НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
_інформаційних систем і технологій__
(назва кафедри)

_____ Швиденко М.З, к.е.н., проф.
(підпис) (ПІБ)

“__” _____ 2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему

_____Інформаційна система з моніторингу повітря _____

Спеціальність 126 – «Інформаційні системи та технології»

Гарант освітньої програми
_____Канд.екон.наук, доцент_____ _____ Мокрієв Максим Володимирович
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи
__Кандидат економічних наук, доцент__ _____ Швиденка М.З.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

__Асистент_____ _____ _Понзель Я.Ю.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ _____Слободенюк М.І._____
(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ

І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
інформаційних систем і технологій
(назва кафедри)

_____ _____
Швиденко М.З к.е.н., проф
(вчене звання і ступінь) (підпис) (ініціали і прізвище)

«__»_____ 2025р.

З А В Д А Н Н Я

на виконання бакалаврської кваліфікаційної роботи студенту

Слободенюку Максиму Ігоровичу
прізвище, ім'я, по батькові

Спеціальність 126 – «Інформаційні системи та технології»

Тема бакалаврської кваліфікаційної роботи
Інформаційна система з моніторингу повітря
затверджена наказом ректора НУБіП від " 16"12 2024р. № 12-С
Термін подання завершеної роботи на кафедру 2025.06.01
рік, місяць, число

Вихідні дані до бакалаврської кваліфікаційної роботи: опис роботи інформаційной системи "Медична картка у смартфоні"

Перелік питань, що розглядаються:

1. Огляд та аналіз предметної області, а саме інформаційних систем моніторингу якості повітря
2. Інформаційне забезпечення
3. Прикладне програмне забезпечення
4. Тестування та вимоги системи
5. Висновки

Дата видачі завдання "08" січня 2025 р.

Календарний план

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної області, постановка мети й завдань дослідження	01.01.2025 – 10.01.2025	
2	Обґрунтування та вибір архітектурного підходу і технологічного стеку	11.01.2025 – 20.01.2025	
3	Проєктування структури бази даних: логічна модель, ER-діаграма, 3НФ	21.01.2025 – 31.01.2025	
4	Розробка серверної частини: API, маршрутизація, авторизація, захист	01.02.2025 – 15.02.2025	
5	Розробка клієнтської частини: мапа станцій, графік забруднення, звіти	16.02.2025 – 29.02.2025	
6	Інтеграція з MySQL та реалізація ключового функціоналу: станції моніторингу, результати спостереження	01.03.2025 – 10.03.2025	
7	Додаткові модулі: реєстрація станцій моніторингу, звіти в CSV та PDF	11.03.2025 – 20.03.2025	
8	Тестування функціональності системи відповідно до реальних сценаріїв користувача	21.03.2025 – 31.03.2025	
9	Написання та редагування пояснювальної записки	01.04.2025 – 25.04.2025	
10	Консультації з керівником, доопрацювання, оформлення додатків та презентації	26.04.2025 – 01.06.2025	
11	Завершальні перевірки: антиплагіат, предзахист, основний захист	02.06.2025 – 16.06.2025	

Керівник дипломного проекту Швиденко М.З

Асистент Понзель Я. Ю.

(підпис) (прізвище та ініціали)

Завдання прийняв до виконання _____ Слободенюк М.І.

(підпис) (прізвище та ініціали студента)

РЕФЕРАТ

Тема бакалаврської кваліфікаційної роботи: “ Інформаційна система з моніторингу повітря ”.

Автор роботи: Слободенюк Максим І.

Керівник роботи: Швиденко М.З

Асиситент: Понзель Я.Ю.

Пояснювальна записка: 70 с., 18 рис., 21 табл., 1 дод., 29 джерел.

Графічна частина: 10 презентаційних слайдів.

АДАПТИВНИЙ ДИЗАЙН, БАЗА ДАНИХ, ВЕБ-ІНТЕРФЕЙС, ВІЗУАЛІЗАЦІЯ ДАНИХ, ЕКОЛОГІЧНИЙ МЕНЕДЖМЕНТ, ІНТЕРАКТИВНА КАРТА, МОНІТОРИНГ ПОВІТРЯ, СИСТЕМНИЙ АНАЛІЗ.

Метою роботи є розробка інформаційної системи моніторингу якості повітря, що забезпечує оперативний збір, обробку, аналіз і візуалізацію даних про забруднення для інформування населення та підтримки екологічного менеджменту. У роботі створено інформаційну систему AirGuard, яка реалізує моніторинг забруднювачів у реальному часі. Розроблено архітектуру системи, включаючи логічну та фізичну моделі бази даних у MySQL, а також людино-машинний інтерфейс із інтерактивною картою (Leaflet.js) і графіками (Chart.js). Система реалізована за допомогою PHP, MySQL, HTML5, CSS3, JavaScript, забезпечує адаптивний дизайн, автоматичні сповіщення та генерацію звітів у форматах CSV і PDF. Розроблена система AirGuard відповідає вимогам, забезпечує ефективний моніторинг і має потенціал для масштабування. Пропозиції до розвитку включають інтеграцію з мобільними додатками для push-сповіщень і застосування машинного навчання для прогнозування якості повітря.

ABSTRACT

Title of the bachelor's qualification work: "Information System for Air Quality Monitoring".

Author: Slobodeniuk M.I.

Supervisor: Shvidenko M.Z.

Assistant Ponzel Y.Y.

Explanatory note: 70 pages, 18 figures, 21 tables, 1 appendices, 29 sources.

Graphical part: 10 presentation slides.

ADAPTIVE DESIGN, DATABASE, ECOLOGICAL MANAGEMENT, INTERACTIVE MAP, AIR MONITORING, SYSTEM ANALYSIS, VISUALIZATION, WEB INTERFACE.

The purpose of this work is to develop an air quality monitoring information system that enables real-time collection, processing, analysis, and visualization of pollution data to inform the public and support environmental management. The study presents AirGuard, an information system designed for real-time pollutant monitoring. The system's architecture was developed, including logical and physical database models in MySQL, as well as a human-machine interface featuring an interactive map (Leaflet.js) and data charts (Chart.js). The system was implemented using PHP, MySQL, HTML5, CSS3, and JavaScript, ensuring adaptive design, automated alerts, and report generation in CSV and PDF formats. AirGuard meets all functional requirements, provides efficient monitoring, and has scalability potential. Future development proposals include integration with mobile apps for push notifications and application of machine learning for air quality forecasting

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AJAX - Asynchronous JavaScript and XML (асинхронний обмін даними між клієнтом і сервером)

API - Application Programming Interface (інтерфейс програмного забезпечення)

AQI - Air Quality Index (індекс якості повітря)

CSS - Cascading Style Sheets (каскадні таблиці стилів)

HTML - HyperText Markup Language (мова розмітки гіпертексту)

HTTP - HyperText Transfer Protocol (протокол передачі гіпертексту)

JSON - JavaScript Object Notation (формат обміну даними)

PHP - мова програмування для веб-розробки

SQL - Structured Query Language (мова структурованих запитів)

WCAG - Web Content Accessibility Guidelines (правила доступності веб-контенту)

ЛМІ - людино-машинний інтерфейс

ПЗ - програмне забезпечення

СУБД - система управління базами даних

ТЗ - технічне завдання

ВСТУП

У сучасних умовах зростання екологічних загроз забруднення повітря є однією з ключових проблем, що впливають на здоