потрапляє в атмосферу саме внаслідок роботи автомобільних двигунів [1].

Транспортний потік мегаполісу має низку характерних особливостей, що визначають специфіку його екологічного впливу. Передусім це добова циклічність: ранкові (7:00–10:00) та вечірні (17:00–20:00) години пік супроводжуються різким зростанням кількості транспортних засобів на дорогах, формуванням заторів та, відповідно, збільшенням обсягів шкідливих викидів. У позапіковий період та вночі інтенсивність руху суттєво знижується, що позитивно впливає на якість повітря [2].

Другою важливою характеристикою є просторова неоднорідність. Основні магістралі та перехрестя зазнають значно вищого навантаження порівняно з житловими кварталами, що створює виражену просторову диференціацію рівня забруднення. Дослідження Європейського агентства довкілля засвідчують, що концентрація діоксиду азоту безпосередньо біля автомагістралей може втричі перевищувати фонові значення, зафіксовані на відстані лише 200–300 метрів від проїжджої частини [2].

Третьою особливістю транспортних потоків мегаполісу є залежність обсягу викидів від режиму руху. Під час вільного руху зі швидкістю 50–70 км/год двигун працює в оптимальному режимі, і питомі викиди на один кілометр є мінімальними. Натомість рух у заторі (швидкість нижче 15–20 км/год) характеризується частими циклами розгону–гальмування та тривалою роботою двигуна на холостому ходу, що збільшує питомі викиди монооксиду вуглецю (CO) на 50–80%, а оксидів азоту – на 30–60% порівняно з режимом вільного руху [3].

Окремою складовою екологічного впливу транспортних потоків є акустичне забруднення. Шум від автомобільного транспорту формується трьома основними джерелами: роботою двигуна, контактом шин із дорожнім покриттям та аеродинамічними ефектами. Згідно з настановами Всесвітньої організації охорони здоров'я, середньодобовий рівень транспортного шуму не повинен перевищувати 53 дБ, тоді як фактичні вимірювання на магістральних вулицях великих міст фіксують значення в діапазоні 70–85 дБ [4]. Тривалий вплив такого рівня шуму призводить до порушень сну, зниження когнітивних функцій та підвищення ризику серцево-судинних захворювань.

Ще одним аспектом є вплив складу транспортного потоку. Вантажні автомобілі та автобуси з дизельними двигунами генерують значно більші обсяги твердих частинок PM2.5 та PM10, ніж легкові автомобілі з бензиновими двигунами. Водночас зростання частки електромобілів у транспортному потоці поступово змінює структуру викидів, зменшуючи обсяги газоподібних забруднювачів, але зберігаючи проблему дрібнодисперсних частинок від зносу шин та гальмівних колодок [5].

В Україні проблема транспортного забруднення є особливо гострою для таких мегаполісів як Київ, Харків, Одеса та Дніпро. За даними Державної служби статистики, кількість зареєстрованих автомобілів в Україні станом на 2023 рік перевищила 10 мільйонів, причому значна їх частина зосереджена саме у великих містах. Зростання автомобілізації випереджає розвиток дорожньої

інфраструктури, що посилює проблему заторів та відповідно збільшує екологічне навантаження [6].

Класифікація забруднювачів атмосферного повітря, що генеруються автомобільним транспортом, передбачає поділ на первинні та вторинні. Первинні забруднювачі надходять безпосередньо від джерела: вихлопні гази двигунів внутрішнього згоряння містять монооксид вуглецю, оксиди азоту, діоксид вуглецю та леткі органічні сполуки. Вторинні забруднювачі утворюються в атмосфері внаслідок хімічних реакцій між первинними речовинами та компонентами повітря під дією сонячного випромінювання – зокрема, тропосферний озон (O_3), який є потужним подразником дихальних шляхів [7].

Окремою категорією є забруднювачі не вихлопного походження. Знос шин генерує мікрочастинки каучуку та сажі, знос гальмівних колодок – дрібнодисперсні частинки металів (мідь, залізо, барій), а знос дорожнього покриття та ресуспензія (повторне піднімання осілого пилу колесами) формують значну частку PM10 у міському повітрі. Дослідження Timmers та Achten [5] показали, що навіть повна електрифікація автопарку не усуне проблему твердих частинок, оскільки не вихлопні викиди становлять від 50 до 85% загального обсягу PM10 та від 10 до 40% PM2.5 залежно від типу дорожнього покриття та швидкості руху.

Сезонний фактор також суттєво впливає на рівень транспортного забруднення. У холодну пору року двигуни потребують тривалішого прогріву, що збільшує обсяг викидів CO під час перших хвилин роботи. Влітку, навпаки, висока інтенсивність сонячного випромінювання прискорює фотохімічні реакції та утворення тропосферного озону [23].

Вплив топографії міста на розподіл транспортного забруднення також заслуговує на увагу. Вулиці-каньйони – вузькі вулиці, оточені високими будівлями – характеризуються обмеженою вентиляцією, що призводить до накопичення забруднювачів на рівні пішоходів. Дослідження, проведені у містах Європи, показують, що концентрація NO_2 у вуличних каньйонах може вдвічі

перевищувати значення, виміряні на відкритих площах за тих самих транспортних умов [24]. Для Києва це актуально для таких вулиць як Хрещатик, Саксаганського, Велика Васильківська, де висота забудови значно перевищує ширину проїжджої частини.

Електромобілі, частка яких у транспортному потоці поступово зростає, змінюють структуру викидів, але не усувають проблему повністю. Відсутність прямих вихлопних газів компенсується збільшеною масою транспортного засобу (через важкі акумуляторні батареї), що інтенсифікує знос шин та дорожнього покриття. Крім того, рекуперативне гальмування, характерне для електромобілів, зменшує знос гальмівних колодок, проте цей ефект частково нівелюється збільшенням маси.

Міжнародний досвід управління транспортним забрудненням включає кілька підходів. Зони низьких викидів (Low Emission Zones, LEZ), впроваджені у Лондоні, Берліні, Амстердамі та інших містах, обмежують в'їзд найбільш забруднюючих транспортних засобів у центральні райони. Дослідження ефективності LEZ у Лондоні показало зниження концентрації PM10 на 12–15% та NO_2 на 8–10% протягом перших трьох років функціонування [25]. Впровадження аналогічних зон в українських мегаполісах потребує саме тих даних, які здатна надати система моніторингу: просторовий розподіл забруднення, його зв'язок із інтенсивністю трафіку та ефективність вжитих заходів.

1.2 Аналіз екологічних параметрів та існуючих засобів моніторингу

Ефективний моніторинг екологічного впливу транспортних потоків передбачає контроль двох груп параметрів: показників якості атмосферного повітря та характеристик самого транспортного потоку. Саме одночасний аналіз цих груп дає змогу встановити причинно-наслідкові зв'язки між інтенсивністю руху та рівнем забруднення довкілля.

Екологічні параметри. Серед забруднювачів атмосферного повітря, що безпосередньо пов'язані з роботою автотранспорту, виділяють кілька ключових речовин (табл. 1.1). Дрібнодисперсні тверді частинки PM2.5 (діаметр менше 2,5 мкм) здатні проникати глибоко в легені та потрапляти у кров'яне русло, що підвищує ризик серцево-судинних та респіраторних захворювань. Тверді частинки PM10 (діаметр менше 10 мкм) осідають у верхніх дихальних шляхах і також становлять загрозу для здоров'я [7].

Таблиця 1.1

Екологічні параметри, що підлягають моніторингу

Параметр	Позначення	Одиниця	ГДК (середньодобова)	Основне джерело
Дрібнодисперсні частинки	PM2.5	мкг/м³	25	Знос шин, гальм, двигуни
Тверді частинки	PM10	мкг/м³	50	Знос дороги, ресуспензія пилу
Діоксид азоту	NO_2	мкг/м³	40	Високотемпературне горіння палива
Монооксид вуглецю	CO	мг/м³	2,0	Неповне згоряння палива
Діоксид вуглецю	CO_2	ppm	500	Продукт повного згоряння
Рівень шуму	–	дБ	65	Двигун, шини, аеродинаміка

Діоксид азоту (NO_2) утворюється при високотемпературному горінні палива в двигунах внутрішнього згоряння. Ця речовина подразнює дихальні шляхи та сприяє утворенню тропосферного озону – вторинного забруднювача, шкідливого для здоров'я людини та рослин. Монооксид вуглецю (CO) є продуктом неповного згоряння палива і в підвищених концентраціях порушує транспортну функцію гемоглобіну. Діоксид вуглецю (CO_2), хоча й не є токсичним у типових атмосферних концентраціях, є основним парниковим газом і його обсяги безпосередньо залежать від інтенсивності руху [3].

Значення гранично допустимих концентрацій (ГДК), наведені в таблиці 1.1, визначені відповідно до рекомендацій ВООЗ [1] та чинного законодавства України, зокрема Закону «Про охорону атмосферного повітря» [8]. Окрім хімічних забруднювачів, додатково фіксуються температура та відносна вологість повітря, оскільки ці метеорологічні фактори впливають на розсіювання та трансформацію забруднювачів.

Параметри транспортного потоку. Для встановлення кореляції між інтенсивністю руху та рівнем забруднення необхідно реєструвати характеристики транспортного потоку. До основних характеристик транспортного потоку, що використовуються під час аналізу екологічного впливу міського транспорту, належать інтенсивність руху, середня швидкість транспортних засобів та рівень заторності дорожньої мережі, характеристика яких наведена в таблиці 1.2.

Таблиця 1.2

Параметри транспортного потоку

Параметр	Одиниця	Характеристика
Інтенсивність потоку	авто/год	Кількість транспортних засобів, що проїхали через перетин за годину
Середня швидкість	км/год	Середня швидкість руху на контрольній ділянці
Рівень заторності	бали (1–10)	Інтегральна оцінка завантаженості: 1 – вільний рух, 10 – повна зупинка

Огляд існуючих засобів моніторингу. На сьогодні функціонує низка платформ, що забезпечують моніторинг якості атмосферного повітря. Для визначення місця розроблюваної системи серед наявних рішень проведено порівняльний аналіз чотирьох найбільш поширених платформ, що мають відкритий або частково відкритий доступ до даних.

OpenAQ [9] – глобальна платформа відкритих даних про якість повітря, що агрегує інформацію з державних станцій моніторингу понад 100 країн. Платформа надає безкоштовний API та відкритий вихідний код. Серед переваг – стандартизований формат даних, широке географічне покриття та активна

спільнота розробників. Проте OpenAQ не містить жодних даних про транспортні потоки, не пропонує аналітичних інструментів і не має функцій прогнозування чи генерації звітів.

AQICN (World Air Quality Index) [10] – проєкт, що візуалізує індекс якості повітря для міст усього світу. Платформа має зручну картографічну візуалізацію та базову систему сповіщень про перевищення норм. Водночас аналітичні можливості обмежені переглядом поточних і простих історичних графіків. Безкоштовний API має суттєві обмеження за кількістю запитів. Кореляційний аналіз із транспортними даними не передбачено.

Sensor.Community [11] – громадський проєкт, що об'єднує мережу низьковартісних сенсорів якості повітря, встановлених волонтерами по всьому світу. Перевагою є висока просторова роздільна здатність у окремих європейських містах та повна відкритість даних. Основні недоліки – нерівномірне покриття (в Україні мережа розріджена), відсутність систематичної верифікації даних та обмежені аналітичні інструменти без можливості побудови кореляцій з транспортними показниками.

SaveEcoBot [12] – українська платформа моніторингу якості повітря, що агрегує дані як із державних, так і з громадських станцій на території України. Має зручний інтерфейс українською мовою, картографічну візуалізацію та базову систему сповіщень. Серед недоліків – відсутність інтеграції з транспортними даними, обмежені можливості аналітики, відсутність прогнозування та генерації комплексних звітів.

Для визначення функціональних можливостей та обмежень сучасних систем екологічного моніторингу доцільно провести порівняльний аналіз найбільш поширених платформ, що використовуються для збору, обробки та візуалізації даних про стан атмосферного повітря. Такий аналіз дозволяє оцінити рівень підтримки аналітичних функцій, можливості інтеграції транспортних параметрів, наявність засобів прогнозування та формування звітності, а також визначити напрями вдосконалення існуючих рішень. Порівняльну характеристику сучасних платформ моніторингу наведено в таблиці 1.3.

Таблиця 1.3

Порівняльний аналіз існуючих платформ моніторингу

Критерій	OpenAQ	AQICN	Sensor. Community	SaveEco Bot	Eco Monitor
Відкритий доступ	Так (API)	Частково	Так	Так	Так
Дані трафіку	Ні	Ні	Ні	Ні	Так
Кореляційний аналіз	Ні	Ні	Ні	Ні	Так
Прогнозування	Ні	Обмежено	Ні	Ні	Так
Генерація звітів	Ні	Ні	Ні	Обмежено	Так
Карта станцій	Так	Так	Так	Так	Так
Алерти ГДК	Ні	Так	Так	Так	Так
Покриття UA	Обмежене	Середнє	Низьке	Високе	Локальне
Імпорт/експорт	API	Ні	CSV	Ні	CSV

Як видно з табл. 1.3, жодна з розглянутих платформ не поєднує моніторинг якості повітря з аналізом транспортних потоків. Відсутні функції кореляційного аналізу залежності забруднення від інтенсивності руху, прогнозування рівня забруднення на основі транспортних даних та генерації структурованих звітів із повною статистикою. Ці обмеження підтверджують доцільність розробки власного рішення – системи EcoMonitor, яка усуває виявлені недоліки.

У науковій літературі проблематику моніторингу транспортного забруднення висвітлено у працях Gulia S. та співавторів [13], де виконано систематичний огляд методів контролю якості повітря вздовж автомагістралей і класифіковано підходи на стаціонарні, мобільні та сенсорні. Yi W. та співавтори [14] досліджують точність та калібрування низьковартісних сенсорів і визнають їх прийнятність для задач скринінгу та виявлення просторових закономірностей забруднення. У роботі Zheng Y. та співавторів [15] запропоновано методи аналізу міських даних для прогнозування якості повітря, зокрема із застосуванням лінійної регресії як базового методу для короткострокового прогнозу за наявності достатнього обсягу навчальних даних.

Система EcoMonitor проєктується як платформа для інтеграції даних стаціонарного моніторингу (як референтного, так і низьковартісного) із транспортними показниками. На відміну від PurpleAir або Sensor.Community, які обмежуються збором та відображенням екологічних даних, EcoMonitor забезпечує кореляційний аналіз із характеристиками транспортного потоку, прогнозування рівня забруднення та генерацію структурованих звітів.

Застосування індексу якості повітря (AQI) є поширеною практикою для комунікації результатів моніторингу із широкою аудиторією. Різні країни використовують дещо відмінні шкали AQI: американська EPA AQI базується на шести категоріях від 0 до 500, європейський CAQI – на п'яти категоріях від 0 до 100, а китайська AQI-шкала має власну градацію. У системі EcoMonitor реалізовано спрощену п'ятирівневу шкалу, базовану на концентрації PM2.5 відповідно до рекомендацій ВООЗ [1]: добре (≤12 мкг/м³), прийнятно (≤25), помірно (≤50), погано (≤75), небезпечно (>75). Кожному рівню відповідає колір візуалізації, що використовується на карті та дашборді.

1.3 Постановка задачі та вимоги до системи

Результати аналізу предметної області, проведеного у попередніх підрозділах, підтвердили доцільність розробки веб-орієнтованої системи моніторингу екологічних параметрів транспортних потоків мегаполісів EcoMonitor. Необхідність створення такого програмного рішення обумовлена постійним зростанням транспортного навантаження у великих містах, підвищенням рівня забруднення атмосферного повітря та потребою у своєчасному отриманні аналітичної інформації для оцінювання екологічного стану міського середовища [3], [7]. Дослідження існуючих інформаційних платформ екологічного моніторингу показало, що більшість із них орієнтована лише на відображення окремих показників якості повітря без комплексного врахування транспортних характеристик, аналізу взаємозв’язків між параметрами та засобів прогнозування зміни екологічного стану. Це ускладнює

процес оперативного прийняття рішень та знижує ефективність використання систем моніторингу в умовах динамічного міського середовища.

Відповідно до сформульованої у вступі мети дослідження, розроблювана система повинна забезпечувати автоматизований збір, накопичення, зберігання, візуалізацію та аналітичну обробку даних про якість атмосферного повітря, рівень акустичного забруднення та характеристики транспортного потоку. Об’єктом дослідження є процеси моніторингу та аналізу екологічних параметрів, пов’язаних із транспортними потоками великих міст, а предметом дослідження виступають методи та програмні засоби побудови інформаційних систем для автоматизованого збору, обробки, візуалізації та прогнозування екологічних показників, зумовлених транспортною активністю. Таке формулювання відповідає загальній концепції кваліфікаційної роботи та забезпечує узгодженість між теоретичною та практичною складовими дослідження.

Під час проєктування системи EcoMonitor основна увага приділялась побудові єдиного інформаційного середовища, у якому екологічні та транспортні дані інтегруються в централізовану систему аналізу. Передбачено, що застосунок здійснює приймання інформації від станцій моніторингу, виконує її перевірку, структуризацію та подальше збереження у базі даних із можливістю оперативного формування графіків, картографічних представлень і прогнозних оцінок. Загальну логіку взаємодії основних компонентів системи наведено на рисунку 1.1.

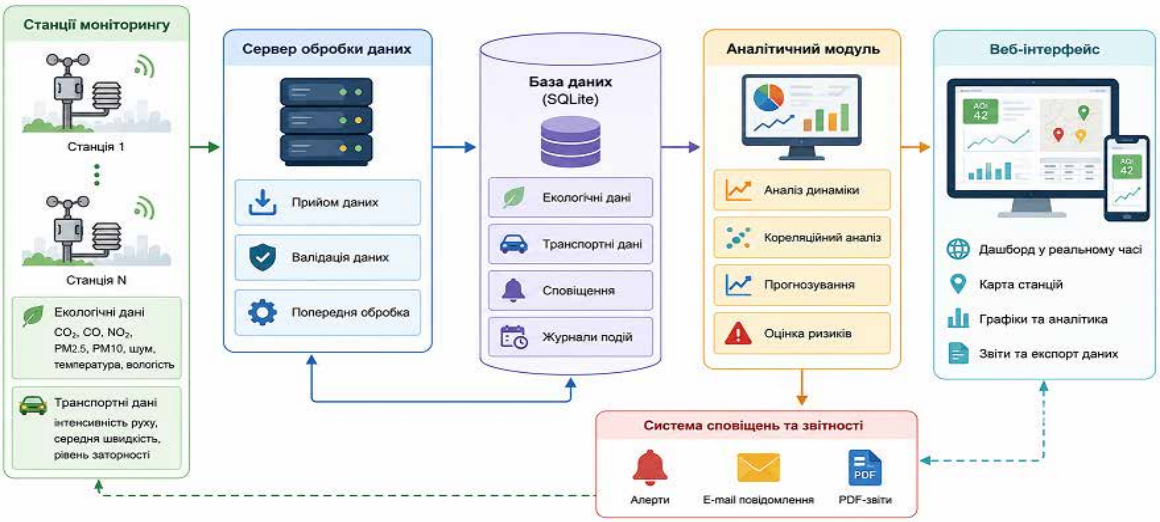

Рис. 1.1 Загальна схема функціонування системи EcoMonitor

Наведена схема демонструє централізовану модель функціонування системи, у якій серверна частина забезпечує приймання та обробку даних, аналітичний модуль виконує аналіз і прогнозування, а веб-інтерфейс реалізує інтерактивну взаємодію користувача з інформацією.

Функціональні вимоги формувались з урахуванням можливостей реалізованого веб-застосунку, створеного із використанням Flask, SQLite, JavaScript та бібліотек інтерактивної візуалізації даних. Система підтримує екологічний моніторинг у режимі реального часу, аналітичне опрацювання інформації та автоматизоване формування звітів. Основні функціональні характеристики системи наведено у табл. 1.4.

Таблиця 1.4

Функціональні вимоги до системи EcoMonitor

ID	Вимога	Опис
FR-01	Збір екологічних даних	Реєстрація CO_2, CO, NO_2, PM2.5, PM10, шуму, температури, вологості від станцій
FR-02	Збір транспортних даних	Фіксація кількості авто/год, швидкості руху, рівня заторності
FR-03	Дашборд реального часу	KPI-картки, таблиця стану станцій, графіки за добу
FR-04	Інтерактивна карта	Відображення станцій на карті з кольоровою індикацією AQI
FR-05	Аналітика	Графіки динаміки параметрів, порівняння станцій, вибір періоду
FR-06	Кореляційний аналіз	Scatter-plot залежності забруднення від трафіку, коефіцієнт Пірсона
FR-07	Прогнозування	Прогноз рівня забруднення на 24 год методом лінійної регресії
FR-08	Система алертів	Автоматичне порівняння з ГДК, три рівні: увага, небезпека, критично
FR-09	Генерація PDF-звітів	Звіти за добу/тиждень/місяць з таблицями та переліком алертів
FR-10	Імпорт/експорт CSV	Завантаження зовнішніх даних та вивантаження результатів

Наведені функціональні вимоги визначають логіку взаємодії користувача із системою та формують основу реалізації серверної логіки, структури бази даних і механізмів візуалізації інформації. Особливе значення для системи має модуль аналітичної обробки, який забезпечує можливість оцінювання динаміки екологічних показників та виявлення залежностей між транспортним навантаженням і станом атмосферного повітря.

Окрім функціональних характеристик, під час проєктування системи були визначені нефункціональні вимоги, що стосуються продуктивності, адаптивності, сумісності та надійності роботи програмного забезпечення. Їх формування здійснювалось з урахуванням необхідності стабільної роботи веб-застосунку при збільшенні обсягів екологічних даних та кількості станцій моніторингу. Основні нефункціональні вимоги наведено у таблиці 1.5.

Таблиця 1.5

Нефункціональні вимоги до системи EcoMonitor

ID	Категорія	Опис
NFR-01	Продуктивність	Час відгуку не більше 3 с для запитів до БД обсягом до 100 000 записів
NFR-02	Адаптивність	Інтерфейс коректно відображається на екранах від 375 px до 2560 px
NFR-03	Кросплатформність	Робота під Windows 10+, macOS 12+, Linux (Ubuntu 20.04+)
NFR-04	Доступність	Доступ через браузер без встановлення додаткового ПЗ на клієнті
NFR-05	Надійність	Цілісність даних забезпечується транзакціями СУБД та валідацією введення
NFR-06	Масштабованість	Архітектура передбачає можливість збільшення кількості станцій

Сформована сукупність функціональних та нефункціональних вимог визначає технічну основу розроблюваної системи та забезпечує узгодженість між поставленою метою дослідження, архітектурними рішеннями й програмною реалізацією веб-застосунку EcoMonitor. Отримані результати постановки задачі

використовуються для подальшого проєктування структури бази даних, серверної частини системи, модулів аналітичної обробки та засобів інтерактивної візуалізації екологічних показників.

1.4 Висновки до розділу 1

У першому розділі проведено аналіз предметної області моніторингу екологічних параметрів транспортних потоків мегаполісів. Визначено основні особливості транспортних потоків як джерела екологічного впливу: добову циклічність інтенсивності руху, просторову неоднорідність розподілу забруднення та залежність обсягу викидів від режиму руху. Встановлено, що рух у заторі збільшує питомі викиди CO на 50–80% та NO_2 на 30–60% порівняно з вільним рухом.

Систематизовано перелік екологічних параметрів (PM2.5, PM10, NO_2, CO, CO_2, рівень шуму) та параметрів транспортного потоку (інтенсивність, швидкість, заторність), що підлягають моніторингу. Проведено порівняльний аналіз чотирьох існуючих платформ (OpenAQ, AQICN, Sensor.Community, SaveEcoBot), який виявив спільний недолік – відсутність інтеграції екологічних даних із характеристиками транспортних потоків, а також відсутність кореляційного аналізу, прогнозування та генерації комплексних звітів.

Сформульовано постановку задачі, визначено мету, об'єкт та предмет дослідження, перелік із семи задач. Специфіковано десять функціональних (FR-01–FR-10) та шість нефункціональних (NFR-01–NFR-06) вимог до системи EcoMonitor, що слугуватимуть основою для подальшого проєктування.

2 ПРОЄКТУВАННЯ СИСТЕМИ ТА ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Архітектура та функціональна структура системи

Архітектуру системи моніторингу екологічних параметрів транспортних потоків EcoMonitor сформовано з урахуванням необхідності обробки екологічних і транспортних даних у режимі реального часу, підтримки інтерактивної візуалізації та стабільної роботи веб-застосунку при зміні обсягів інформації. Програмне забезпечення побудовано за трирівневою моделлю, що передбачає поділ системи на клієнтський рівень, серверний рівень та рівень даних [16].

Клієнтський рівень реалізовано у вигляді веб-інтерфейсу, що функціонує у браузері користувача. Формування сторінок здійснюється за допомогою HTML, CSS та шаблонізатора Jinja2, а для побудови графіків використано бібліотеку Chart.js. Відображення станцій моніторингу та просторових характеристик екологічних показників реалізовано із використанням бібліотеки Leaflet.js, яка забезпечує підтримку інтерактивної карти та відображення поточних параметрів станцій. Обмін даними між клієнтською та серверною частинами системи здійснюється через HTTP-запити у форматі JSON.

Серверний рівень реалізовано мовою Python із використанням мікрофреймворку Flask 3.0, який забезпечує маршрутизацію HTTP-запитів, обробку REST API та формування HTML-сторінок веб-застосунку [13]. Архітектура серверної частини побудована за модульним принципом і включає модулі обробки даних, аналітики, генерації сповіщень, прогнозування та формування PDF-звітів. Для прогнозування рівня забруднення використано метод множинної лінійної регресії (OLS) [17].

Рівень даних представлений реляційною системою керування базами даних SQLite 3, що забезпечує збереження інформації у файлі ecomonitor.db. Використання SQLite обумовлене підтримкою транзакцій ACID, інтеграцією у

стандартну бібліотеку Python та достатнім рівнем продуктивності для задач екологічного моніторингу [17]. Такий підхід дозволив спростити розгортання системи та зменшити вимоги до серверного середовища.

Архітектурну структуру системи та взаємозв'язки між її основними компонентами наведено на рисунку 2.1.

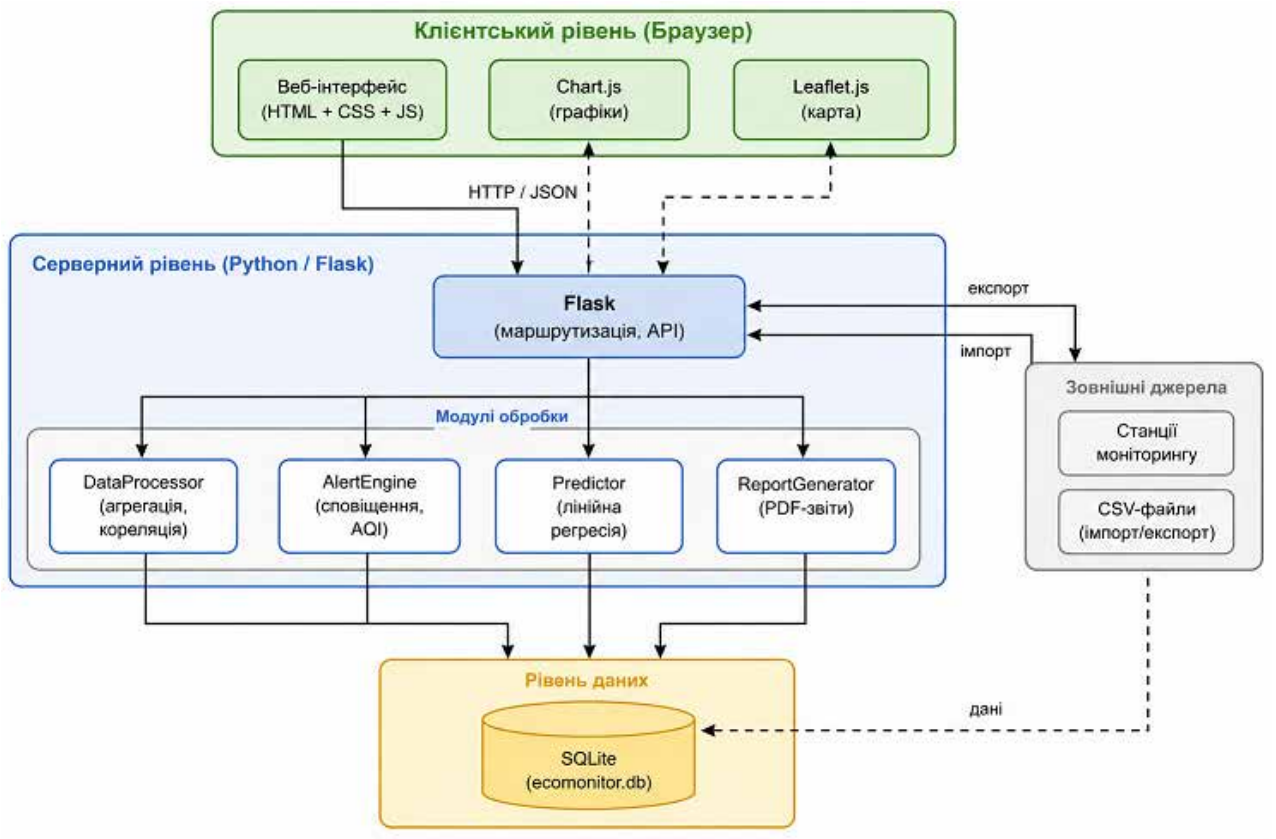

Рис. 2.1 Архітектурна діаграма системи EcoMonitor

Побудована архітектура забезпечує логічний розподіл функцій між компонентами системи та дозволяє реалізувати незалежну модифікацію окремих модулів без зміни загальної структури застосунку. Використання модульного підходу також спрощує подальше масштабування системи та інтеграцію додаткових засобів аналізу даних.

Для формалізації функціональної структури програмного забезпечення використано діаграму варіантів використання UML, яка відображає взаємодію користувачів із системою та визначає основні сценарії її експлуатації. У системі

передбачено декілька категорій користувачів із різними рівнями доступу до функціональних можливостей застосунку.

Функціональну структуру системи визначено на основі діаграми варіантів використання (Use Case), яка ідентифікує трьох акторів та десять основних варіантів використання. Актор «Оглядач» має доступ до перегляду дашборду, карти та журналу сповіщень. Актор «Аналітик» додатково використовує модулі аналітики, кореляційного аналізу, прогнозування, генерації звітів та експорту даних. Актор «Адміністратор» має повний доступ, включаючи управління станціями моніторингу та імпорт зовнішніх даних.

Загальну структуру взаємодії користувачів із системою наведено на рисунку 2.2.

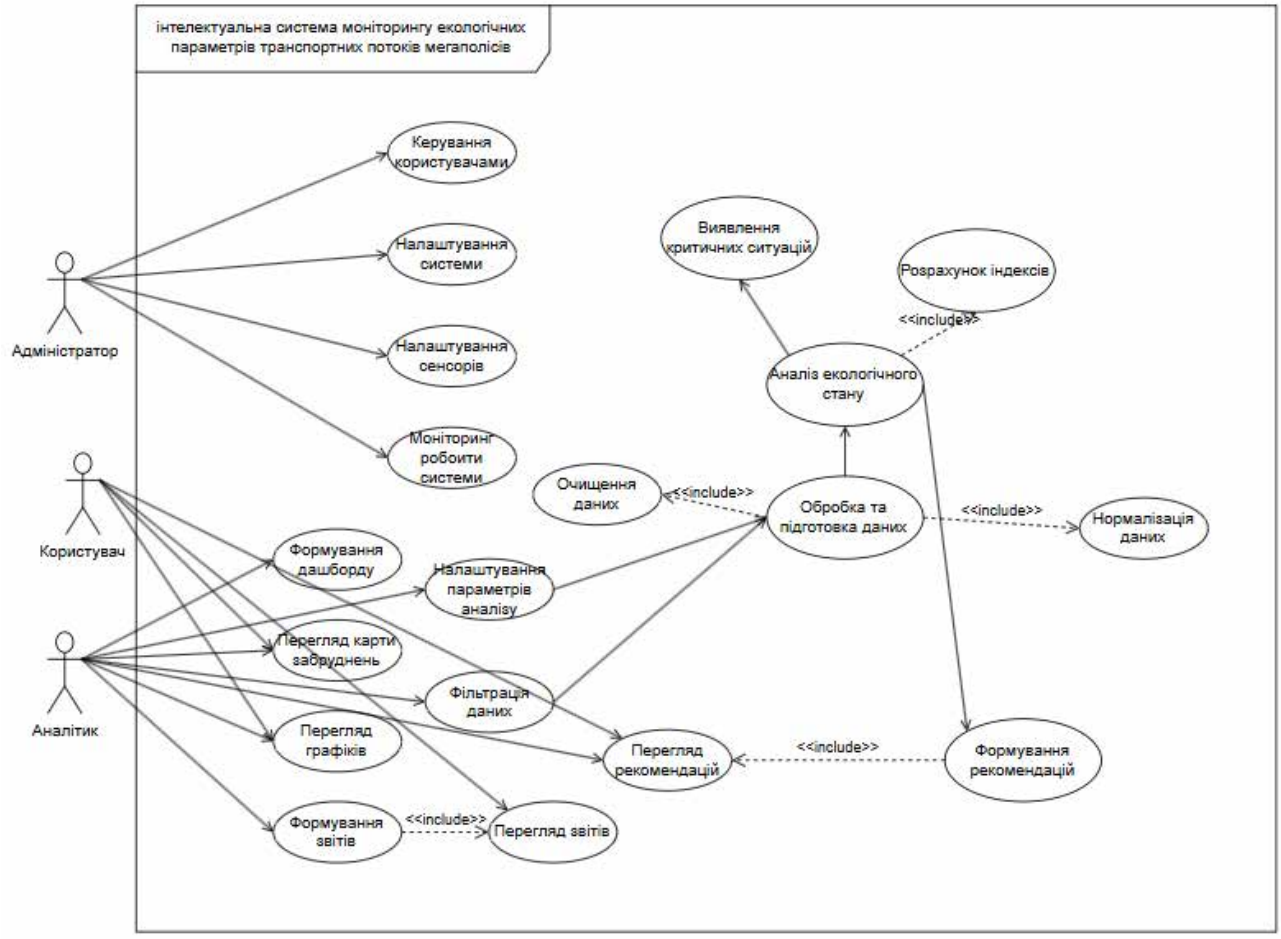

Рис. 2.2 Діаграма варіантів використання системи EcoMonitor

Діаграма демонструє диференційований підхід до організації доступу користувачів до функцій системи. Користувачі базового рівня отримують доступ

до інформаційних компонентів, зокрема перегляду дашборду, карти станцій моніторингу та графічної візуалізації екологічних параметрів. Аналітичний рівень доступу передбачає використання модулів кореляційного аналізу, прогнозування та генерації звітів. Адміністративний рівень забезпечує керування конфігурацією системи, параметрами станцій моніторингу та операціями імпорту й експорту даних.

Під час проєктування системи особлива увага приділялась організації взаємодії між програмними компонентами при розгортанні застосунку. Система функціонує за клієнт-серверною моделлю та складається з двох основних вузлів: клієнтського вузла, представленого веб-браузером користувача, і серверного вузла, на якому розгорнуті Flask-застосунок та база даних SQLite. Передавання інформації між вузлами здійснюється за протоколом HTTP із використанням JSON-структур. Діаграму розгортання системи наведено на рисунку 2.3.

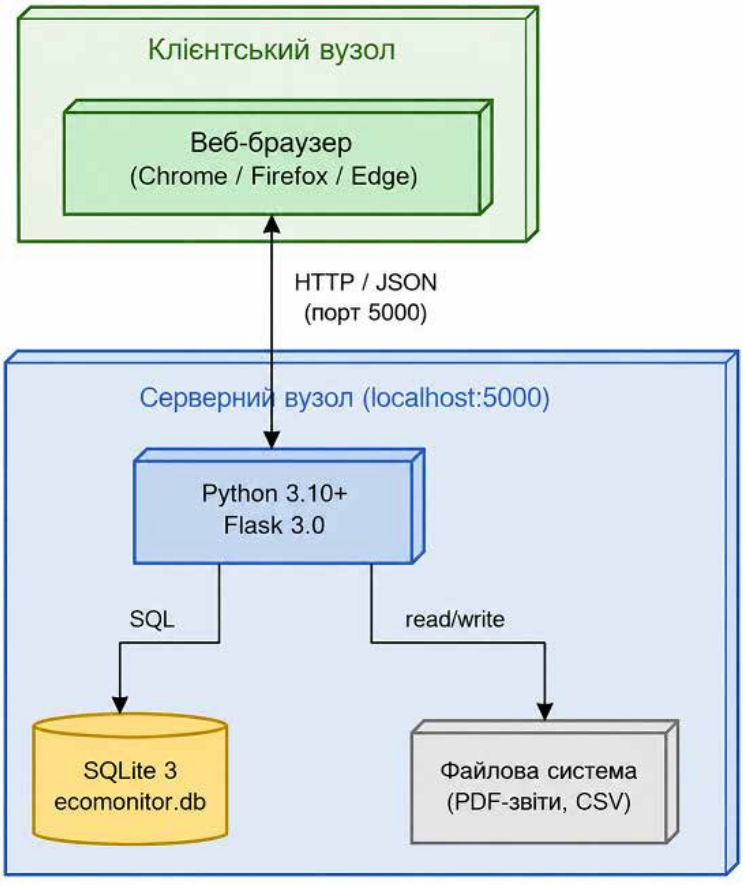

Рис. 2.3 Діаграма розгортання системи

Наведена діаграма відображає взаємодію програмних компонентів під час виконання застосунку та демонструє централізований характер обробки екологічних і транспортних даних. Така організація системи забезпечує спрощення адміністрування, зменшення вимог до клієнтського середовища та підтримку роботи застосунку через стандартний веб-браузер без необхідності встановлення додаткового програмного забезпечення.

2.2 Моделювання процесів і потоків даних

Моделювання процесів обробки даних у системі EcoMonitor виконано із використанням UML (Unified Modeling Language), що забезпечує формалізоване представлення логіки функціонування програмного забезпечення, взаємодії компонентів та потоків передавання інформації [18]. Для опису поведінки системи застосовано діаграму діяльності, діаграму послідовності, діаграму класів та блок-схему алгоритму розрахунку індексу якості повітря. Використання графічних моделей дозволило формалізувати ключові процеси функціонування застосунку та визначити механізми взаємодії між програмними модулями.

Діаграма діяльності (Activity Diagram). На рис. 2.4 представлено діаграму діяльності, що моделює основний процес обробки даних у системі – від отримання вимірювань до оновлення інтерфейсу. Процес розпочинається з надходження даних від станції моніторингу (або імпорту з CSV-файлу). На першому етапі виконується валідація даних: перевіряється наявність обов'язкових полів, коректність діапазонів значень та формат часових міток.

У разі успішної валідації дані зберігаються у відповідних таблицях бази даних (ecological_readings для екологічних показників, traffic_data для характеристик транспортного потоку). Після збереження виконуються два паралельних процеси: модуль AlertEngine порівнює отримані значення з пороговими (ГДК) та за потреби створює записи у таблиці alerts, а модуль

DataProcessor оновлює агреговану статистику. Після завершення обох гілок оновлюється інтерфейс дашборду та карти.

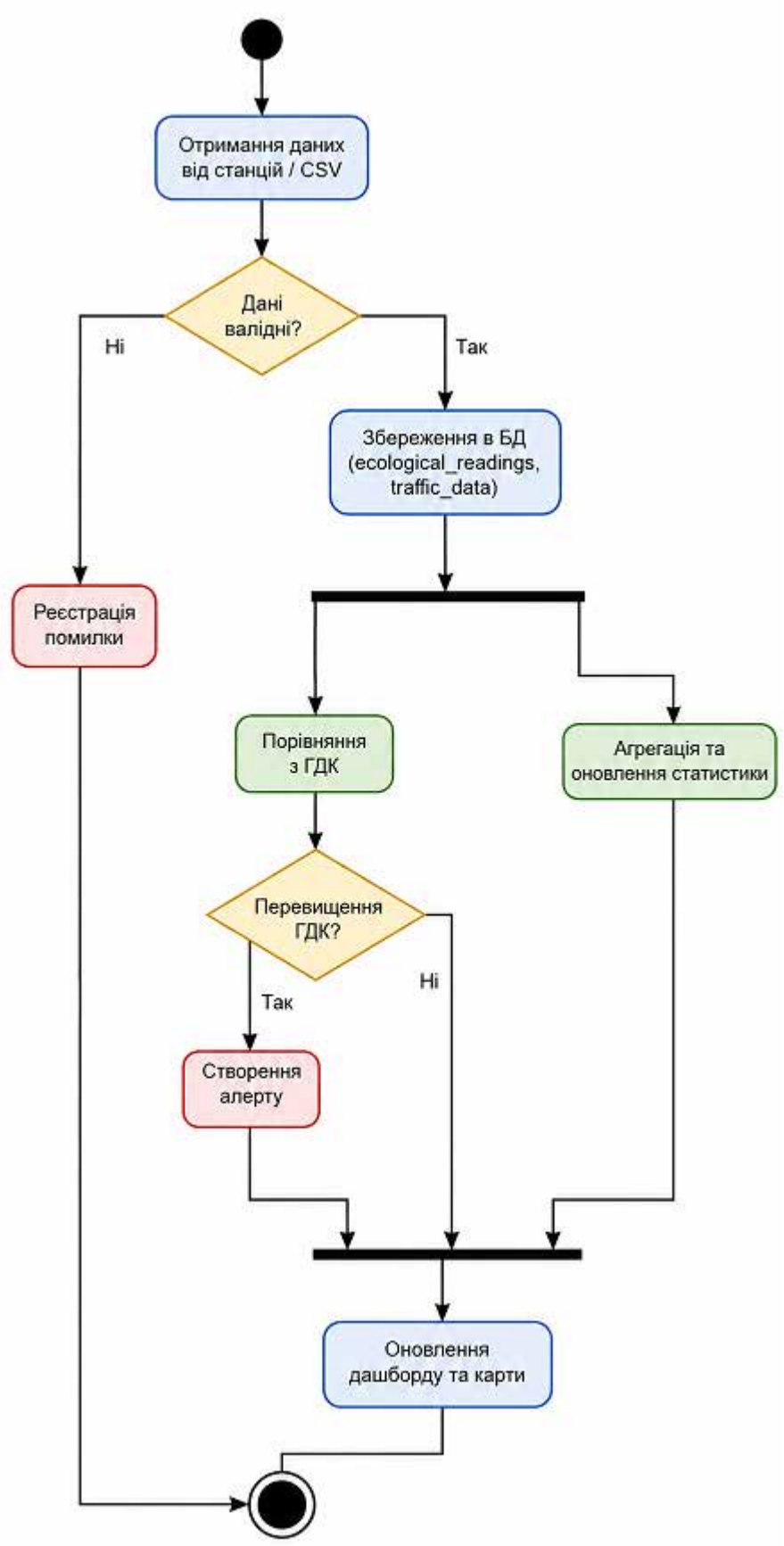

Рис. 2.4 Діаграма діяльності процесу обробки даних

Наведена діаграма відображає послідовність операцій під час обробки екологічних вимірювань. Після етапу валідації дані зберігаються у таблицях ecological_readings та traffic_data, після чого паралельно виконуються оновлення статистичних показників і перевірка перевищення гранично допустимих концентрацій. У випадку виявлення критичних значень модуль AlertEngine формує відповідне сповіщення, а результати обробки передаються до інформаційної панелі та картографічного інтерфейсу системи.

Діаграма послідовності (Sequence Diagram). Для деталізації взаємодії компонентів під час генерації сповіщень побудовано діаграму послідовності (рис. 2.5). Діаграма описує сценарій надходження нового вимірювання від станції моніторингу та процес перевірки на перевищення ГДК.

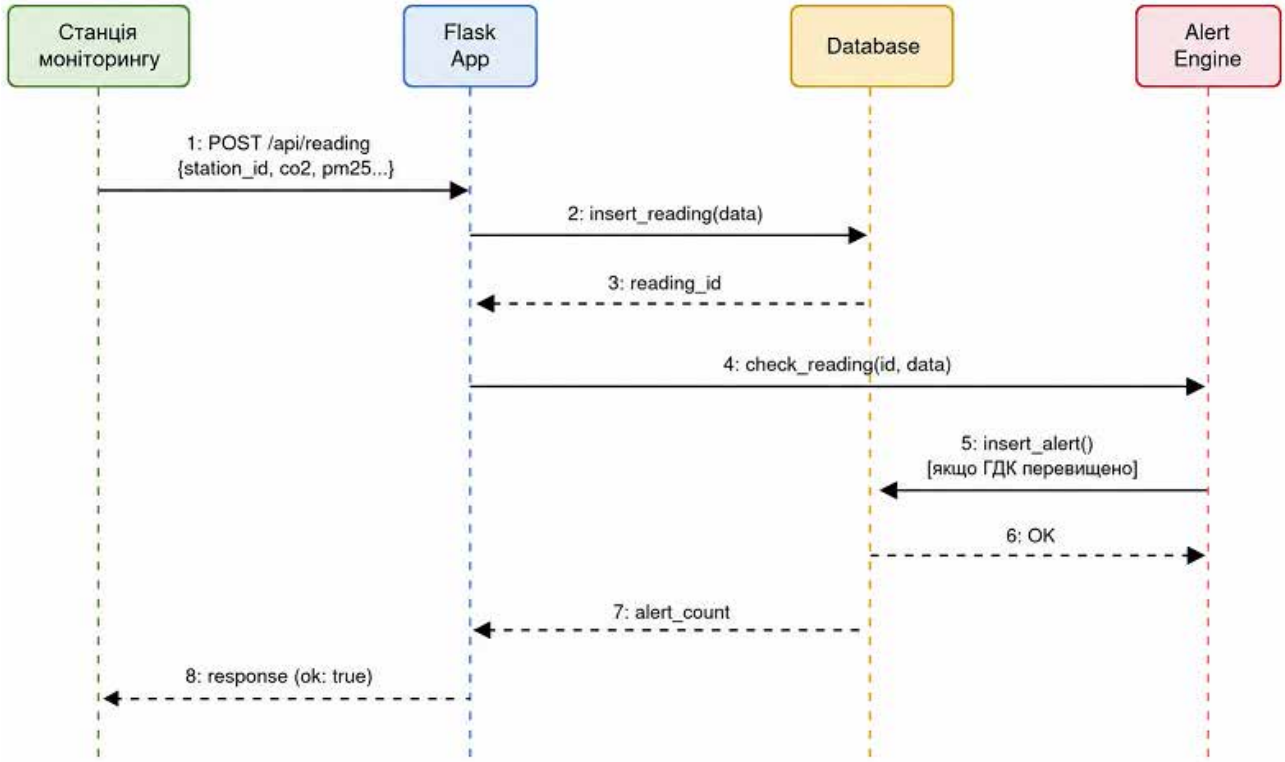

Рис. 2.5 Діаграма послідовності процесу генерації сповіщень

Послідовність взаємодії складається з восьми кроків. Станція надсилає POST-запит із даними вимірювання до Flask App. Сервер викликає метод insert_reading() класу Database, який зберігає дані та повертає ідентифікатор запису (reading_id). Далі Flask App передає ідентифікатор та дані до AlertEngine, який виконує метод check_reading(). Цей метод ітерує по параметрах (PM2.5,

PM10, NO_2, CO, CO_2), порівнює кожне значення з трьома порогами (warning, danger, critical) і за потреби викликає insert_alert() для збереження сповіщення. Результат повертається клієнту.

Діаграма класів (Class Diagram). Статичну структуру серверної частини системи описує діаграма класів (рис. 2.6). Центральним елементом є клас Database, який інкапсулює усю логіку взаємодії з СУБД SQLite: ініціалізацію схеми, CRUD-операції для кожної сутності та аналітичні запити. Чотири модулі бізнес-логіки (DataProcessor, AlertEngine, Predictor, ReportGenerator) залежать від Database та використовують його для отримання і збереження даних.

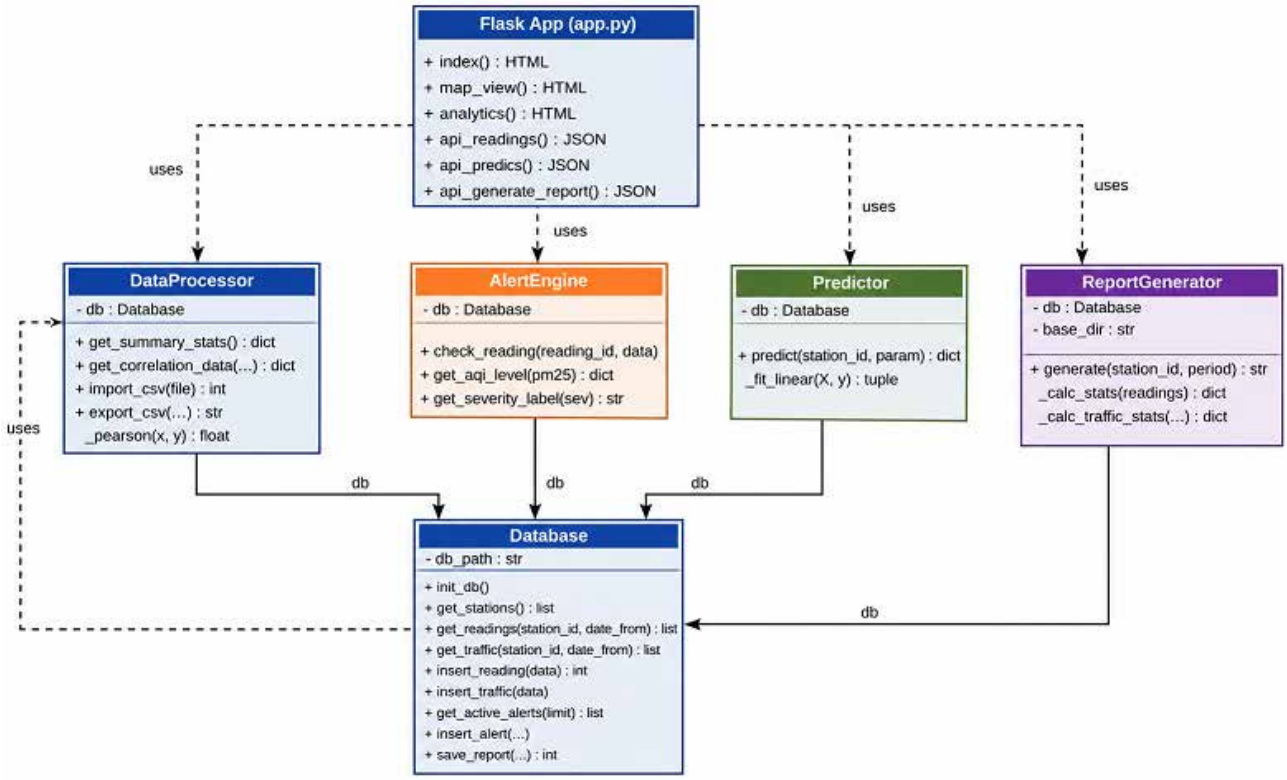

Рис. 2.6 Діаграма класів серверної частини системи

Клас DataProcessor реалізує методи агрегації зведеної статистики, обчислення коефіцієнта кореляції Пірсона між параметрами трафіку та забруднення, а також імпорт і експорт даних у форматі CSV. Клас AlertEngine відповідає за порівняння вимірювань із нормативними порогами та розрахунок індексу якості повітря (AQI). Клас Predictor реалізує множинну лінійну регресію методом найменших квадратів (OLS) без зовнішніх залежностей. Клас

ReportGenerator формує PDF-звіти з таблицями статистики та переліком алертів. Flask App виступає контролером, що координує роботу усіх модулів, обробляє HTTP-запити та формує відповіді у форматі HTML або JSON.

Одним із ключових елементів аналітичного модуля системи є алгоритм розрахунку індексу якості повітря AQI, який використовується для кольорової індикації рівня забруднення у веб-інтерфейсі та на інтерактивній карті. Для кольорової індикації стану повітря на дашборді та карті використовується алгоритм розрахунку AQI на основі концентрації PM2.5. Блок-схему алгоритму представлено на рисунку 2.7.

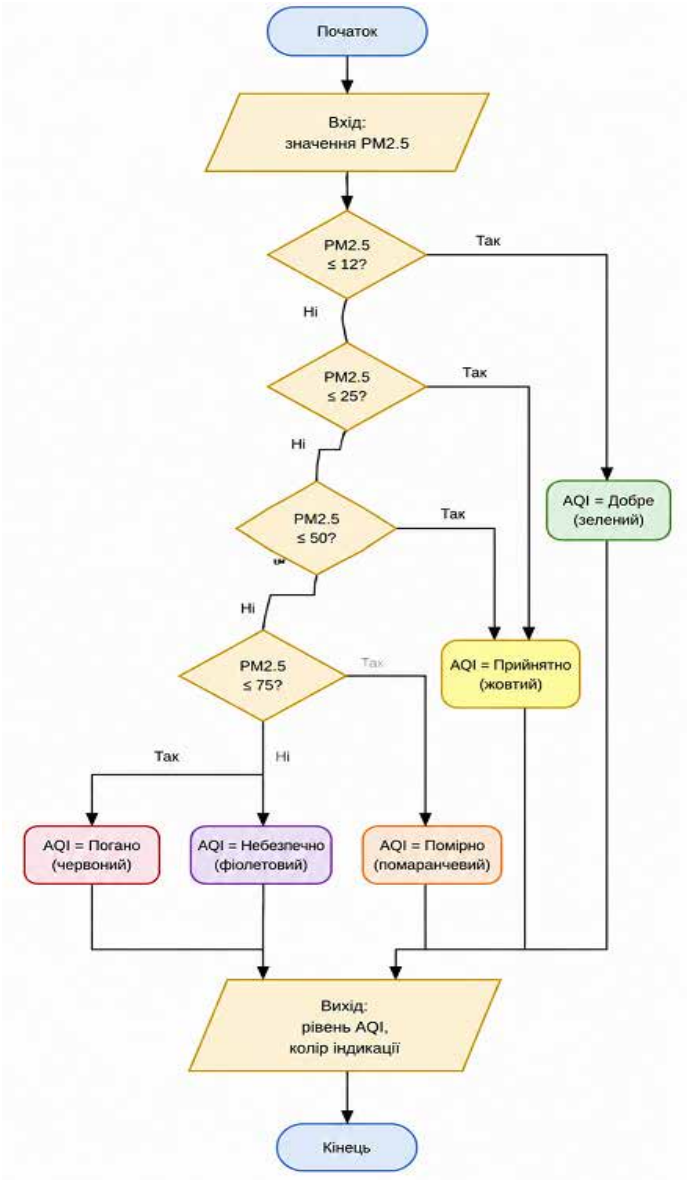

Рис. 2.7 Блок-схема алгоритму розрахунку індексу якості повітря

Алгоритм приймає на вхід значення PM2.5 (мкг/м³) та послідовно порівнює його з пороговими значеннями: 12 (добре), 25 (прийнятно), 50 (помірно), 75 (погано). Якщо жоден з порогів не задовольняється, присвоюється рівень «Небезпечно». Кожному рівню відповідає колір індикації, що використовується у маркерах карти та KPI-картках дашборду.

Аналіз потоків даних у системі дозволив визначити основні напрями передавання інформації між програмними компонентами та зовнішніми джерелами. Основні характеристики потоків даних наведено у таблиці 2.1. Потоки даних у системі EcoMonitor можна розподілити на три категорії за напрямком та періодичністю: вхідні потоки (дані від станцій моніторингу та CSV-імпорт), внутрішні потоки (обмін даними між модулями серверної частини) та вихідні потоки (візуалізації, сповіщення, звіти, CSV-експорт).

Таблиця 2.1

Характеристика потоків даних у системі EcoMonitor

Тип потоку	Джерело	Призначення	Формат
Вхідний потік	Станції моніторингу	Передавання екологічних та транспортних показників	JSON
Вхідний потік	CSV-файли	Імпорт історичних даних	CSV
Внутрішній потік	Серверні модулі	Обмін аналітичними результатами	Python-об’єкти
Вихідний потік	Веб-інтерфейс	Відображення графіків та карти	HTML / JSON
Вихідний потік	Модуль звітності	Формування звітів	PDF
Вихідний потік	Експорт даних	Вивантаження результатів аналізу	CSV

Вхідні потоки формуються двома способами. Основний спосіб – періодичне надходження вимірювань від станцій моніторингу. У демонстраційній версії системи дані генеруються методом _seed() класу Database із моделюванням реалістичних патернів (добові цикли, різниця між типами зон, вплив вихідних днів). У промисловій версії передбачається підключення реальних сенсорів через REST API або протокол MQTT із періодичністю

опитування від 1 до 15 хвилин. Другий спосіб – ручний імпорт історичних даних із CSV-файлів через модуль DataProcessor.

Внутрішні потоки визначаються архітектурою модулів. Flask App отримує HTTP-запит від клієнта та делегує обробку відповідному модулю. Наприклад, запит до сторінки дашборду ініціює виклик DataProcessor.get_summary_stats(), який виконує агрегуючий SQL-запит до бази даних та повертає словник із одинадцятьма метриками. Запит до API кореляційного аналізу ініціює DataProcessor.get_correlation_data(), який виконує JOIN-запит між таблицями ecological_readings та traffic_data, обчислює коефіцієнт Пірсона та повертає результат у форматі JSON.

Особливістю внутрішнього потоку генерації сповіщень є його асинхронний характер. При кожному новому вимірюванні модуль AlertEngine перевіряє шість параметрів (PM2.5, PM10, NO_2, CO, CO_2, шум) проти трьох порогів (warning, danger, critical), що потенційно може породити до 18 сповіщень від одного вимірювання. На практиці кількість сповіщень значно менша, оскільки не всі параметри одночасно перевищують пороги.

Вихідні потоки включають HTML-сторінки (рендеринг шаблонів Jinja2), JSON-відповіді API (для побудови графіків Chart.js на клієнті), PDF-файли звітів та CSV-файли експорту. Обсяг вихідного JSON-потоку для графіків аналітики за тиждень становить приблизно 50–80 КБ (700–900 записів), за місяць – 200–350 КБ, що забезпечує прийнятний час завантаження навіть при повільному з'єднанні.

Реалізована модель потоків даних забезпечує цілісність інформації, узгодженість взаємодії між модулями системи та підтримку аналітичної обробки екологічних і транспортних показників.

2.3 Проєктування бази даних системи

Проєктування бази даних виконано відповідно до принципів реляційної моделі з нормалізацією до третьої нормальної форми (3НФ). Логічну модель

даних представлено у вигляді ER-діаграми (Entity-Relationship), що визначає сутності, їхні атрибути та зв'язки між ними [19].

Логічну структуру бази даних побудовано із використанням шести взаємопов’язаних таблиць, які описують станції моніторингу, результати екологічних вимірювань, транспортні параметри, сповіщення, звіти та користувачів системи. Центральне місце у структурі займають таблиці ecological_readings та traffic_data, які забезпечують зберігання часових рядів екологічних і транспортних показників.

Для забезпечення структурованого зберігання екологічних, транспортних та службових даних у системі EcoMonitor сформовано реляційну модель бази даних, яка охоплює основні сутності предметної області та взаємозв’язки між ними. Структура бази даних орієнтована на підтримку процесів моніторингу, аналітичної обробки, формування сповіщень і генерації звітів. Основні таблиці бази даних системи та їх призначення наведено у табл. 2.2.

Таблиця 2.2

Характеристика таблиць бази даних системи EcoMonitor

Назва таблиці	Призначення	Основні атрибути
monitoring_stations	Зберігання інформації про станції моніторингу	id, name, latitude, longitude, zone_type
ecological_readings	Зберігання результатів екологічних вимірювань	station_id, timestamp, co2, co, no2, pm25, pm10, noise_level, temperature, humidity
traffic_data	Зберігання параметрів транспортного потоку	station_id, timestamp, vehicle_count, avg_speed, congestion_level
alerts	Зберігання сповіщень про перевищення гранично допустимих концентрацій	reading_id, parameter, threshold_value, actual_value, severity
reports	Зберігання метаданих сформованих PDF-звітів	station_id, period, filepath, created_at
users	Зберігання облікових записів користувачів та ролей доступу	login, password_hash, role

Таблиця monitoring_stations містить реєстр станцій моніторингу із зазначенням географічних координат та типу зони розташування. Центральною таблицею системи є ecological_readings, яка забезпечує зберігання екологічних показників та пов’язана зі станціями моніторингу через зовнішній ключ station_id. Таблиця traffic_data використовується для накопичення характеристик транспортного потоку та підтримує виконання кореляційного аналізу спільно з таблицею ecological_readings. Таблиця alerts містить інформацію про перевищення нормативних значень екологічних параметрів, а reports використовується для зберігання інформації про сформовані PDF-звіти. Таблиця users забезпечує підтримку механізму авторизації та розподілу прав доступу відповідно до ролей користувачів системи.

На рисунку 2.8 представлено ER-діаграму бази даних із зазначенням типів зв'язків між сутностями.

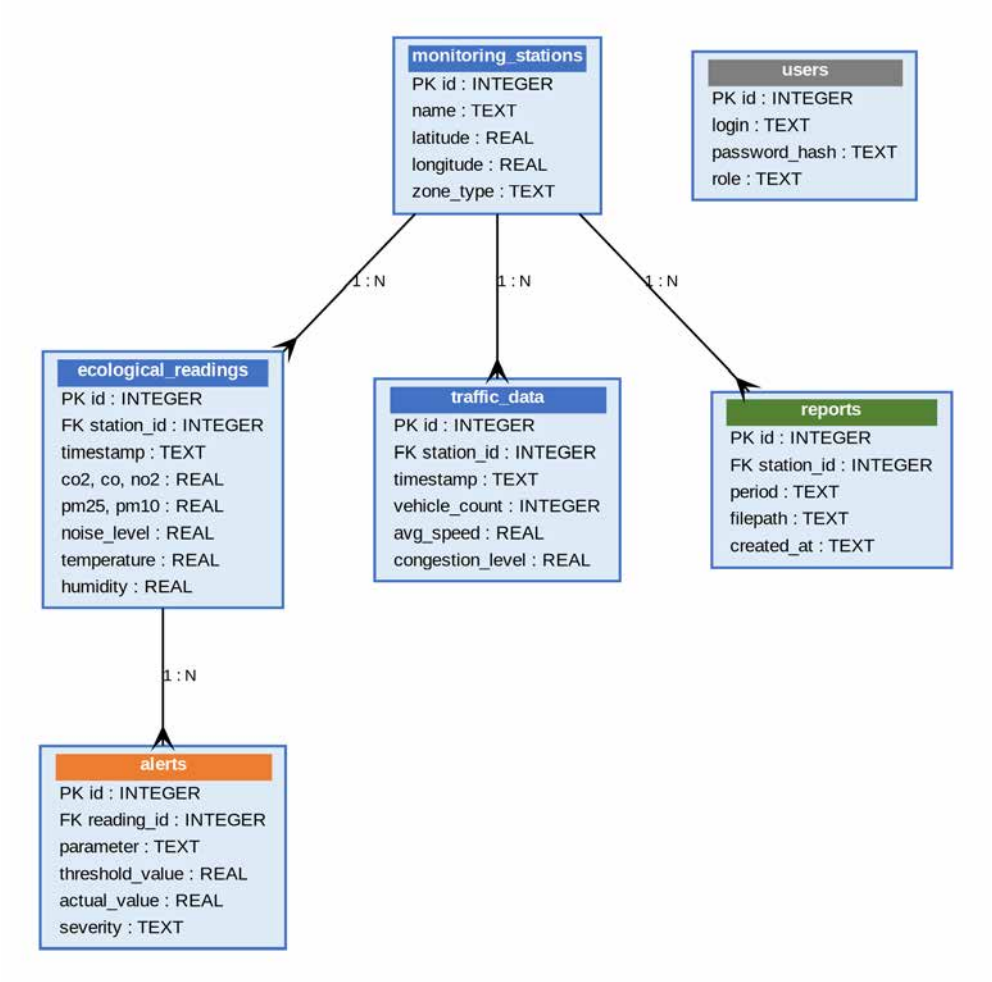

Рис. 2.8 ER-діаграма бази даних системи EcoMonitor

Наведена ER-діаграма демонструє структуру взаємозв’язків між таблицями системи. Таблиця monitoring_stations містить інформацію про станції моніторингу та пов’язана із таблицями ecological_readings, traffic_data і reports за принципом «один до багатьох». Таблиця alerts зв’язана з ecological_readings через зовнішній ключ reading_id та використовується для зберігання інформації про перевищення нормативних значень екологічних параметрів. Таблиця users забезпечує підтримку механізму авторизації та зберігає логіни, хеші паролів і ролі користувачів системи.

Структура зв’язків забезпечує підтримку процесів моніторингу, аналітичної обробки та формування звітності. Для зберігання екологічних показників використовується таблиця ecological_readings, структура якої наведена у табл. 2.3. Таблиця містить часову мітку вимірювання, концентрації забруднювачів та метеорологічні параметри, що використовуються у процесах аналізу і прогнозування.

Таблиця 2.3

Структура таблиці ecological_readings

Поле	Тип	Ключ	Опис
id	INTEGER	PK	Унікальний ідентифікатор запису
station_id	INTEGER	FK	Зовнішній ключ на monitoring_stations
timestamp	TEXT	IDX	Часова мітка вимірювання (ISO 8601)
co2	REAL	–	Концентрація CO_2, ppm
co	REAL	–	Концентрація CO, мг/м³
no2	REAL	–	Концентрація NO_2, мкг/м³
pm25	REAL	–	Концентрація PM2.5, мкг/м³
pm10	REAL	–	Концентрація PM10, мкг/м³
noise_level	REAL	–	Рівень шуму, дБ
temperature	REAL	–	Температура повітря, °C
humidity	REAL	–	Відносна вологість, %

Структура таблиці забезпечує зберігання повного набору параметрів, необхідних для екологічного моніторингу та подальшої аналітичної обробки даних. Використання часових міток у форматі ISO 8601 дозволяє спростити

виконання SQL-запитів часових рядів та підвищує читабельність інформації під час налагодження системи.

Для підвищення продуктивності роботи застосунку використано систему індексування, орієнтовану на оптимізацію найпоширеніших типів запитів. Основні індекси бази даних наведено у таблиці 2.4.

Таблиця 2.4

Індекси бази даних

Назва індексу	Поля	Призначення
idx_readings_station_ts	station_id, timestamp	Прискорення запитів часових рядів
idx_traffic_station_ts	station_id, timestamp	Прискорення JOIN з ecological_readings
idx_alerts_reading	reading_id	Швидкий пошук алертів за вимірюванням

Використання складених індексів по полях (station_id, timestamp) дозволяє ефективно виконувати запити фільтрації за станцією та часовим діапазоном, які є найбільш поширеними у модулях аналітики, кореляційного аналізу та прогнозування. Індекс idx_alerts_reading забезпечує швидкий пошук сповіщень, пов'язаних із конкретним вимірюванням, що необхідно для формування звітів.

Фізична модель бази даних реалізована за допомогою DDL-скриптів мовою SQL, які автоматично виконуються при ініціалізації системи методом init_db() класу Database. Скрипт включає оператори CREATE TABLE IF NOT EXISTS для кожної з шести таблиць, оператори CREATE INDEX для створення індексів та заповнення демонстраційними даними (8 станцій моніторингу з 60-денною історією вимірювань). Обраний підхід до проєктування бази даних забезпечує нормалізовану структуру зберігання, ефективні запити за рахунок індексів та референційну цілісність через зовнішні ключі з увімкненою директивою PRAGMA foreign_keys=ON. Вибір типів даних для полів таблиць бази даних виконано з урахуванням особливостей SQLite. На відміну від інших СУБД, SQLite використовує систему динамічної типізації, де тип значення визначається самим значенням, а не стовпцем таблиці. Проте оголошення типів у DDL-операторах є рекомендованою практикою для документування схеми та сумісності з інструментами візуалізації баз даних [17].

Для зберігання часових міток обрано тип TEXT у форматі ISO 8601 (YYYY-MM-DD HH:MM:SS). Альтернативним підходом є зберігання Unix timestamp як INTEGER, що забезпечує компактніше зберігання та швидше порівняння. Проте текстовий формат обрано з міркувань зручності налагодження та читабельності SQL-запитів: оператор WHERE timestamp >= '2024-01-01' є інтуїтивно зрозумілим, тоді як WHERE timestamp >= 1704067200 потребує додаткового перетворення.

Стратегію індексування визначено на основі аналізу типових запитів системи. Найбільш поширеним патерном є фільтрація за станцією та часовим діапазоном, що використовується модулями аналітики, кореляційного аналізу та прогнозування. Для цього патерну створено складені індекси idx_readings_station_ts та idx_traffic_station_ts, що покривають обидва поля (station_id, timestamp). Порядок полів в індексі відповідає порядку їх використання у WHERE-частині запитів, що забезпечує оптимальне використання B-дерева індексу [19].

Обсяг бази даних у демонстраційній конфігурації (8 станцій, 60 днів, вимірювання кожні 2 години) становить приблизно 15 000 записів у кожній із таблиць ecological_readings та traffic_data, загальний розмір файлу ecomonitor.db – близько 5 МБ. При масштабуванні до 50 станцій із щохвилинним опитуванням обсяг зростатиме приблизно на 70 МБ на місяць, що залишається в межах ефективної роботи SQLite (рекомендований максимум – 1 ГБ для однофайлової бази). При перевищенні цього обсягу рекомендовано міграцію на PostgreSQL або архівування старих даних.

2.4 Висновки до розділу 2

У другому розділі виконано проєктування системи моніторингу екологічних параметрів транспортних потоків та її інформаційного забезпечення.

Розроблено трирівневу архітектуру системи EcoMonitor, що включає клієнтський рівень (HTML/CSS/JS, Chart.js, Leaflet.js), серверний рівень (Python, Flask 3.0, п'ять модулів бізнес-логіки) та рівень даних (SQLite 3). Побудовано архітектурну діаграму, діаграму розгортання та діаграму варіантів використання з трьома акторами (оглядач, аналітик, адміністратор) і десятьма варіантами використання.

Виконано моделювання процесів обробки даних: побудовано діаграму діяльності основного процесу (від отримання вимірювань до оновлення інтерфейсу), діаграму послідовності генерації сповіщень та діаграму класів серверної частини. Розроблено блок-схему алгоритму розрахунку індексу якості повітря (AQI) на основі концентрації PM2.5.

Спроєктовано реляційну базу даних із шести таблиць, нормалізованих до 3НФ: monitoring_stations, ecological_readings, traffic_data, alerts, reports, users. Визначено типи зв'язків між сутностями (1:N), побудовано ER-діаграму та створено три складених індекси для оптимізації найбільш поширених запитів.

3 РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Вибір технологій і засобів розробки

Розроблення веб-орієнтованої системи моніторингу екологічних параметрів транспортних потоків потребувало вибору технологічного стеку, здатного забезпечити підтримку серверної обробки даних, інтерактивної візуалізації, аналітичних обчислень та роботи із реляційною базою даних. Під час вибору технологій основна увага приділялась кросплатформності, відкритості програмного забезпечення, наявності засобів для реалізації REST API, підтримці статистичного аналізу та можливості розгортання системи без складної серверної інфраструктури [16].

Мова програмування Python 3. Серверну частину системи реалізовано мовою Python версії 3.10 та вище. Обрання Python обумовлено кількома факторами. По-перше, мова має потужну стандартну бібліотеку, що включає модуль sqlite3 для роботи з СУБД SQLite без встановлення додаткових залежностей. По-друге, Python є де-факто стандартом для задач обробки та аналізу даних, що безпосередньо відповідає аналітичному характеру розроблюваної системи. По-третє, наявність мікрофреймворку Flask суттєво спрощує створення REST API та рендеринг HTML-шаблонів [16].

Фреймворк Flask 3.0. Flask є легковаговим WSGI-фреймворком для побудови веб-додатків на Python. На відміну від повнофункціональних фреймворків (Django, FastAPI), Flask дотримується філософії мінімалізму: він надає лише базові компоненти – маршрутизацію URL, обробку HTTP-запитів, систему шаблонів Jinja2 та підтримку сесій – залишаючи розробнику свободу у виборі додаткових бібліотек. Така архітектура є оптимальною для системи EcoMonitor, оскільки бізнес-логіка інкапсульована у власних модулях, а Flask виконує виключно роль контролера [20].

СУБД SQLite 3. SQLite обрано як реляційну СУБД рівня даних. Основними перевагами SQLite для цього проєкту є: вбудованість у стандартну бібліотеку

Python (модуль sqlite3), сервepless-архітектура (вся база зберігається в одному файлі ecomonitor.db), підтримка транзакцій ACID, сумісність із SQL-стандартом та достатня продуктивність для обсягів даних, характерних для моніторингових систем з кількістю станцій до кількох десятків. Для забезпечення цілісності даних увімкнено директиву PRAGMA foreign_keys=ON, а для підвищення продуктивності конкурентного доступу – PRAGMA journal_mode=WAL [17].

Бібліотека Chart.js 4. Для інтерактивної візуалізації графіків на клієнтському рівні обрано бібліотеку Chart.js версії 4. Chart.js підтримує широкий спектр типів діаграм: лінійні, стовпчикові, scatter-plot, комбіновані – що повністю покриває потреби модулів аналітики, кореляційного аналізу та прогнозування. Бібліотека має адаптивний рендеринг (автоматичне масштабування під розмір контейнера), підтримку кількох осей Y та вбудовані анімації. Підключення здійснюється через CDN без необхідності локальної збірки [21].

Бібліотека Leaflet.js 1.9. Для картографічної візуалізації станцій моніторингу обрано Leaflet.js – найпоширенішу відкриту JavaScript-бібліотеку для інтерактивних карт. Leaflet забезпечує відображення тайлових карт (OpenStreetMap), розміщення маркерів із кастомними іконками, popup-вікна з інформацією та полігони (кола) для відображення зон забруднення. Бібліотека є компактною (42 KB gzip) та має широку сумісність із мобільними браузерами [22].

Шаблонізатор Jinja2. Серверний рендеринг HTML-сторінок виконується шаблонізатором Jinja2, що є стандартним компонентом Flask. Jinja2 підтримує успадкування шаблонів (template inheritance), що дозволяє визначити базовий макет сторінки (base.html) із бічною навігацією, верхньою панеллю та областю контенту, а кожна окрема сторінка (dashboard.html, map.html тощо) розширює базовий шаблон, перевизначаючи лише блок контенту. Такий підхід забезпечує єдність дизайну та спрощує підтримку коду.

Каскадні таблиці стилів CSS. Візуальне оформлення інтерфейсу реалізовано через єдиний файл style.css із використанням CSS Custom Properties

(змінних) для централізованого управління кольоровою палітрою, відступами та тінями. Адаптивність інтерфейсу забезпечується медіа-запитами (@media) для трьох контрольних точок: 1024 px, 768 px та нижче. Шрифтове оформлення базується на гарнітурі IBM Plex Sans, що забезпечує високу читабельність на екранах різних розмірів.

Основні технології, використані під час реалізації системи EcoMonitor, наведено у табл. 3.1.

Таблиця 3.1

Технологічний стек системи EcoMonitor

Компонент	Технологія	Призначення
Мова сервера	Python 3.10+	Реалізація бізнес-логіки, API, обробка даних
Веб-фреймворк	Flask 3.0	Маршрутизація, рендеринг шаблонів, REST API
Шаблонізатор	Jinja2	Серверний рендеринг HTML-сторінок
СУБД	SQLite 3	Зберігання даних у файлі ecomonitor.db
Графіки	Chart.js 4	Лінійні, стовпчикові, scatter-plot діаграми
Карта	Leaflet.js 1.9	Інтерактивна карта станцій (OpenStreetMap)
Стилі	CSS3	Адаптивний дизайн із CSS Custom Properties
Шрифт	IBM Plex Sans	Основний шрифт інтерфейсу (Google Fonts CDN)
Іконки	Font Awesome 6	Піктограми навігації та KPI-карток
PDF-звіти	SimplePDF / fpdf2	Генерація PDF із таблицями та статистикою

Наведений технологічний стек забезпечує підтримку серверної обробки даних, інтерактивної візуалізації та аналітичних функцій системи. Використані програмні засоби є відкритими та безкоштовними, що спрощує розгортання застосунку та подальшу підтримку програмного забезпечення.

Структуру програмного проєкту сформовано за модульним принципом із розділенням серверної логіки, HTML-шаблонів та статичних ресурсів. Такий підхід спрощує супровід програмного коду, ізолює функціональні компоненти та забезпечує зручність масштабування системи. Основні файли та каталоги проєкту наведено у таблиці 3.2.

Таблиця 3.2

Структура файлів проєкту

Файл / каталог	Призначення
app.py	Головний модуль Flask-додатку: маршрути сторінок та API
init_db.py	Скрипт ініціалізації БД з демонстраційними даними
requirements.txt	Перелік Python-залежностей (flask, fpdf2)
modules/database.py	Клас Database: DDL-схема, CRUD, індекси, seed-дані
modules/data_processor.py	Клас DataProcessor: агрегація, кореляція, CSV
modules/alert_engine.py	Клас AlertEngine: перевірка ГДК, розрахунок AQI
modules/predictor.py	Клас Predictor: лінійна регресія (OLS)
modules/report_generator.py	Клас ReportGenerator: генерація PDF-звітів
templates/base.html	Базовий шаблон: sidebar, topbar, блоки контенту
templates/dashboard.html	Дашборд: KPI-картки, таблиця станцій, графіки
templates/map.html	Інтерактивна карта Leaflet із маркерами станцій
templates/analytics.html	Аналітика: динаміка параметрів, порівняння станцій
templates/correlation.html	Кореляційний аналіз: scatter-plot, Пірсон
templates/predictions.html	Прогнозування: графік історії та прогнозу
templates/alerts.html	Журнал сповіщень із фільтрацією за рівнем
templates/reports.html	Генерація PDF, імпорт/експорт CSV
static/css/style.css	Стилі інтерфейсу: KPI, таблиці, адаптивність

Структура проєкту забезпечує розділення відповідальності між компонентами системи та спрощує підтримку програмного забезпечення. Головний модуль app.py координує взаємодію між серверними компонентами та обробляє HTTP-запити, каталог modules містить функціональні модулі бізнес-логіки, а каталог templates використовується для серверного рендерингу веб-сторінок. Така організація коду забезпечує підтримку модульної архітектури системи та підвищує зручність подальшої модифікації програмного забезпечення.

3.2 Реалізація модулів збору, обробки та збереження даних

Серверна частина системи EcoMonitor складається з п'яти модулів бізнес-логіки, кожен із яких відповідає за окрему функціональну область. У цьому підрозділі розглянуто реалізацію ключових модулів: AlertEngine (система сповіщень), DataProcessor (кореляційний аналіз) та Predictor (прогнозування).

Модуль alert_engine.py реалізує механізм автоматичної генерації сповіщень при перевищенні гранично допустимих концентрацій (ГДК). Центральним елементом модуля є словник THRESHOLDS, що містить порогові значення для кожного екологічного параметра. Для кожного параметра визначено три рівні загрози: warning (увага), danger (небезпека) та critical (критично). Значення порогів відповідають рекомендаціям ВООЗ та нормативам України.

На рисунку 3.1 представлено фрагмент коду модуля AlertEngine зі словником порогових значень. Для параметра PM2.5 встановлено пороги 25, 50 та 75 мкг/м³ відповідно для рівнів warning, danger та critical. Для PM10 – 50, 100 та 150 мкг/м³. Для NO_2 – 40, 80 та 120 мкг/м³.

```
Модуль системи сповіщень.
Порівняння показників з ГДК та генерація алертів.
"""


# Гранично допустимі концентрації (ГДК) згідно з нормативами
THRESHOLDS = {
    "pm25": {
        "unit": "мкг/м³",
        "warning": 25,
        "danger": 50,
        "critical": 75,
        "name": "PM2.5",
    },
    "pm10": {
        "unit": "мкг/м³",
        "warning": 50,
        "danger": 100,
        "critical": 150,
        "name": "PM10",
    },
    "no2": {
        "unit": "мкг/м³",
        "warning": 40,
        "danger": 80,
        "critical": 120,
        "name": "NO₂",
```

Рис. 3.1 Фрагмент коду модуля AlertEngine: словник порогових значень ГДК

Метод check_reading() класу AlertEngine ітерує по всіх параметрах словника THRESHOLDS для кожного нового вимірювання. Якщо значення параметра перевищує один із порогів, метод визначає рівень загрози (warning, danger або critical) та викликає метод insert_alert() класу Database для збереження сповіщення. Таким чином, одне вимірювання може породити кілька сповіщень – по одному для кожного параметра, що перевищив норму.

Додатково модуль містить статичний метод get_aqi_level(), що обчислює індекс якості повітря (AQI) на основі концентрації PM2.5. Метод повертає словник із трьома полями: level (текстовий опис рівня), color (колір для візуалізації) та index (числовий індекс від 1 до 6). Цей метод використовується шаблонами дашборду та карти для кольорової індикації стану повітря.

Модуль data_processor.py відповідає за агрегацію зведеної статистики, обчислення кореляційних залежностей між параметрами трафіку та забруднення, а також за імпорт і експорт даних у форматі CSV.

Ключовим методом модуля є get_correlation_data(), який виконує JOIN-запит між таблицями ecological_readings та traffic_data для отримання пар значень «забруднення – трафік» за обраний період. На рисунку 3.2 представлено фрагмент SQL-запиту та підготовки даних для кореляційного аналізу.

```
def get_correlation_data(self, station_id, param, days=30) -> dict:
    """Дані для кореляційного аналізу: трафік vs забруднення."""
    import sqlite3
    conn = sqlite3.connect(self.db.db_path)
    conn.row_factory = sqlite3.Row

    date_from = (datetime.now() - timedelta(days=days)).isoformat()

    sql = f"""
        SELECT e.{param} AS eco_val, t.vehicle_count, t.avg_speed,
               t.congestion_level, e.timestamp
        FROM ecological_readings e
        JOIN traffic_data t
            ON t.station_id = e.station_id AND t.timestamp = e.timestamp
        WHERE e.station_id = ? AND e.timestamp >= ?
        ORDER BY e.timestamp
    """
    rows = conn.execute(sql, (station_id, date_from)).fetchall()
    conn.close()

    eco_vals = [r["eco_val"] for r in rows if r["eco_val"] is not None]
    traffic_vals = [r["vehicle_count"] for r in rows if r["eco_val"] is not None]
    speed_vals = [r["avg_speed"] for r in rows if r["eco_val"] is not None]
    timestamps = [r["timestamp"] for r in rows if r["eco_val"] is not None]
```

Рис. 3.2 Фрагмент коду методу get_correlation_data() класу DataProcessor

SQL-запит виконує з'єднання таблиць ecological_readings та traffic_data за ключами station_id та timestamp, що дозволяє отримати синхронізовані пари значень для кореляційного аналізу. Після отримання результатів запиту дані розподіляються по окремих списках: eco_vals (значення обраного екологічного параметра), traffic_vals (кількість транспортних засобів), speed_vals (середня швидкість) та timestamps (часові мітки).

Для обчислення коефіцієнта кореляції Пірсона реалізовано приватний статичний метод _pearson(x, y), що обчислює коефіцієнт за класичною формулою без використання зовнішніх бібліотек. Коефіцієнт Пірсона r приймає значення від –1 до +1, де значення близькі до +1 вказують на сильний прямий зв'язок, значення близькі до –1 – на сильний зворотний зв'язок, а значення близькі до 0 – на відсутність лінійного зв'язку.

Метод get_summary_stats() обчислює зведену статистику для дашборду: середні значення PM2.5, CO_2, NO_2, рівня шуму, кількості транспортних засобів, середньої швидкості та кількості алертів за поточну добу. Ці дані відображаються у KPI-картках на головній сторінці.

Методи import_csv() та export_csv() забезпечують обмін даними із зовнішніми системами. Метод import_csv() приймає об'єкт файлу, парсить його як CSV за допомогою модуля csv стандартної бібліотеки та послідовно вставляє записи у таблицю ecological_readings. Метод export_csv() виконує зворотну операцію: формує CSV-файл із результатами запиту за обраний період та станцію.

Модуль predictor.py реалізує прогнозування рівня забруднення на основі даних транспортного потоку методом множинної лінійної регресії. Модуль не використовує зовнішніх бібліотек для машинного навчання – алгоритм OLS (Ordinary Least Squares) реалізовано із нуля.

Метод predict() класу Predictor виконує повний цикл прогнозування: отримання навчальних даних, побудову моделі та формування прогнозу. На рисунку 3.3 представлено фрагмент коду, що виконує отримання даних та підготовку ознак для регресійної моделі.

```python
class Predictor:
    def __init__(self, db):
        self.db = db

    def predict(self, station_id: int, param: str = "pm25") -> dict:
        """
        Побудувати модель лінійної регресії та спрогнозувати
        значення параметра на наступні 24 години.
        """
        conn = sqlite3.connect(self.db.db_path)
        conn.row_factory = sqlite3.Row

        date_from = (datetime.now() - timedelta(days=30)).isoformat()

        sql = f"""
            SELECT e.{param} AS eco_val, t.vehicle_count, t.avg_speed,
                   t.congestion_level, e.timestamp
            FROM ecological_readings e
            JOIN traffic_data t
                ON t.station_id = e.station_id AND t.timestamp = e.timestamp
            WHERE e.station_id = ? AND e.timestamp >= ?
            ORDER BY e.timestamp
        """
        rows = conn.execute(sql, (station_id, date_from)).fetchall()
        conn.close()
```

Рис. 3.3 Фрагмент коду класу Predictor: отримання даних та підготовка ознак

Наведений фрагмент коду демонструє процес формування навчальної вибірки для побудови регресійної моделі прогнозування. На цьому етапі виконується вибірка історичних транспортних та екологічних даних із бази SQLite, їх синхронізація за часовими мітками та формування матриці ознак для подальшого навчання моделі. Використання агрегованих транспортних показників дозволяє враховувати вплив інтенсивності руху та рівня заторності на зміну концентрації забруднювачів у міському середовищі.

Модель приймає на вхід три ознаки: кількість транспортних засобів (vehicle_count), середню швидкість руху (avg_speed) та рівень заторності (congestion_level). Цільовою змінною є значення обраного екологічного параметра (за замовчуванням PM2.5). Навчальна вибірка формується за останні 30 днів.

Для побудови прогнозу на 24 години вперед використовується модель добового циклу транспортного потоку. На рисунку 3.4 показано фрагмент коду, що моделює зміну інтенсивності трафіку залежно від години доби та обчислює прогнозоване значення забруднення.

```
predictions = []
now = datetime.now()
for i in range(1, 13):
    future_time = now + timedelta(hours=i * 2)
    hour = future_time.hour

    # Моделюємо добовий цикл трафіку
    if 7 <= hour <= 9 or 17 <= hour <= 19:
        rush_factor = 1.5
    elif 10 <= hour <= 16:
        rush_factor = 1.1
    elif 0 <= hour <= 5:
        rush_factor = 0.35
    else:
        rush_factor = 0.7

    pred_traffic = [
        last_traffic[0] * rush_factor,
        last_traffic[1] / rush_factor,
        last_traffic[2] * rush_factor,
    ]

    pred_val = intercept
    for c, x in zip(coeffs, pred_traffic):
        pred_val += c * x
```

Рис. 3.4 Фрагмент коду Predictor: моделювання добового циклу та обчислення прогнозу

Добовий цикл моделюється через коефіцієнт rush_factor: під час ранкових (7:00–9:00) та вечірніх (17:00–19:00) годин пік коефіцієнт становить 1.5, у денний час (10:00–16:00) – 1.1, вночі (0:00–5:00) – 0.35, в інший час – 0.7. Прогнозовані характеристики трафіку обчислюються множенням останніх фактичних значень на rush_factor, після чого застосовується побудована регресійна модель для отримання прогнозу забруднення.

Для реалізації прогнозування використано метод множинної лінійної регресії, який дозволяє встановити залежність між транспортними показниками та рівнем екологічного забруднення. Побудова моделі виконується засобами власної реалізації алгоритму найменших квадратів без використання сторонніх бібліотек машинного навчання, що забезпечує прозорість обчислень та можливість адаптації алгоритму до специфіки системи моніторингу.

Приватний статичний метод _fit_linear() реалізує множинну лінійну регресію методом найменших квадратів (табл. 3.3). Алгоритм розширює матрицю ознак стовпцем одиниць (для вільного члена), формує нормальне

рівняння $X^TX\beta = X^Ty$ та розв'язує його методом Гауса з частковим вибором головного елемента. Метод повертає вектор коефіцієнтів, вільний член та коефіцієнт детермінації R^2, який характеризує частку дисперсії цільової змінної, пояснену моделлю.

Таблиця 3.3

Методи класу Predictor

Метод	Параметри	Результат
predict()	station_id, param	dict: predictions, history, model_info
_fit_linear()	X (матриця), y (вектор)	tuple: coeffs, intercept, R^2

Наведені методи забезпечують повний цикл прогнозування: від підготовки навчальної вибірки та побудови математичної моделі до формування прогнозних значень екологічних параметрів і передачі результатів до веб-інтерфейсу системи. Отримані прогнозні дані використовуються для побудови графіків, формування статистичних звітів та оцінювання можливих змін екологічної ситуації.

Модуль database.py є фундаментом серверної частини системи. Клас Database інкапсулює всю логіку взаємодії з SQLite: ініціалізацію схеми (DDL-оператори CREATE TABLE, CREATE INDEX), CRUD-операції для кожної сутності та аналітичні запити. З'єднання з базою даних управляється через контекстний менеджер _conn(), що забезпечує автоматичне закриття з'єднання та фіксацію транзакції.

Метод init_db() виконує ініціалізацію бази даних: створює шість таблиць та три індекси, а у разі порожньої бази – запускає метод _seed(), що генерує демонстраційні дані. Демо-набір включає 8 станцій моніторингу Києва з 60-денною історією вимірювань (кожні 2 години), що становить близько 15 000 записів у кожній із таблиць ecological_readings та traffic_data. Дані моделюють реалістичні патерни: добові цикли з піками у ранкові та вечірні години, зниження трафіку у вихідні дні, просторову диференціацію за типами зон (магістраль, перехрестя, житловий район).

Модуль генерації звітів (ReportGenerator). Модуль report_generator.py формує PDF-звіти зі статистикою за обраний період. Звіт включає три секції: зведена таблиця екологічних параметрів (середнє, мінімум, максимум, ГДК, статус), статистика транспортного потоку та перелік сповіщень. Модуль підтримує два механізми генерації PDF: основний – через бібліотеку fpdf2, та резервний – через вбудований клас SimplePDF, що не потребує зовнішніх залежностей.

3.3 Реалізація інтерфейсу моніторингу та візуалізації показників

Веб-інтерфейс системи EcoMonitor побудовано за принципом успадкування шаблонів Jinja2. Базовий шаблон base.html визначає загальну структуру сторінки: бічну навігаційну панель (sidebar) із сімома пунктами меню, верхню панель (topbar) із заголовком сторінки та поточним часом, а також область контенту, яку перевизначає кожна дочірня сторінка. Бічна навігація реалізує адаптивну поведінку: на екранах вужче 768 px вона приховується та відкривається кнопкою «гамбургер» (Додаток Б).

Для реалізації аналітичного модуля розроблено окрему сторінку analytics.html, яка забезпечує інтерактивний аналіз екологічних параметрів та візуалізацію результатів моніторингу у режимі реального часу. Основним призначенням сторінки є надання користувачу засобів для дослідження динаміки змін концентрацій забруднювачів, порівняння показників для різних станцій моніторингу та аналізу екологічної ситуації за визначений часовий інтервал. Інтерфейс сторінки побудовано таким чином, щоб забезпечити швидку зміну параметрів аналізу без перезавантаження веб-сторінки, що підвищує зручність роботи із системою та покращує оперативність отримання результатів.

Формування сторінки здійснюється із використанням HTML, CSS та шаблонізатора Jinja2, який дозволяє динамічно генерувати елементи інтерфейсу на основі даних, отриманих із серверної частини застосунку. Для побудови інтерактивних графіків використовується бібліотека Chart.js, що забезпечує

візуалізацію часових рядів екологічних показників та підтримує оновлення даних через HTTP-запити у форматі JSON. Фрагмент шаблону сторінки аналітики з елементами керування наведено на рис. 3.5.

```
% block content %}
div class="controls-bar">
  <div class="control-group">
      <label>Станція</label>
      <select id="stationSelect">
          <option value="">Усі станції</option>
          {% for s in stations %}
          <option value="{{ s.id }}">{{ s.name }}</option>
          {% endfor %}
      </select>
  </div>
  <div class="control-group">
      <label>Період</label>
      <select id="periodSelect">
          <option value="1">Доба</option>
          <option value="7" selected>Тиждень</option>
          <option value="30">Місяць</option>
          <option value="60">2 місяці</option>
      </select>
  </div>
  <div class="control-group">
      <label>Параметр</label>
      <select id="paramSelect">
          <option value="pm25">PM2.5</option>
          <option value="pm10">PM10</option>
          <option value="co2">CO₂</option>
          <option value="no2">NO₂</option>
          <option value="co">CO</option>
          <option value="noise_level">Шум</option>
      </select>
```

Рис. 3.5 Фрагмент шаблону analytics.html: елементи управління вибором параметрів

Наведений фрагмент шаблону демонструє реалізацію системи вибору параметрів моніторингу за допомогою випадаючих списків. Інтерфейс дозволяє обирати станцію моніторингу, часовий інтервал аналізу та тип екологічного параметра, серед яких підтримуються PM2.5, PM10, CO_2, NO_2, CO та рівень шуму. Для часових інтервалів передбачено режими перегляду даних за добу, тиждень, місяць та два місяці. Використання таких елементів керування забезпечує гнучкість аналітичного модуля та дозволяє формувати вибірки даних відповідно до потреб користувача.

Елементи управління реалізовано через HTML-елементи <select> з динамічним наповненням списку станцій через цикл Jinja2 {% for s in stations %}. При натисканні кнопки «Оновити» JavaScript-функція loadCharts() виконує асинхронні запити до API-ендпоінтів /api/readings та /api/traffic, отримує дані у

форматі JSON та будує чотири графіки Chart.js: динаміку обраного параметра з лінією ГДК, порівняння середніх значень по станціях, графік трафіку та швидкості, а також графік температури та вологості.

Кожен графік використовує специфічний тип візуалізації: динаміка параметра – лінійний графік із заповненням (fill: true), порівняння станцій – стовпчикова діаграма з кольоровим кодуванням (червоний для станцій з перевищенням ГДК, синій – у нормі), трафік – комбінований графік (стовпці для кількості авто, лінія для швидкості з окремою віссю Y).

Сторінка кореляційного аналізу (correlation.html) забезпечує візуальне та числове дослідження залежності рівня забруднення від характеристик транспортного потоку. На рисунку 3.6 показано фрагмент шаблону з елементами вибору станції, екологічного параметра та періоду аналізу.

```
<div class="controls-bar">
    <div class="control-group">
        <label>Станція</label>
        <select id="stationSelect">
            {% for s in stations %}
            <option value="{{ s.id }}">{{ s.name }}</option>
            {% endfor %}
        </select>
    </div>
    <div class="control-group">
        <label>Екологічний параметр</label>
        <select id="paramSelect">
            <option value="pm25">PM2.5</option>
            <option value="pm10">PM10</option>
            <option value="co2">CO₂</option>
            <option value="no2">NO₂</option>
            <option value="co">CO</option>
        </select>
    </div>
    <div class="control-group">
        <label>Період (днів)</label>
        <select id="daysSelect">
            <option value="7">7 днів</option>
            <option value="14">14 днів</option>
            <option value="30" selected>30 днів</option>
            <option value="60">60 днів</option>
        </select>
```

Рис. 3.6 Фрагмент шаблону correlation.html: параметри кореляційного аналізу

При натисканні кнопки «Аналізувати» виконується запит до API /api/correlation_data, який повертає масиви значень забруднення та трафіку, а також обчислені коефіцієнти кореляції Пірсона. На основі цих даних будуються два scatter-plot діаграми: «Трафік vs Забруднення» та «Швидкість vs Забруднення». Під графіками відображається блок інтерпретації, що

автоматично генерує текстовий опис сили та напрямку зв'язку на основі значення коефіцієнта Пірсона.

Алгоритм інтерпретації класифікує силу зв'язку за абсолютним значенням коефіцієнта: $|r| > 0.7$ – сильний зв'язок, $0.4 < |r| \leq 0.7$ – помірний, $|r| \leq 0.4$ – слабкий. Напрямок визначається знаком коефіцієнта: позитивний r означає прямий зв'язок (збільшення трафіку супроводжується зростанням забруднення), негативний – зворотний (збільшення швидкості супроводжується зменшенням забруднення).

Інтерактивна карта станцій (map.html). Картографічна візуалізація реалізована на базі Leaflet.js з використанням тайлів OpenStreetMap. Кожна станція моніторингу відображається на карті кастомним маркером – круглою іконкою з числовим значенням PM2.5 та кольором, що відповідає рівню AQI. На рисунку 3.7 представлено фрагмент JavaScript-коду, що створює кастомні маркери.

```
stations.forEach(s => {
    const reading = latest.find(r => r.station_id === s.id) || {};
    const pm25 = reading.pm25 || 0;
    const color = getColor(pm25);

    const icon = L.divIcon({
        className: 'custom-marker',
        html: `<div style="
            background: ${color};
            width: 32px; height: 32px;
            border-radius: 50%;
            border: 3px solid white;
            box-shadow: 0 2px 8px rgba(0,0,0,0.3);
            display: flex; align-items: center; justify-content: center;
            color: white; font-size: 11px; font-weight: 700;
        ">${Math.round(pm25)}</div>`,
        iconSize: [32, 32],
        iconAnchor: [16, 16],
    });
```

Рис. 3.7 Фрагмент коду map.html: створення кастомних маркерів Leaflet

Для кожної станції створюється об'єкт L.divIcon із HTML-розміткою кола розміром 32×32 пікселі. Колір фону визначається функцією getColor(), яка приймає значення PM2.5 та повертає відповідний колір: зелений (≤12), жовтий (≤25), помаранчевий (≤50), червоний (≤75) або фіолетовий (>75). При натисканні на маркер відкривається popup-вікно з повною інформацією про станцію: назва, тип зони, поточні значення всіх параметрів та кольорова плашка з рівнем AQI.

Додатково для кожної станції відображається напівпрозоре коло (L.circle), радіус якого пропорційний рівню PM2.5. Це створює ефект теплової карти, що дозволяє візуально оцінити зони підвищеного забруднення без необхідності натискати на окремі маркери.

Дашборд (dashboard.html). Головна сторінка системи – дашборд – надає зведену інформацію у чотирьох KPI-картках: кількість активних станцій, середнє значення PM2.5, середня інтенсивність трафіку та кількість алертів за поточну добу. Кожна картка містить іконку, числове значення та підпис. Нижче KPI-карток розміщено таблицю поточного стану станцій із кольоровою індикацією кожного параметра (зелений – норма, жовтий – увага, червоний – перевищення) та бейджем якості повітря.

Під таблицею розміщено два графіки за останню добу: лінійний графік PM2.5 із пунктирною лінією ГДК (25 мкг/м³) та комбінований графік трафіку (стовпці – кількість авто/год, лінія – швидкість). Графіки оновлюються при завантаженні сторінки через асинхронні запити до API /api/readings та /api/traffic з параметром days=1.

У нижній частині дашборду відображається таблиця останніх сповіщень (до 8 записів) із зазначенням часу, станції, параметра, фактичного значення та рівня загрози. Рівень загрози візуалізується кольоровим бейджем: жовтий (увага), помаранчевий (небезпека), червоний (критично).

Сторінка прогнозування (predictions.html). Сторінка прогнозування дозволяє користувачу обрати станцію та параметр, після чого система будує регресійну модель та відображає результати. У верхній частині розміщено чотири KPI-картки: значення R^2 (точність моделі), кількість навчальних зразків, максимальне прогнозоване значення на 24 години та коефіцієнт впливу трафіку.

Основний графік поєднує дві лінії: суцільна синя – фактичні історичні дані (останні 48 точок), пунктирна помаранчева – прогноз на 12 точок (кожні 2 години = 24 години). Під графіком розміщено таблицю з деталізацією прогнозу: час, прогнозоване значення та очікувана кількість транспорту. У нижній частині

відображається блок з параметрами моделі: метод, ознаки, R^2, коефіцієнти регресії та вільний член.

Сторінка звітів (reports.html). Сторінка звітів поєднує три функціональних блоки. Перший блок – форма генерації PDF-звіту з вибором станції та періоду (доба, тиждень, місяць). При натисканні кнопки «Згенерувати PDF» виконується POST-запит до API /api/generate_report, після чого з'являється посилання для завантаження. Другий блок – імпорт та експорт CSV: завантаження файлу через елемент <input type="file"> та кнопка експорту. Третій блок – таблиця раніше згенерованих звітів із можливістю завантаження.

Адаптивний дизайн. Інтерфейс системи реалізовано з урахуванням вимоги NFR-02 (адаптивність від 375 до 2560 px). Файл style.css містить три контрольні точки медіа-запитів. На екранах ширше 1024 px KPI-картки розміщуються у чотири колонки, графіки – у два стовпці. На екранах від 768 до 1024 px KPI-картки перебудовуються у дві колонки, графіки – у один стовпець. На мобільних екранах (менше 768 px) бічна навігація приховується та відкривається кнопкою, усі елементи розміщуються у один стовпець.

3.4 Висновки до розділу 3

У третьому розділі виконано розроблення програмного забезпечення системи моніторингу екологічних параметрів транспортних потоків EcoMonitor.

Обґрунтовано вибір технологічного стеку: Python 3 із Flask 3.0 для серверної частини, SQLite 3 для зберігання даних, Chart.js 4 для графіків, Leaflet.js 1.9 для карти, Jinja2 для шаблонів та CSS3 для адаптивного дизайну. Описано структуру проєкту, що включає 17 файлів, організованих за принципом розділення відповідальності.

Реалізовано п'ять модулів бізнес-логіки: Database (CRUD-операції та ініціалізація з 15 000+ демо-записами), DataProcessor (агрегація, кореляція Пірсона, імпорт/експорт CSV), AlertEngine (трирівнева система сповіщень ГДК,

розрахунок AQI), Predictor (множинна лінійна регресія OLS із моделюванням добового циклу) та ReportGenerator (PDF-звіти з fallback-механізмом).

Розроблено веб-інтерфейс із семи сторінок: дашборд (KPI-картки, таблиця стану, графіки), інтерактивна карта (Leaflet з кастомними маркерами та тепловими зонами), аналітика (4 графіки з вибором станції, періоду та параметра), кореляційний аналіз (scatter-plot з інтерпретацією), прогнозування (графік історії та прогнозу на 24 години), журнал сповіщень та модуль звітів (PDF, CSV). Інтерфейс забезпечує адаптивність від 375 до 2560 px.

4 ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ СИСТЕМИ

4.1 Організація та проведення тестування

Тестування системи EcoMonitor виконано з метою перевірки стабільності роботи програмного забезпечення, коректності реалізації функціональних можливостей та відповідності системи встановленим вимогам. Процес тестування охоплював серверну частину застосунку, веб-інтерфейс, API-запити, модулі аналітичної обробки даних та механізми генерації звітів. Основна увага приділялась перевірці цілісності обробки екологічних і транспортних даних, коректності функціонування модулів прогнозування та формування сповіщень, а також стабільності роботи веб-застосунку під час виконання типових сценаріїв використання.

Організація тестування передбачала поєднання модульного, функціонального та інтеграційного підходів. Модульне тестування використовувалось для перевірки окремих компонентів серверної логіки, функціональне – для контролю коректності роботи інтерфейсу та API-маршрутів, а інтеграційне – для аналізу взаємодії між серверними модулями, базою даних та клієнтською частиною системи. Проведення тестування здійснювалось із використанням тестового клієнта Flask, SQL-запитів до SQLite та аналізу HTTP-відповідей веб-застосунку [20].

Для перевірки працездатності серверної частини виконано модульне тестування основних програмних компонентів системи. Перевірялась коректність створення структури бази даних, робота CRUD-операцій, формування статистичних показників, генерація сповіщень, побудова прогнозних моделей та створення PDF-звітів. Результати модульного тестування серверних модулів наведено у табл. 4.1.

Таблиця 4.1

Результати модульного тестування серверних модулів

Модуль	Перевірка	Статус	Примітка
Database	Ініціалізація схеми БД: 6 таблиць, 3 індекси	OK	8 станцій, 15000+ записів
	Отримання станцій моніторингу: get_stations() повертає 8 записів	OK	Усі поля заповнені
	Отримання останніх вимірювань: get_latest_readings() – 8 записів	OK	По одному на станцію
DataProcessor	Формування статистики: get_summary_stats() – 11 метрик	OK	Значення > 0
	Кореляційний аналіз: get_correlation_data() – r = 0.99	OK	353 спостереження
	Експорт CSV : export_csv() – файл створено	OK	Коректний CSV
AlertEngine	Формування сповіщень: check_reading() – 3 рівні	OK	warning, danger, critical
Predictor	Побудова прогнозу: predict() – 12 точок прогнозу	OK	$R^2 = 0.9808$
ReportGenerator	Генерація PDF-звіту: generate() – PDF створено	OK	5727 байт

Результати модульного тестування (таблиця 4.1) підтверджують коректність роботи кожного серверного модуля. Усі дев'ять перевірок завершилися успішно. Модуль Database коректно ініціалізує схему та генерує демонстраційні дані для 8 станцій із 60-денною історією. Модуль DataProcessor обчислює коефіцієнт кореляції Пірсона між інтенсивністю трафіку та рівнем PM2.5, який становить 0.99, що вказує на сильний позитивний зв'язок. Модуль Predictor будує регресійну модель із коефіцієнтом детермінації $R^2 = 0.9808$, що свідчить про високу пояснювальну здатність.

Функціональне тестування веб-застосунку виконувалось шляхом перевірки HTTP-маршрутів та API-ендпоінтів системи. Аналізувалась

коректність маршрутизації, формування HTML-сторінок, обробки GET та POST-запитів, а також повернення JSON-відповідей для клієнтської частини. Результати тестування маршрутів наведено у табл. 4.2.

Таблиця 4.2

Результати функціонального тестування маршрутів

Маршрут	Метод	Статус	Вимога	Призначення
/	GET	200 OK	FR-03 (дашборд)	Інформаційна панель
/map	GET	200 OK	FR-04 (карта)	Інтерактивна карта
/analytics	GET	200 OK	FR-05 (аналітика)	Аналітичний модуль
/correlation	GET	200 OK	FR-06 (кореляція)	Кореляційний аналіз
/predictions	GET	200 OK	FR-07 (прогнозування)	Прогнозування
/alerts	GET	200 OK	FR-08 (алерти)	Журнал сповіщень
/reports	GET	200 OK	FR-09 (звіти)	Формування звітів
/api/readings	GET	200 OK	FR-01 (дані)	Отримання екологічних даних
/api/traffic	GET	200 OK	FR-02 (трафік)	Отримання транспортних даних
/api/stats	GET	200 OK	FR-03 (статистика)	Формування статистики
/api/correlation_data	GET	200 OK	FR-06 (кореляція)	Дані для кореляційного аналізу
/api/predict	GET	200 OK	FR-07 (прогноз)	Прогнозування
/api/generate_report	POST	200 OK	FR-09 (PDF)	Генерація PDF
/api/export_csv	GET	200 OK	FR-10 (експорт)	Експорт CSV

Усі HTTP-маршрути та API-ендпоінти повернули статус 200 OK, що підтверджує коректність реалізації серверної маршрутизації, взаємодії з базою даних та роботи клієнтського інтерфейсу. Отримані результати свідчать про стабільну роботу механізмів обробки запитів та формування відповідей у форматах HTML і JSON.

Додатково виконано перевірку відповідності реалізованого функціоналу основним вимогам системи (FR-01–FR-10). Оцінювалась підтримка збору

екологічних і транспортних даних, функціонування дашборду, інтерактивної карти, аналітичного модуля, прогнозування, системи сповіщень та механізмів експорту звітів. Результати перевірки наведено у табл. 4.3.

Таблиця 4.3

Перевірка відповідності функціональним вимогам

ID	Вимога	Спосіб перевірки	Результат
FR-01	Збір екологічних даних	БД містить записи CO_2, CO, NO_2, PM2.5, PM10, шум	Виконано
FR-02	Збір транспортних даних	БД містить vehicle_count, avg_speed, congestion	Виконано
FR-03	Дашборд	KPI-картки, таблиця станцій, 2 графіки за добу	Виконано
FR-04	Інтерактивна карта	8 маркерів на карті, popup із показниками, AQI	Виконано
FR-05	Аналітика	4 графіки, вибір станції/періоду/параметра	Виконано
FR-06	Кореляційний аналіз	2 scatter-plot, Пірсон r = 0.99, інтерпретація	Виконано
FR-07	Прогнозування	12 точок прогнозу, $R^2 = 0.98$, графік	Виконано
FR-08	Система алертів	3 рівні сповіщень, журнал, лічильник на дашборді	Виконано
FR-09	Генерація PDF	Звіт із 3 секціями, завантаження файлу	Виконано
FR-10	Імпорт/експорт CSV	Імпорт файлу, експорт із фільтрацією	Виконано

Результати тестування підтверджують працездатність реалізованого програмного забезпечення та коректність функціонування основних компонентів системи EcoMonitor. Система забезпечує повний цикл обробки даних: від отримання екологічних і транспортних показників до формування аналітичних звітів, прогнозування рівня забруднення та автоматичного створення сповіщень при перевищенні нормативних значень.

4.2 Аналіз результатів роботи системи

Для аналізу результатів роботи системи виконано серію тестових сценаріїв із використанням демонстраційного набору даних, що включає 8 станцій моніторингу Києва з 60-денною історією вимірювань. У цьому підрозділі представлено та проаналізовано результати роботи кожного функціонального модуля.

На рисунку 4.1 представлено головну сторінку системи – дашборд із зведеною інформацією. У верхній частині розміщено чотири KPI-картки: 8 активних станцій, середнє PM2.5 = 46.3 мкг/м³, середня інтенсивність трафіку 1431 авто/год та 244 алерти за поточну добу. Нижче розміщено таблицю поточного стану станцій із кольоровою індикацією: зелені значення відповідають нормі, жовті – перевищенню першого порогу, червоні – суттєвому перевищенню ГДК.

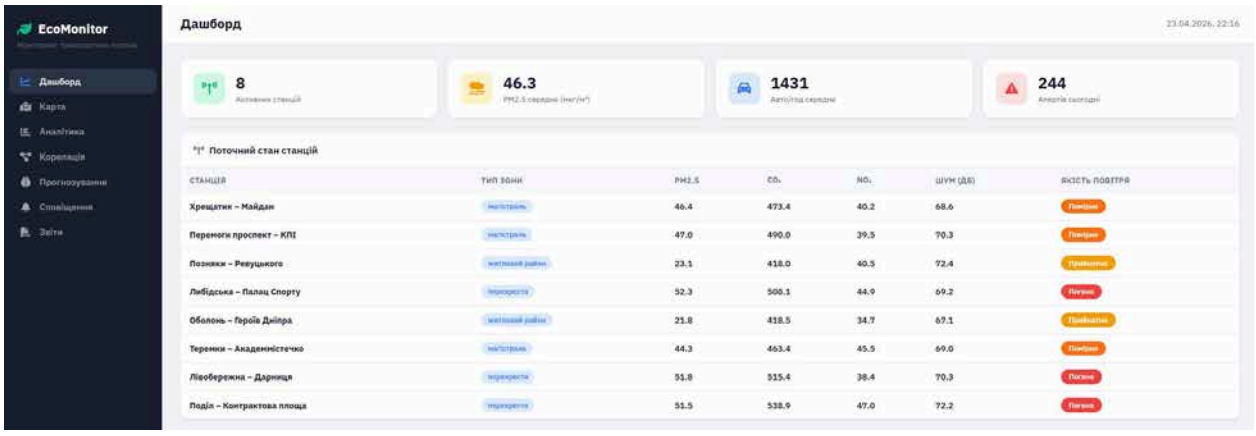

Рис. 4.1 Дашборд системи EcoMonitor: KPI-картки та таблиця стану станцій

Таблиця стану демонструє чітку просторову диференціацію: станції на магістралях та перехрестях (Хрещатик, Либідська, Лівобережна, Поділ) мають рівень якості повітря «Помірно» або «Погано» (PM2.5 від 44 до 52 мкг/м³), тоді як станції у житлових районах (Позняки, Оболонь) демонструють рівень «Прийнятно» (PM2.5 від 21 до 23 мкг/м³). Ця закономірність відповідає

очікуванням, оскільки магістралі та перехрестя характеризуються значно вищою інтенсивністю транспортного потоку.

На рисунку 4.2 представлено графік динаміки PM2.5 за останню добу. Графік чітко демонструє добовий цикл забруднення: мінімальні значення (8–15 мкг/м³) спостерігаються в нічні години (00:00–05:00), різке зростання відбувається під час ранкового піку (06:00–09:00) із досягненням значень 70–95 мкг/м³, помірне зниження у денний час (10:00–16:00) та повторне зростання під час вечірнього піку (17:00–20:00). Пунктирна червона лінія позначає ГДК PM2.5 = 25 мкг/м³ – добре видно, що протягом більшої частини доби значення перевищує цей поріг.

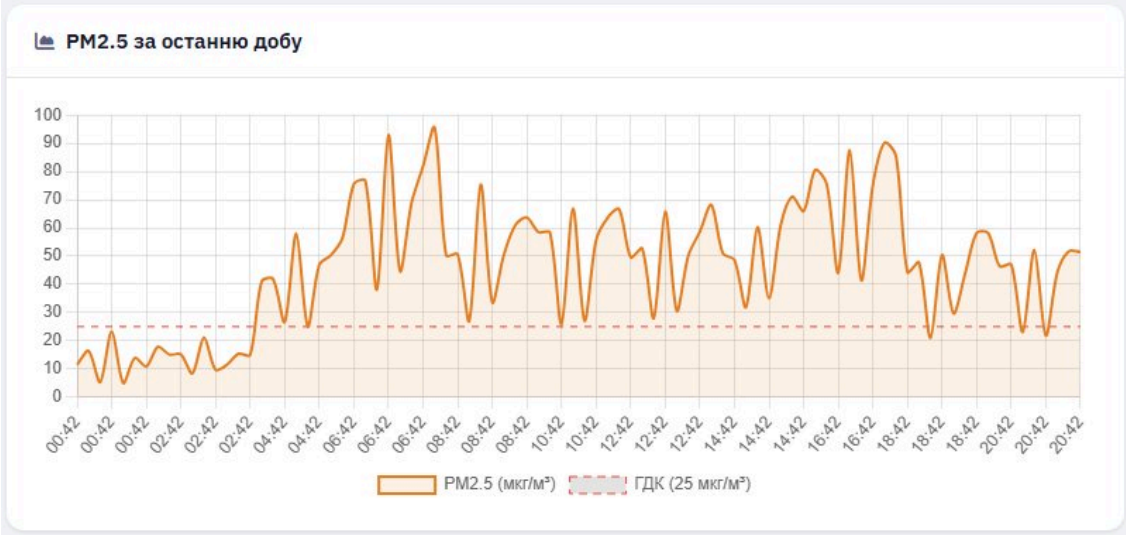

Рис. 4.2 Графік PM2.5 за останню добу з лінією ГДК

На рисунку 4.3 представлено комбінований графік інтенсивності трафіку за ту ж добу. Стовпчики відображають кількість транспортних засобів на годину, лінія – середню швидкість руху. Спостерігається чіткий зворотний зв'язок: у години пік (06:00–09:00 та 16:00–19:00) кількість авто сягає 2500–3100 на годину, а швидкість знижується до 10–20 км/год (затори). У нічний час трафік мінімальний (200–500 авто/год), швидкість зростає до 50–55 км/год. Ця динаміка корелює з графіком PM2.5: піки забруднення збігаються з піками трафіку.

Рис. 4.3 Комбінований графік інтенсивності трафіку та швидкості за добу

Інтерактивна карта. На рисунку 4.4 представлено інтерактивну карту станцій моніторингу на базі Leaflet.js із тайлами OpenStreetMap. Кожна станція позначена круглим маркером, колір якого відповідає рівню AQI: помаранчевий – «Помірно» (PM2.5 від 25 до 50), червоний – «Погано» (PM2.5 від 50 до 75). Числа на маркерах відображають поточне значення PM2.5. Відкрите popup-вікно станції «Перемоги проспект – КПІ» демонструє повний набір показників: PM2.5 = 47, PM10 = 82.2, CO_2 = 490, NO_2 = 39.5, Шум = 70.3 дБ.

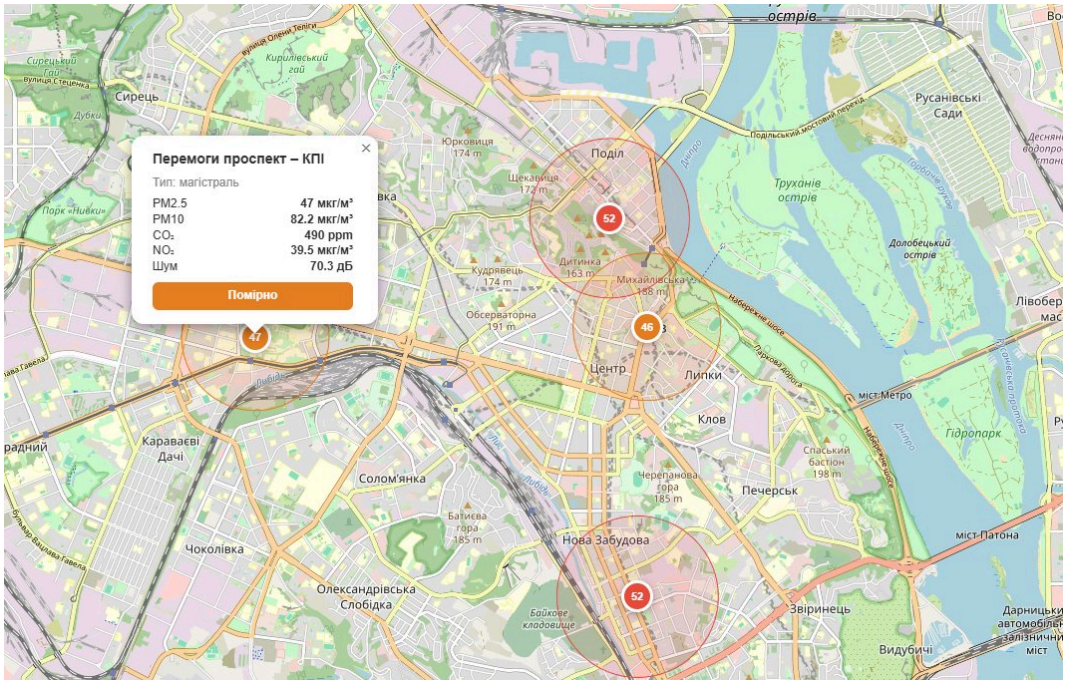

Рис. 4.4 Інтерактивна карта станцій моніторингу з popup-вікном

Навколо кожного маркера відображається напівпрозоре коло, радіус якого пропорційний рівню PM2.5, що створює ефект теплової карти. На карті чітко видно, що зони підвищеного забруднення зосереджені вздовж основних магістралей та на перехрестях, тоді як житлові райони (Позняки, Оболонь) мають менший радіус зон забруднення.

Аналітика. Модуль аналітики надає чотири типи графіків із можливістю вибору станції, періоду та параметра. На рисунку 4.5 показано графік динаміки CO_2 за місяць для усіх станцій із лінією ГДК (500 ppm). Графік демонструє чітку добову циклічність: значення CO_2 коливаються від 100–200 ppm (нічний мінімум) до 700–950 ppm (денний максимум під час годин пік), причому ГДК 500 ppm перевищується щоденно протягом 8–12 годин.

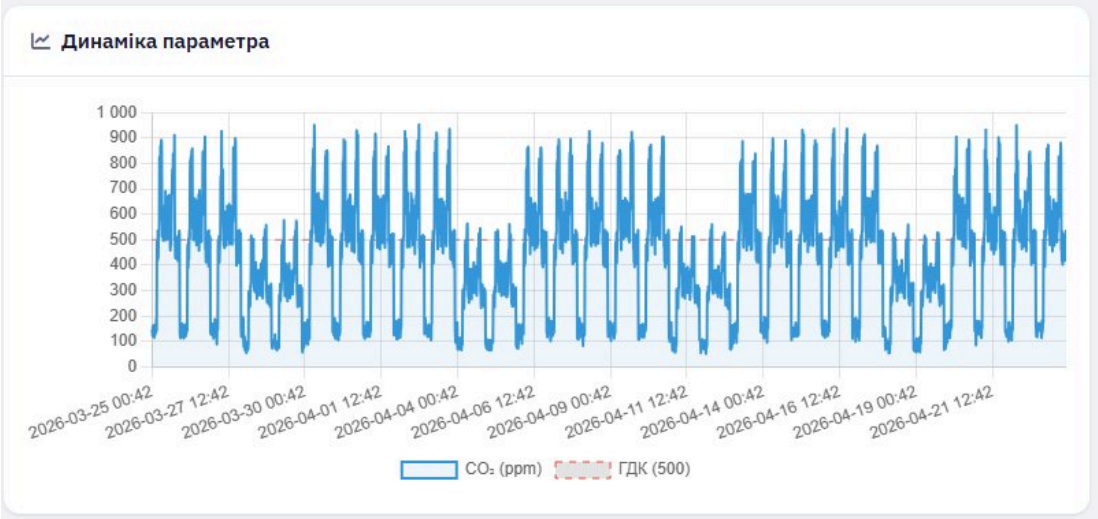

Рис. 4.5 Динаміка CO_2 за місяць із лінією ГДК

На рисунку 4.6 представлено порівняння середніх значень PM2.5 по станціях за тиждень. Стовпчики забарвлені у два кольори: синій – значення в межах ГДК, червоний – перевищення. Станції у житлових районах (Позняки та Оболонь) мають середні значення близько 22 мкг/м³ (нижче ГДК = 25), тоді як станції на магістралях та перехрестях – від 40 до 49 мкг/м³ (значне перевищення). Ця різниця у 1.8–2.2 рази відповідає даним Європейського агентства довкілля про підвищену концентрацію забруднювачів поблизу автомагістралей [2].

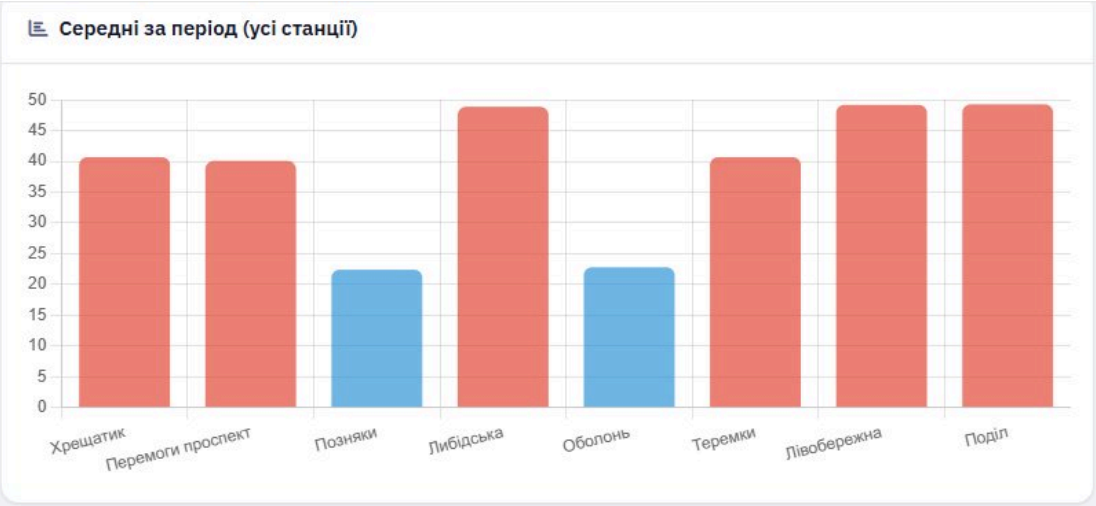

Рис. 4.6 Порівняння середніх PM2.5 по станціях за тиждень

На рисунку 4.7 показано тижневий графік трафіку та швидкості, що демонструє тижневу циклічність: у робочі дні чітко виділяються два піки (ранковий та вечірній), у вихідні інтенсивність трафіку знижується на 35–45%, а швидкість руху зростає.

Рис. 4.7 Трафік та швидкість руху за тиждень

На рисунку 4.8 представлено графік температури та вологості повітря за тиждень. Температура коливається в діапазоні 13–19°C із добовими циклами (мінімум вночі, максимум вдень). Вологість має зворотну залежність від температури: 45–55% вдень та 65–75% вночі. Ці метеорологічні дані є

додатковим контекстом для інтерпретації екологічних показників, оскільки температура та вологість впливають на розсіювання забруднювачів.

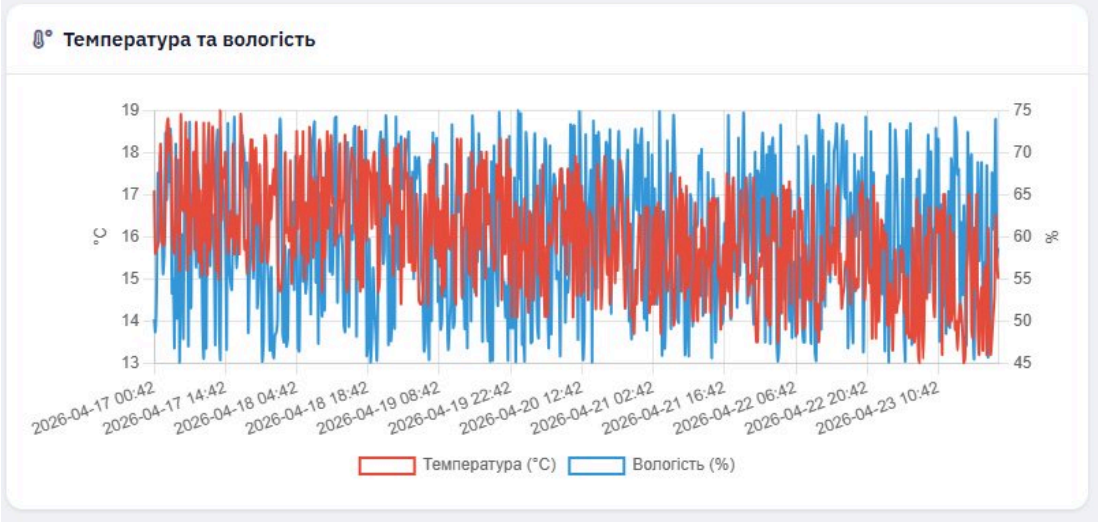

Рис. 4.8 Температура та вологість повітря за тиждень

Кореляційний аналіз. Модуль кореляційного аналізу дозволяє дослідити залежність рівня забруднення від характеристик транспортного потоку. На рисунку 4.9 представлено scatter-plot діаграму залежності PM2.5 від кількості транспортних засобів на годину для станції «Хрещатик – Майдан» за 30 днів. Точки утворюють чітку висхідну тенденцію: при збільшенні кількості авто від 200 до 3200 на годину концентрація PM2.5 зростає від 5–10 до 75–85 мкг/м³. Коефіцієнт кореляції Пірсона r = 0.99 підтверджує наявність сильного прямого лінійного зв'язку.

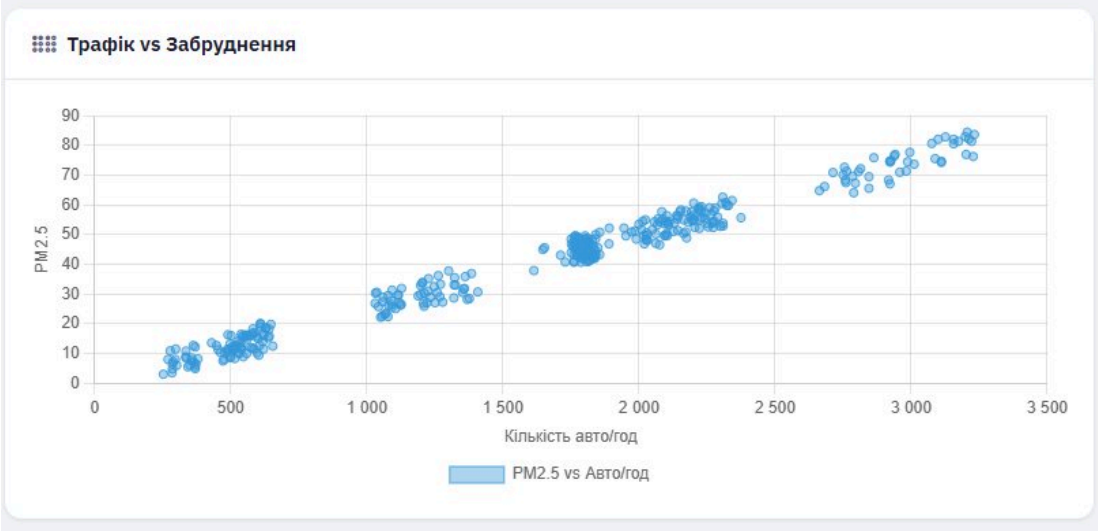

Рис 4.9 Scatter-plot: залежність PM2.5 від кількості авто/год (r = 0.99)

На рисунку 4.10 представлено scatter-plot діаграму залежності PM2.5 від швидкості руху. Точки демонструють чітку спадну тенденцію: при зниженні швидкості від 55 до 10 км/год концентрація PM2.5 зростає від 5–15 до 75–85 мкг/м³. Це підтверджує, що затори (низька швидкість) супроводжуються підвищенням забруднення через тривалу роботу двигунів на холостому ходу та часті цикли розгону–гальмування. Коефіцієнт кореляції має від'ємне значення, що вказує на зворотний зв'язок.

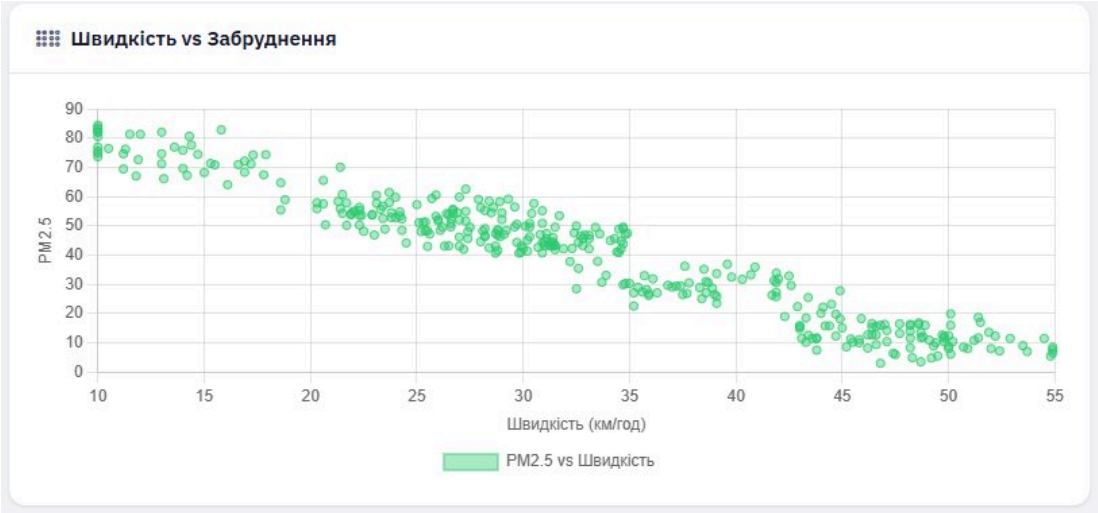

Рис. 4.10 Scatter-plot: залежність PM2.5 від швидкості руху

Результати кореляційного аналізу узгоджуються з даними наукових досліджень [3, 13]: інтенсивність транспортного потоку є домінуючим фактором, що визначає рівень забруднення повітря вздовж міських магістралей. Сильний від'ємний зв'язок зі швидкістю підтверджує негативний екологічний ефект заторів.

Прогнозування. На рисунку 4.11 представлено результати роботи модуля прогнозування для станції «Хрещатик – Майдан» за параметром PM2.5. У верхній частині відображено чотири метрики моделі: R^2 = 0.9808 (модель пояснює 98% дисперсії), 359 навчальних зразків, максимальний прогноз на 24 години = 67.3 мкг/м³, коефіцієнт впливу трафіку = 0.024687.

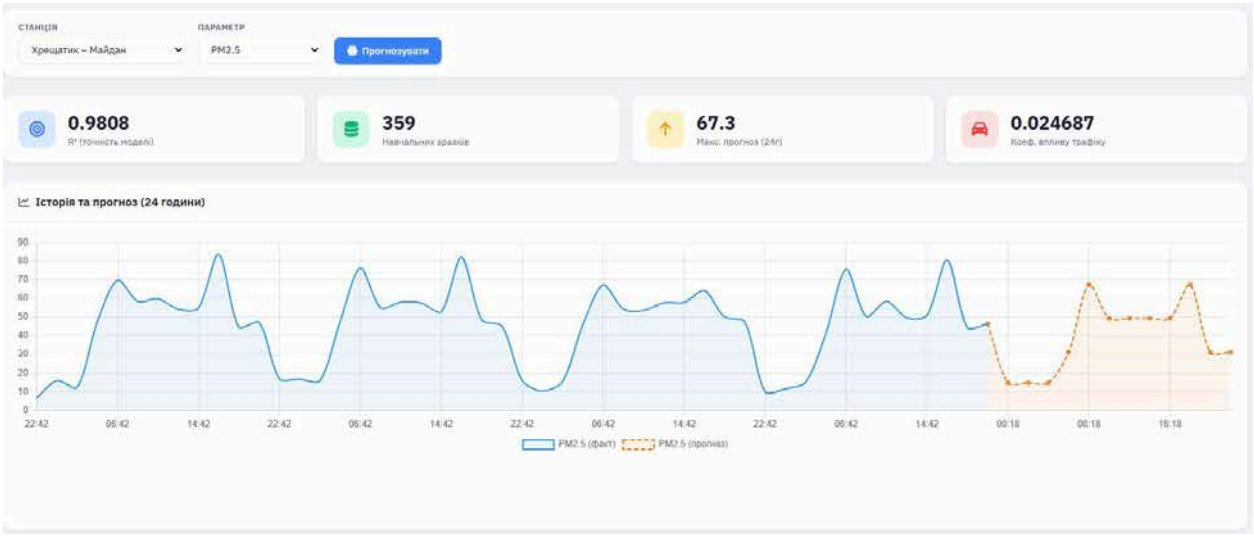

Рис. 4.11 Прогнозування PM2.5: історія (синя лінія) та прогноз на 24 години (помаранчева пунктирна)

Графік поєднує фактичні дані (суцільна синя лінія) та прогноз (пунктирна помаранчева). Фактичні дані демонструють характерну добову циклічність із піками від 60 до 83 мкг/м³ та мінімумами від 5 до 20 мкг/м³. Прогнозна лінія продовжує цю тенденцію: модель коректно передбачає зростання забруднення під час ранкового піку (прогноз до 67 мкг/м³) та зниження у нічні години. Високе значення $R^2 = 0.98$ свідчить про те, що три обрані ознаки (кількість авто, швидкість, заторність) достатньо описують варіативність рівня PM2.5.

Коефіцієнт впливу трафіку 0.024687 означає, що збільшення кількості автомобілів на 100 одиниць на годину призводить до зростання концентрації PM2.5 приблизно на 2.5 мкг/м³ за інших рівних умов. Цей результат має практичне значення для прийняття управлінських рішень щодо обмеження трафіку на найбільш забруднених ділянках.

4.3 Рекомендації щодо впровадження та експлуатації

Результати тестування та аналізу функціонування системи EcoMonitor дозволили сформувати рекомендації щодо впровадження програмного забезпечення у реальних умовах експлуатації. Під час розроблення рекомендацій враховувались особливості роботи систем моніторингу у міському середовищі,

необхідність безперервного збору екологічних даних, підтримка веб-доступу до аналітичних модулів та забезпечення стабільної роботи серверної частини при зміні обсягів інформації.

Для розгортання серверної частини системи достатньо використання комп’ютера або віртуального сервера середнього рівня продуктивності. Основні апаратні та програмні вимоги до функціонування системи наведено у табл. 4.4.

Таблиця 4.4

Мінімальні та рекомендовані вимоги до системи

Компонент	Мінімум	Рекомендовано
Процесор	2 ядра, 2 ГГц	4 ядра, 3 ГГц
Оперативна пам'ять	4 ГБ	8 ГБ
Дисковий простір	500 МБ	2 ГБ (SSD)
Операційна система (сервер)	Ubuntu 20.04 / Win 10	Ubuntu 22.04 LTS
Python	3.10	3.11 або 3.12
Веб-браузер (клієнт)	Chrome 90+ / Firefox 90+	Chrome 120+ / Edge 120+
Мережеве підключення	100 Мбіт/с	1 Гбіт/с

Наведені вимоги забезпечують стабільне функціонування серверної частини, виконання аналітичних обчислень та підтримку інтерактивного веб-інтерфейсу. Використання SSD-накопичувачів дозволяє прискорити виконання SQL-запитів та підвищує швидкість формування PDF-звітів.

Розгортання системи здійснюється шляхом встановлення Python-залежностей із файлу requirements.txt, ініціалізації структури бази даних за допомогою скрипта init_db.py та запуску Flask-застосунку через app.py. Для промислової експлуатації доцільним є використання WSGI-сервера Gunicorn у поєднанні із зворотним проксі-сервером Nginx, що забезпечує підтримку HTTPS, кешування статичних ресурсів та балансування навантаження [20]. Такий підхід дозволяє підвищити стабільність роботи системи та забезпечити підтримку одночасних підключень користувачів.

Одним із перспективних напрямів розвитку інфраструктури системи є перехід до контейнерної архітектури із використанням Docker. Контейнеризація дозволяє ізолювати серверну частину застосунку, веб-сервер та систему керування базами даних у незалежних середовищах виконання, що спрощує перенесення системи між різними серверними платформами. Для автоматизації процесів оновлення та розгортання доцільним є використання CI/CD-пайплайнів на базі GitHub Actions або GitLab CI, які забезпечують автоматичне тестування та розгортання нових версій програмного забезпечення після проходження перевірок коректності коду.

Під час експлуатації системи важливим аспектом є інтеграція із реальними джерелами екологічних і транспортних даних. Поточна версія системи використовує демонстраційний набір даних, однак архітектура серверної частини підтримує підключення зовнішніх API та сенсорних станцій моніторингу. Основні напрями інтеграції наведено у табл. 4.5.

Таблиця 4.5

Напрями інтеграції EcoMonitor із зовнішніми джерелами даних

Джерело даних	Технологія інтеграції	Призначення
Сенсорні станції	MQTT / REST API	Передавання екологічних вимірювань
OpenAQ, SaveEcoBot	REST API	Верифікація показників якості повітря
Google Traffic, HERE Traffic	REST API	Отримання транспортних показників
PostgreSQL	SQL / ORM	Масштабування системи

Використання зовнішніх API дозволить забезпечити автоматичне надходження даних у режимі реального часу та підвищить практичну цінність системи для екологічного моніторингу міського середовища. У випадку збільшення кількості станцій моніторингу понад 50 доцільною є міграція із SQLite на PostgreSQL, що забезпечить підтримку конкурентного доступу та масштабованість серверної частини.

Для підтримки стабільної роботи системи необхідним є виконання регулярного технічного обслуговування. Рекомендовані операції супроводу наведено у табл. 4.6.

Таблиця 4.6

Рекомендовані операції технічного обслуговування

Операція	Періодичність	Призначення
Резервне копіювання БД	Щоденно	Захист від втрати даних
Очищення каталогу reports	Щотижнево	Оптимізація дискового простору
Оновлення нормативів ГДК	Щомісячно	Актуалізація екологічних порогів
Перебудова регресійної моделі	Щоквартально	Підтримка точності прогнозування
Оптимізація бази даних (VACUUM)	За потреби	Зменшення розміру БД

Під час експлуатації системи в умовах українських міст необхідно враховувати можливі перебої електропостачання та нестабільність каналів зв’язку. З цією метою доцільним є використання елементів периферійних обчислень (Edge Computing), за яких станції моніторингу оснащуються локальним буфером пам’яті та акумуляторним живленням. У випадку втрати з’єднання із сервером станція повинна продовжувати накопичення вимірювань із подальшою синхронізацією даних після відновлення зв’язку. Такий підхід забезпечує цілісність часових рядів і підтримує коректність роботи модулів прогнозування та кореляційного аналізу.

Важливим напрямом подальшого розвитку системи є підвищення рівня інформаційної безпеки. Для захисту API-ендпоінтів доцільним є використання JWT-токенів або API-ключів, що дозволить запобігти передаванню фальсифікованих екологічних даних. Додатково рекомендовано впровадження механізмів Rate Limiting для обмеження кількості запитів до серверної частини та захисту системи від перевантаження. Конфіденційні параметри конфігурації, включаючи SECRET_KEY та параметри підключення до бази даних, доцільно зберігати у змінних середовища та файлах .env, виключених із системи контролю версій.

Для оптимізації продуктивності системи при збільшенні кількості користувачів рекомендовано використання Redis як проміжного шару кешування. Кешування результатів статистичних обчислень дозволить зменшити кількість SQL-запитів до реляційної бази даних та забезпечить виконання нефункціональної вимоги щодо часу відгуку системи не більше 3 секунд. Використання механізмів кешування є особливо актуальним для дашбордів реального часу та модулів кореляційного аналізу, які виконують значну кількість агрегованих обчислень.

Практичне впровадження системи EcoMonitor створює можливість використання результатів екологічного моніторингу для підтримки управлінських рішень у сфері міського планування та транспортної інфраструктури. Аналітичні звіти та результати прогнозування можуть використовуватись для оптимізації режимів роботи світлофорних об’єктів, оцінювання ефективності транспортних обмежень, планування озеленення міських територій та визначення зон із підвищеним рівнем екологічного навантаження. Реалізований програмний комплекс може бути використаний як основа для створення муніципальних інформаційно-аналітичних систем екологічного моніторингу та підтримки концепцій Smart City.

4.4 Висновки до розділу 4

У четвертому розділі проведено тестування та аналіз результатів роботи системи моніторингу екологічних параметрів транспортних потоків EcoMonitor.

Модульне тестування підтвердило коректність роботи п'яти серверних модулів (9 перевірок, усі успішні). Функціональне тестування усіх 14 маршрутів веб-додатку (7 сторінок та 7 API-ендпоінтів) завершилося зі статусом 200 OK. Перевірку пройшли усі 10 функціональних вимог FR-01–FR-10.

Аналіз результатів роботи системи підтвердив наявність очікуваних закономірностей: добову циклічність рівня забруднення із піками під час годин пік, просторову диференціацію між магістралями та житловими районами

(різниця PM2.5 у 1.8–2.2 рази), сильну позитивну кореляцію між інтенсивністю трафіку та рівнем PM2.5 ($r = 0.99$) та сильну від'ємну кореляцію зі швидкістю руху. Регресійна модель продемонструвала високу точність прогнозування ($R^2 = 0.98$).

Сформульовано рекомендації щодо впровадження: визначено апаратні та програмні вимоги, описано порядок розгортання (три кроки), надано рекомендації щодо промислового розгортання (Gunicorn + Nginx), інтеграції з реальними сенсорами та регулярного обслуговування. Визначено п'ять перспективних напрямків подальшого розвитку системи.

ВИСНОВКИ

У бакалаврській кваліфікаційній роботі виконано проєктування та розробку веб-орієнтованої системи моніторингу екологічних параметрів транспортних потоків мегаполісів EcoMonitor. У процесі виконання роботи отримано такі результати.

1. Проведено аналіз предметної області та досліджено особливості транспортних потоків мегаполісів як джерела екологічного впливу. Визначено ключові характеристики, що зумовлюють специфіку транспортного забруднення: добову циклічність інтенсивності руху, просторову неоднорідність розподілу забруднювачів та залежність обсягу викидів від режиму руху. Встановлено, що рух у заторі збільшує питомі викиди монооксиду вуглецю на 50–80% та оксидів азоту на 30–60% порівняно з вільним рухом.
2. Систематизовано перелік екологічних параметрів (PM2.5, PM10, NO_2, CO, CO_2, рівень шуму) та параметрів транспортного потоку (інтенсивність, швидкість, заторність), що підлягають моніторингу, із зазначенням гранично допустимих концентрацій відповідно до рекомендацій ВООЗ та чинного законодавства України. Проведено порівняльний аналіз чотирьох існуючих платформ (OpenAQ, AQICN, Sensor.Community, SaveEcoBot), який виявив спільний недолік – відсутність інтеграції екологічних даних із характеристиками транспортних потоків.
3. Спроєктовано трирівневу архітектуру системи (клієнтський, серверний та рівень даних), побудовано комплекс UML-діаграм: варіантів використання (3 актори, 10 варіантів), діяльності, послідовності, класів та розгортання. Розроблено блок-схему алгоритму розрахунку індексу якості повітря (AQI).
4. Спроєктовано реляційну базу даних із шести таблиць нормалізованих до третьої нормальної форми.

5. Обрано та обґрунтовано стек технологій: Python 3 із Flask 3.0 для серверної частини, SQLite 3 для зберігання даних, Chart.js та Leaflet.js для клієнтського інтерфейсу. Реалізовано п'ять модулів бізнес-логіки: Database (CRUD-операції), DataProcessor (агрегація та кореляційний аналіз Пірсона), AlertEngine (генерація сповіщень при перевищенні ГДК), Predictor (множинна лінійна регресія для прогнозування рівня забруднення) та ReportGenerator (генерація PDF-звітів).
6. Розроблено веб-інтерфейс системи, що включає сім сторінок: дашборд із KPI-картками та графіками, інтерактивну карту станцій із кольоровою індикацією AQI, модуль аналітики з часовими рядами та порівнянням станцій, модуль кореляційного аналізу зі scatter-plot діаграмами та обчисленням коефіцієнта Пірсона, модуль прогнозування з візуалізацією історії та прогнозу на 24 години, журнал сповіщень та модуль генерації PDF-звітів із можливістю імпорту й експорту даних у форматі CSV.
7. Проведено функціональне тестування системи, що охопило усі десять функціональних вимог (FR-01–FR-10). Усі 14 маршрутів веб-додатку (7 сторінок та 7 API-ендпоінтів) коректно повертають відповіді. Кореляційний аналіз підтвердив наявність сильного позитивного зв'язку між інтенсивністю транспортного потоку та рівнем PM2.5 ($r = 0,99$). Регресійна модель продемонструвала високу пояснювальну здатність ($R^2 = 0,98$). Сформульовано рекомендації щодо впровадження системи в реальних умовах, що включають розгортання на виділеному сервері, інтеграцію з реальними сенсорами та регулярне оновлення порогових значень ГДК.

Розроблена система EcoMonitor є функціонально завершеним програмним продуктом, готовим до впровадження. Подальший розвиток системи може включати інтеграцію з реальними сенсорними мережами через протокол MQTT, застосування ансамблевих методів машинного навчання для підвищення точності прогнозування, реалізацію push-сповіщень для мобільних пристроїв та розширення переліку контрольованих параметрів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. World Health Organization. Ambient (outdoor) air pollution. WHO Fact Sheet. Geneva : WHO, 2022. URL: https://www.who.int/news-room/fact-sheets/detail/ambient-(outdoor)-air-quality-and-health (дата звернення: 15.03.2026).
2. World Health Organization. WHO global air quality guidelines: particulate matter (PM2.5 and PM10), ozone, nitrogen dioxide, sulfur dioxide and carbon monoxide. Geneva : WHO, 2021. 273 p. URL: https://www.who.int/publications/i/item/9789240034228 (дата звернення: 15.03.2026).
3. European Environment Agency. Air quality in Europe 2022. EEA Report No 05/2022. Luxembourg : Publications Office of the European Union, 2022. 66 p. URL: https://www.eea.europa.eu/publications/air-quality-in-europe-2022 (дата звернення: 15.03.2026).
4. World Health Organization. Environmental Noise Guidelines for the European Region. Copenhagen : WHO Regional Office for Europe, 2018. 160 p. URL: https://www.who.int/europe/publications/i/item/9789289053563 (дата звернення: 10.03.2026).
5. Про охорону атмосферного повітря : Закон України від 16.10.1992 № 2707-XII. Відомості Верховної Ради України. 1992. № 50. Ст. 678. URL: https://zakon.rada.gov.ua/laws/show/2707-12 (дата звернення: 10.03.2026).
6. Про дорожній рух : Закон України від 30.06.1993 № 3353-XII. Відомості Верховної Ради України. 1993. № 31. Ст. 338. URL: https://zakon.rada.gov.ua/laws/show/3353-12 (дата звернення: 10.03.2026).
7. Pope C. A., Dockery D. W. Health effects of fine particulate air pollution: Lines that connect. Journal of the Air & Waste Management Association. 2006. Vol. 56, No. 6. P. 709–742. DOI: 10.1080/10473289.2006.10464485.
8. Cai H., Xie S. Traffic-related air pollution modeling during the 2008 Beijing Olympic Games: the effects of an odd-even day vehicle restriction scheme. Science

of The Total Environment. 2011. Vol. 409, No. 10. P. 1935–1948. DOI: 10.1016/j.scitotenv.2011.01.025.

9. Timmers V., Achten P. Non-exhaust PM emissions from electric vehicles. Atmospheric Environment. 2016. Vol. 134. P. 10–17. DOI: 10.1016/j.atmosenv.2016.03.017.
10. Gulia S., Nagendra S. M., Khare M., Khanna I. Urban air quality management – A review. Atmospheric Pollution Research. 2015. Vol. 6, No. 2. P. 286–304. DOI: 10.5094/APR.2015.033.
11. Yi W. Y., Lo K. M., Mak T., Leung K. S., Leung Y., Meng M. L. A survey of wireless sensor network based air pollution monitoring systems. Sensors. 2015. Vol. 15, No. 12. P. 31392–31427. DOI: 10.3390/s151229859.
12. Zheng Y., Liu F., Hsieh H. P. U-Air: When Urban Air Quality Inference Meets Big Data. Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. New York : ACM, 2013. P. 1436–1444. DOI: 10.1145/2487575.2488188.
13. Vardoulakis S., Fisher B. E., Pericleous K., Gonzalez-Flesca N. Modelling air quality in street canyons: a review. Atmospheric Environment. 2003. Vol. 37, No. 2. P. 155–182. DOI: 10.1016/S1352-2310(02)00857-9.
14. Ellison R. B., Greaves S. P., Hensher D. A. Five years of London's low emission zone: Effects on vehicle fleet composition and air quality. Transportation Research Part D: Transport and Environment. 2013. Vol. 23. P. 25–33. DOI: 10.1016/j.trd.2013.03.006.
15. Harrison R. M., Allan J., Carruthers D., Heal M. R., Lewis A. C., Marner B., Murrells T., Williams A. Non-exhaust vehicle emissions of particulate matter and VOC from road traffic: A review. Atmospheric Environment. 2021. Vol. 262. 118592. DOI: 10.1016/j.atmosenv.2021.118592.
16. Tam V. W. Y., Le K. N., Tran C. N. N., Kubota T. The Relationship between Roadside PM Concentration and Traffic Characterization: A Case Study in Macao. Sustainability. 2023. Vol. 15, No. 14. 10993. DOI: 10.3390/su151410993.

17. Schmitz S., Towers S., Villena G., Caseiro A., Wegener R., Stenchikov G., Lee S. Quantifying Impacts of Local Traffic Policies on PM Concentrations Using Low Cost Sensors in Berlin. Aerosol and Air Quality Research. 2024. Vol. 24, No. 6. 240050. DOI: 10.4209/aaqr.240050.
18. Popov O., Iatsyshyn A., Kovach V., Artemchuk V., Kameneva I., Taraduda D., Sobyna V., Sokolov D., Dement M., Yatsyshyn T. Risk Assessment for the Population of Kyiv, Ukraine as a Result of Atmospheric Air Pollution. Journal of Health and Pollution. 2020. Vol. 10, No. 25. 200303. DOI: 10.5696/2156-9614-10.25.200303.
19. Wang J., Xu H., Lin Y., Ma C., Yin H. Predicting the Concentration Levels of PM2.5 and O3 for Highly Urbanized Areas Based on Machine Learning Models. Sustainability. 2025. Vol. 17, No. 20. 9211. DOI: 10.3390/su17209211.
20. Piscitello A., Bianco C., Casasso A., Sethi R. Non-exhaust traffic emissions: Sources, characterization, and mitigation measures. Science of The Total Environment. 2021. Vol. 766. 144440. DOI: 10.1016/j.scitotenv.2020.144440.
21. Wei J., Li Z., Lyapustin A., Wang J., Dubovik O., Schwartz J., Sun L., Li C., Liu S., Zhu T. First close insight into global daily gapless 1 km PM2.5 pollution, variability, and health impact. Nature Communications. 2023. Vol. 14. 8349. DOI: 10.1038/s41467-023-43862-3.
22. Hu Y., Peng J., Han Y., Li T., Cao H. A systematic review of urban road traffic CO2 emission models. Carbon Footprints. 2025. Vol. 4. 12. DOI: 10.20517/cf.2025.12.
23. OpenAQ. Open Air Quality Data Platform : офіційний веб-сайт проєкту. URL: https://openaq.org/ (дата звернення: 20.03.2026).
24. World Air Quality Index Project (AQICN) : офіційний веб-сайт проєкту. URL: https://aqicn.org/ (дата звернення: 20.03.2026).
25. Sensor.Community : офіційний веб-сайт громадської мережі сенсорів. URL: https://sensor.community/en/ (дата звернення: 20.03.2026).
26. SaveEcoBot : офіційний веб-сайт української платформи моніторингу якості повітря. URL: https://www.saveecobot.com/ (дата звернення: 20.03.2026).

27. European Centre for Medium-Range Weather Forecasts. Copernicus Atmosphere Monitoring Service (CAMS). URL: https://atmosphere.copernicus.eu/ (дата звернення: 18.03.2026).
28. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3rd ed. Boston : Addison-Wesley, 2003. 175 p.
29. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Boston : Addison-Wesley, 1994. 395 p.
30. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures : Ph.D. dissertation. University of California, Irvine, 2000. 180 p. URL: https://ics.uci.edu/~fielding/pubs/dissertation/top.htm (дата звернення: 10.02.2026).
31. Sommerville I. Software Engineering. 10th ed. Harlow : Pearson, 2015. 816 p.
32. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner's Approach. 9th ed. New York : McGraw-Hill, 2020. 736 p.
33. Myers G. J., Sandler C., Badgett T. The Art of Software Testing. 3rd ed. Hoboken : John Wiley & Sons, 2011. 256 p.
34. ISO/IEC 25010:2023. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Product quality model. Geneva : International Organization for Standardization, 2023. URL: https://www.iso.org/standard/78176.html (дата звернення: 15.02.2026).
35. ISO/IEC/IEEE 29148:2018. Systems and software engineering – Life cycle processes – Requirements engineering. Geneva : International Organization for Standardization, 2018. URL: https://www.iso.org/standard/72089.html (дата звернення: 15.02.2026).
36. IEEE Std 829-2008. IEEE Standard for Software and System Test Documentation. New York : Institute of Electrical and Electronics Engineers, 2008. 150 p. DOI: 10.1109/IEEESTD.2008.4578383.
37. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання. Київ : ДП «УкрНДНЦ», 2016. 16 с.

URL: https://archives.gov.ua/wp-content/uploads/DSTU_8302_2015.pdf (дата звернення: 10.04.2026).

38. Connolly T., Begg C. Database Systems: A Practical Approach to Design, Implementation, and Management. 6th ed. Harlow : Pearson, 2015. 1440 p.
39. Codd E. F. A relational model of data for large shared data banks. Communications of the ACM. 1970. Vol. 13, No. 6. P. 377–387. DOI: 10.1145/362384.362685.
40. Owens M. The Definitive Guide to SQLite. 2nd ed. New York : Apress, 2010. 440 p.
41. Codd E. F. Further Normalization of the Data Base Relational Model. IBM Research Report RJ909. San Jose : IBM, 1971. URL: https://forest.moscito.org/cfdocs/whitepapers/Further_Normalization.pdf (дата звернення: 12.02.2026).
42. Grinberg M. Flask Web Development: Developing Web Applications with Python. 2nd ed. Sebastopol : O'Reilly Media, 2018. 316 p.
43. Python Software Foundation. Python 3 Documentation : офіційна документація. URL: https://docs.python.org/3/ (дата звернення: 20.01.2026).
44. SQLite Consortium. SQLite Documentation : офіційна документація. URL: https://www.sqlite.org/docs.html (дата звернення: 20.01.2026).
45. Pallets Projects. Flask Documentation : офіційна документація. URL: https://flask.palletsprojects.com/ (дата звернення: 22.03.2026).
46. WHATWG. HTML Living Standard. URL: https://html.spec.whatwg.org/ (дата звернення: 18.01.2026).
47. W3C. CSS Snapshot 2023. W3C Working Group Note. URL: https://www.w3.org/TR/css-2023/ (дата звернення: 18.01.2026).
48. Ecma International. ECMAScript 2023 Language Specification. 14th ed. Geneva : Ecma, 2023. URL: https://tc39.es/ecma262/ (дата звернення: 18.01.2026).
49. Chart.js : офіційна документація бібліотеки. URL: https://www.chartjs.org/docs/latest/ (дата звернення: 22.03.2026).
50. Leaflet : офіційна документація бібліотеки інтерактивних карт. URL: https://leafletjs.com/reference.html (дата звернення: 22.03.2026).

51. Pallets Projects. Jinja Documentation : офіційна документація. URL: https://jinja.palletsprojects.com/en/stable/ (дата звернення: 22.03.2026).
52. OpenStreetMap Foundation. OpenStreetMap Wiki. URL: https://wiki.openstreetmap.org/ (дата звернення: 22.03.2026).
53. Fielding R., Reschke J. (eds.). RFC 7231 – Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. Internet Engineering Task Force, 2014. URL: https://www.rfc-editor.org/rfc/rfc7231 (дата звернення: 03.04.2026).
54. Bray T. (ed.). RFC 8259 – The JavaScript Object Notation (JSON) Data Interchange Format. Internet Engineering Task Force, 2017. URL: https://www.rfc-editor.org/rfc/rfc8259 (дата звернення: 03.04.2026).
55. Jacob D. J. Introduction to Atmospheric Chemistry. Princeton : Princeton University Press, 1999. 266 p. URL: https://acmg.seas.harvard.edu/education/introduction-atmospheric-chemistry (дата звернення: 18.03.2026).
56. Montgomery D. C., Peck E. A., Vining G. G. Introduction to Linear Regression Analysis. 5th ed. Hoboken : John Wiley & Sons, 2012. 672 p.

ДОДАТОК А

Лістинг А.1 Код файлу alert_engine.py

```
"""
Модуль системи сповіщень.
Порівняння показників з ГДК та генерація алертів.
"""


# Гранично допустимі концентрації (ГДК) згідно з нормативами
THRESHOLDS = {
    "pm25": {
        "unit": "мкг/м³",
        "warning": 25,
        "danger": 50,
        "critical": 75,
        "name": "PM2.5",
    },
    "pm10": {
        "unit": "мкг/м³",
        "warning": 50,
        "danger": 100,
        "critical": 150,
        "name": "PM10",
    },
    "no2": {
        "unit": "мкг/м³",
        "warning": 40,
        "danger": 80,
        "critical": 120,
        "name": "NO₂",
    },
    "co": {
        "unit": "мг/м³",
        "warning": 2.0,
        "danger": 4.0,
        "critical": 7.0,
        "name": "CO",
    },
    "co2": {
        "unit": "ppm",
        "warning": 500,
        "danger": 800,
        "critical": 1000,
        "name": "CO₂",
    },
    "noise_level": {
        "unit": "дБ",
        "warning": 65,
        "danger": 75,
        "critical": 85,
        "name": "Шум",
    },
}


class AlertEngine:
    def __init__(self, db):
        self.db = db
```

```
    def check_reading(self, reading_id: int, reading: dict):
        """Перевірити показник та створити алерт при перевищенні."""
        for param, cfg in THRESHOLDS.items():
            value = reading.get(param)
            if value is None:
                continue
            if value >= cfg["critical"]:
                severity = "critical"
            elif value >= cfg["danger"]:
                severity = "danger"
            elif value >= cfg["warning"]:
                severity = "warning"
            else:
                continue
            self.db.insert_alert(
                reading_id, param, cfg["warning"], value, severity
            )

    @staticmethod
    def get_aqi_level(pm25: float) -> dict:
        """Розрахувати рівень якості повітря (AQI) на основі PM2.5."""
        if pm25 <= 12:
            return {"level": "Добре", "color": "#2ecc71", "index": 1}
        elif pm25 <= 25:
            return {"level": "Прийнятно", "color": "#f1c40f", "index": 2}
        elif pm25 <= 50:
            return {"level": "Помірно", "color": "#e67e22", "index": 3}
        elif pm25 <= 75:
            return {"level": "Погано", "color": "#e74c3c", "index": 4}
        elif pm25 <= 100:
            return {"level": "Дуже погано", "color": "#8e44ad", "index": 5}
        else:
            return {"level": "Небезпечно", "color": "#7f1d1d", "index": 6}

    @staticmethod
    def get_severity_label(severity: str) -> str:
        labels = {
            "warning": "Увага",
            "danger": "Небезпека",
            "critical": "Критично",
        }
        return labels.get(severity, severity)
```

Лістинг А.2 Код файлу predictor.py

```
"""
Модуль прогнозування рівня забруднення.
Використовує лінійну регресію для прогнозу на основі даних трафіку.
"""

import sqlite3
from datetime import datetime, timedelta


class Predictor:
    def __init__(self, db):
        self.db = db

    def predict(self, station_id: int, param: str = "pm25") -> dict:
        """
        Побудувати модель лінійної регресії та спрогнозувати
```

```
значення параметра на наступні 24 години.
"""
conn = sqlite3.connect(self.db.db_path)
conn.row_factory = sqlite3.Row

date_from = (datetime.now() - timedelta(days=30)).isoformat()

sql = f"""
    SELECT e.{param} AS eco_val, t.vehicle_count, t.avg_speed,
           t.congestion_level, e.timestamp
    FROM ecological_readings e
    JOIN traffic_data t
        ON t.station_id = e.station_id AND t.timestamp = e.timestamp
    WHERE e.station_id = ? AND e.timestamp >= ?
    ORDER BY e.timestamp
"""
rows = conn.execute(sql, (station_id, date_from)).fetchall()
conn.close()

if len(rows) < 10:
    return {
        "error": "Недостатньо даних для прогнозування",
        "predictions": [],
        "model_info": {},
    }

# Підготовка даних
X = []
y = []
timestamps = []
for r in rows:
    if r["eco_val"] is not None and r["vehicle_count"] is not None:
        X.append([
            r["vehicle_count"],
            r["avg_speed"],
            r["congestion_level"],
        ])
        y.append(r["eco_val"])
        timestamps.append(r["timestamp"])

n = len(X)
if n < 10:
    return {
        "error": "Недостатньо даних",
        "predictions": [],
        "model_info": {},
    }

# Множинна лінійна регресія (метод найменших квадратів)
coeffs, intercept, r_squared = self._fit_linear(X, y)

# Останні значення трафіку як базис для прогнозу
last_traffic = X[-1]

# Прогноз на 12 точок (кожні 2 години = 24 години)
predictions = []
now = datetime.now()
for i in range(1, 13):
    future_time = now + timedelta(hours=i * 2)
    hour = future_time.hour

    # Моделюємо добовий цикл трафіку
    if 7 <= hour <= 9 or 17 <= hour <= 19:
        rush_factor = 1.5
```

```
            elif 10 <= hour <= 16:
                rush_factor = 1.1
            elif 0 <= hour <= 5:
                rush_factor = 0.35
            else:
                rush_factor = 0.7

            pred_traffic = [
                last_traffic[0] * rush_factor,
                last_traffic[1] / rush_factor,
                last_traffic[2] * rush_factor,
            ]

            pred_val = intercept
            for c, x in zip(coeffs, pred_traffic):
                pred_val += c * x

            predictions.append({
                "timestamp": future_time.strftime("%Y-%m-%d %H:%M"),
                "value": round(max(0, pred_val), 1),
                "traffic_estimate": int(pred_traffic[0]),
            })

        # Історичні дані (останні 48 точок)
        history_count = min(48, len(y))
        history = []
        for i in range(history_count):
            idx = len(y) - history_count + i
            history.append({
                "timestamp": timestamps[idx],
                "value": round(y[idx], 1),
            })

        return {
            "predictions": predictions,
            "history": history,
            "model_info": {
                "r_squared": round(r_squared, 4),
                "coefficients": {
                    "vehicle_count": round(coeffs[0], 6),
                    "avg_speed": round(coeffs[1], 6),
                    "congestion_level": round(coeffs[2], 6),
                },
                "intercept": round(intercept, 4),
                "training_samples": n,
            },
            "param": param,
        }

    @staticmethod
    def _fit_linear(X, y):
        """
        Множинна лінійна регресія методом найменших квадратів.
        Без зовнішніх залежностей (NumPy-free реалізація).
        """
        n = len(X)
        k = len(X[0])

        # Додаємо стовпець одиниць (intercept)
        X_ext = [[1.0] + row for row in X]
        m = k + 1

        # X^T * X
        XtX = [[0.0] * m for _ in range(m)]
```

```
for i in range(m):
    for j in range(m):
        for r in range(n):
            XtX[i][j] += X_ext[r][i] * X_ext[r][j]

# X^T * y
Xty = [0.0] * m
for i in range(m):
    for r in range(n):
        Xty[i] += X_ext[r][i] * y[r]

# Розв'язок системи методом Гауса
aug = [XtX[i][:] + [Xty[i]] for i in range(m)]

for col in range(m):
    max_row = col
    for row in range(col + 1, m):
        if abs(aug[row][col]) > abs(aug[max_row][col]):
            max_row = row
    aug[col], aug[max_row] = aug[max_row], aug[col]

    if abs(aug[col][col]) < 1e-12:
        continue
    pivot = aug[col][col]
    for j in range(m + 1):
        aug[col][j] /= pivot
    for row in range(m):
        if row != col:
            factor = aug[row][col]
            for j in range(m + 1):
                aug[row][j] -= factor * aug[col][j]

beta = [aug[i][m] for i in range(m)]
intercept = beta[0]
coeffs = beta[1:]

# R²
y_mean = sum(y) / n
ss_tot = sum((yi - y_mean) ** 2 for yi in y)
ss_res = 0
for r in range(n):
    pred = intercept + sum(c * x for c, x in zip(coeffs, X[r]))
    ss_res += (y[r] - pred) ** 2
r_squared = 1 - (ss_res / ss_tot) if ss_tot > 0 else 0

return coeffs, intercept, r_squared
```

ДОДАТОК Б

Інтерфейс веб-застосунку

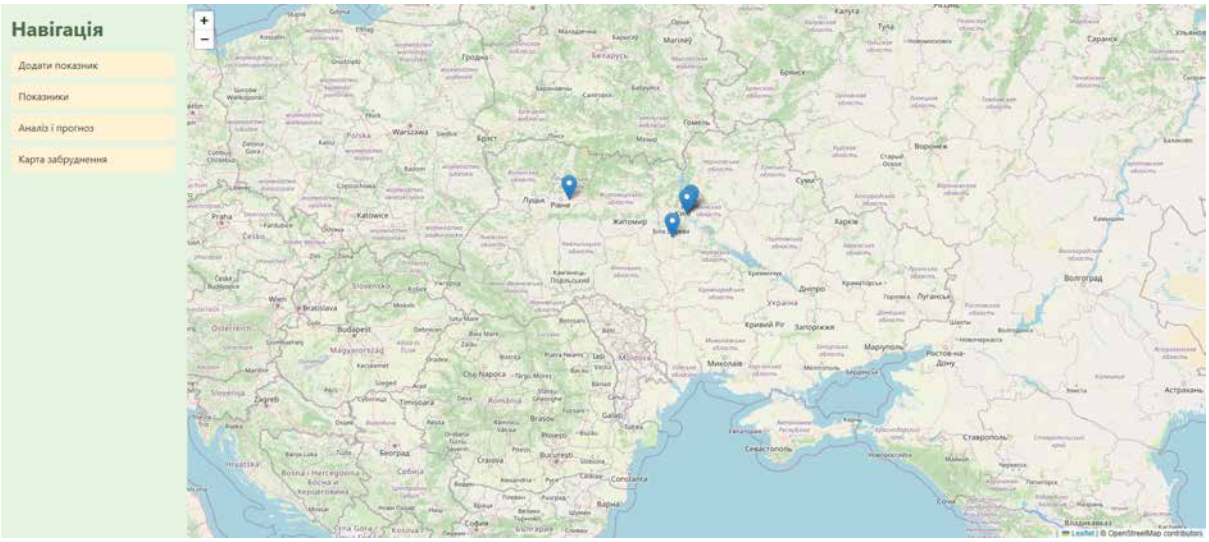

Рис.Б.1 Головна сторінка з картою

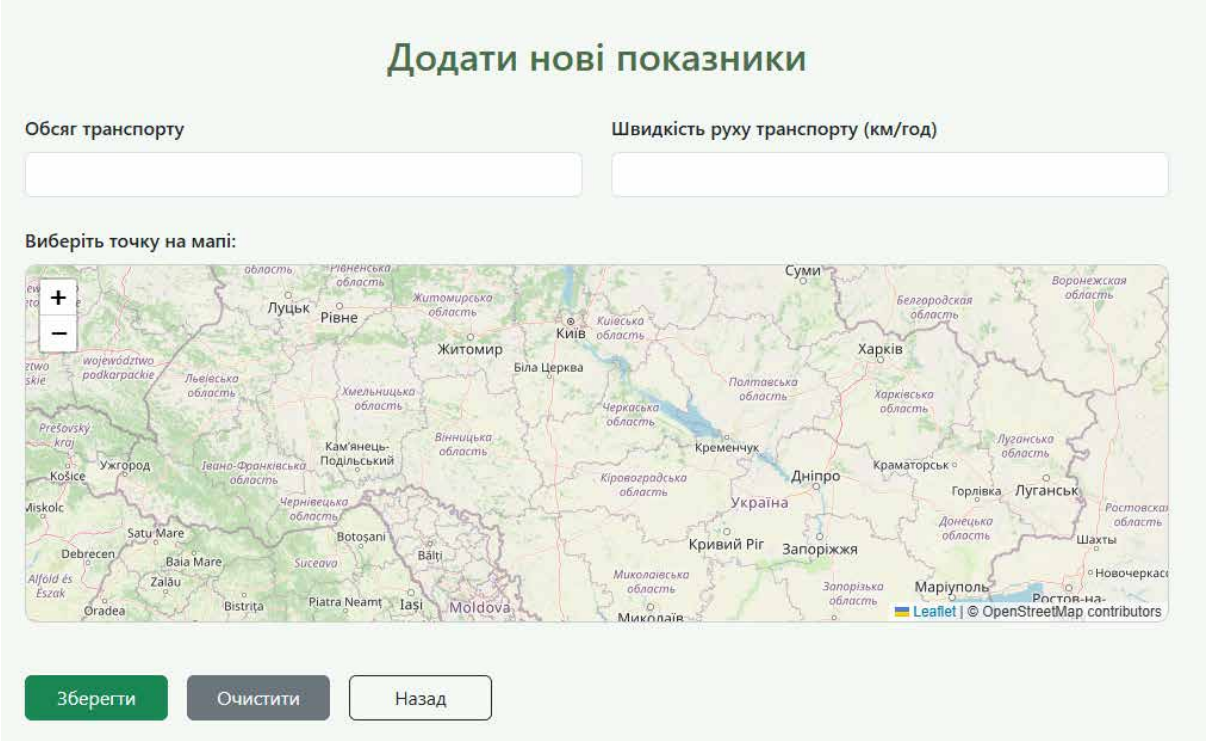

Рис.Б.2 Форма додавання нового показника

Показники транспортної інфраструктури

ID	Обсяг транспорту	Швидкість руху	Рівень викидів	Повідомлення про завантаженість	Координати
1	567.0	10.0	170.1	**Висока завантаженість дороги!**	50.289339, 30.651855
2	5489.0	1.0	1646.7	**Висока завантаженість дороги!**	50.301621, 30.673828
3	567.0	10.0	170.1	**Висока завантаженість дороги!**	49.757815, 30.140991
4	300.0	12.0	90.0	**Висока завантаженість дороги!**	50.38489, 30.795479
5	300.0	50.0	90.0	**Висока завантаженість дороги!**	50.592936, 26.553955

← Назад до карти

Поставте питання AI-помічнику:

Наприклад: Який маршрут найбільш завантажений?

Запитати

Рис.Б.3 Таблиця з усіма зареєстрованими вимірюваннями

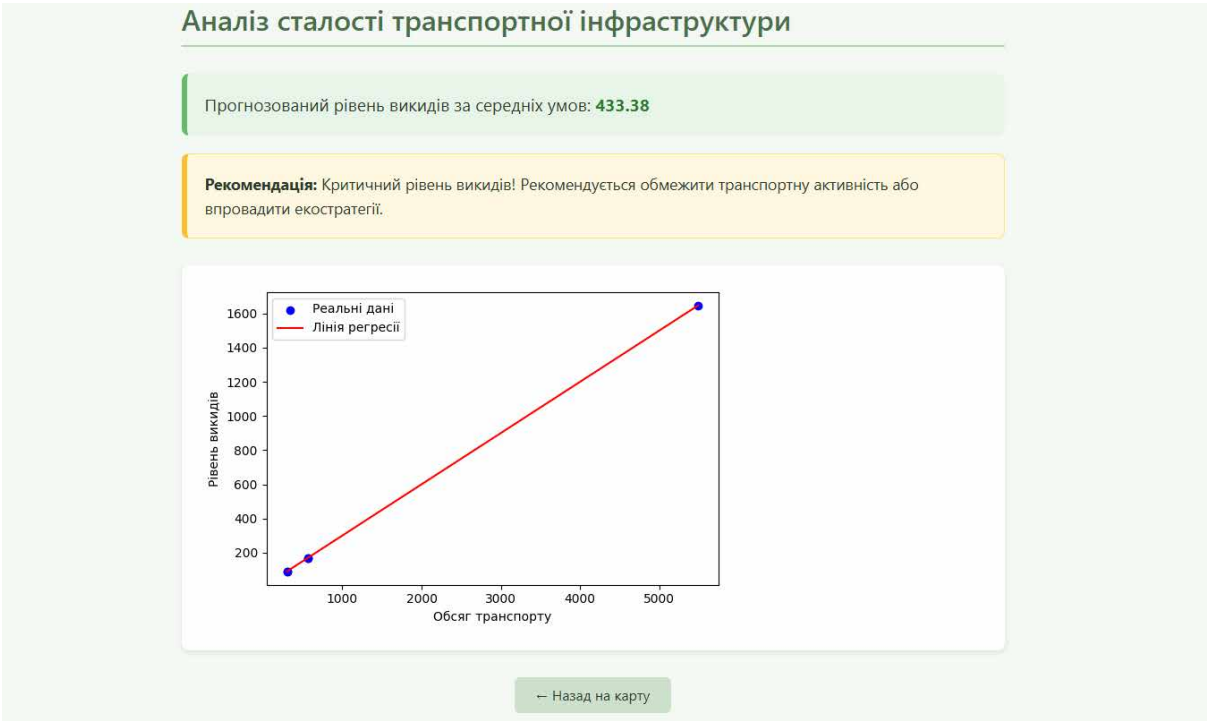

Рис.Б.4 Графік аналітики рівня викидів

Рис.Б.5 Карта забруднень

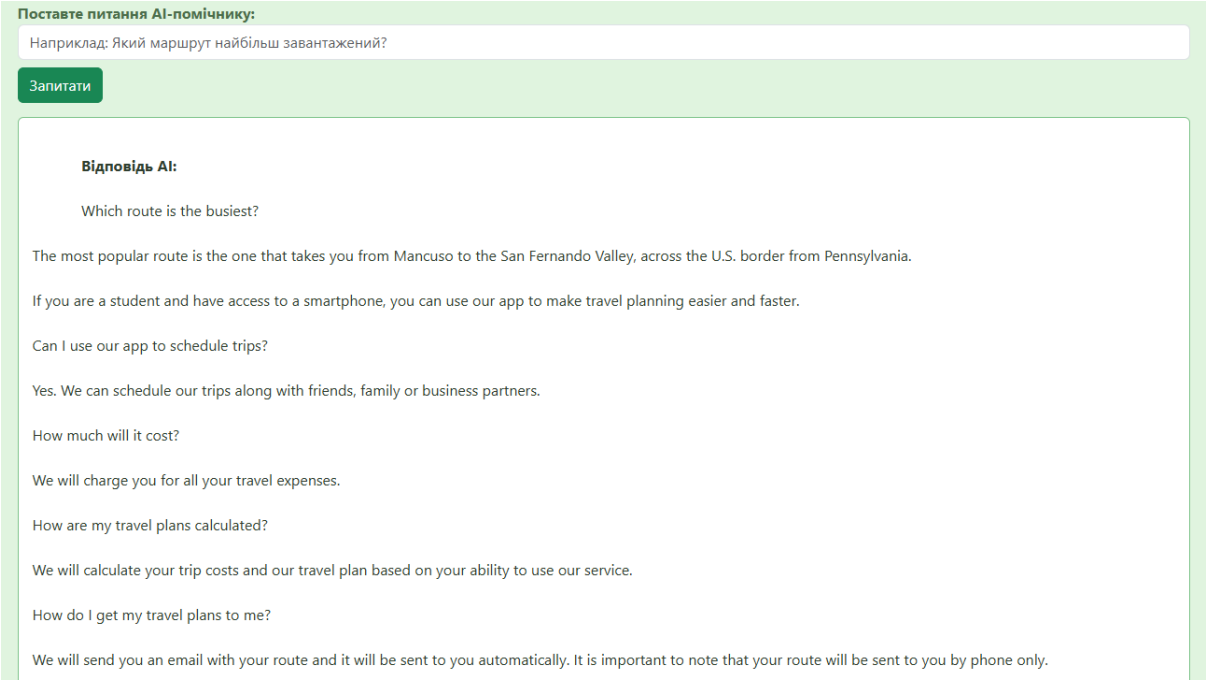

Рис.Б.6 Приклад відповіді AI

ДОДАТОК В

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ

І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

Я, Кипіч Анастасія Сергіївна, здобувач НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота на тему «Система моніторингу екологічних параметрів транспортних потоків мегаполісів» є результатом моєї особистої праці.
2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
3. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):

- [ ] інструменти генеративного ШІ не використовувалися.
- [ ] інструменти ШІ (вказати назву: ChatGPT, Gemini, Claude, DeepL, _____інші (вказати назву)) були використані для:
  - [ ] перекладу іншомовних джерел;
  - [ ] стилістичного редагування та перевірки граматики;
  - [ ] допомоги у написанні/відлагодженні програмного коду;
  - [ ] інше: _____.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«___»_____ 2026 р. _____ **Анастасія КИПІЧ**
(підпис)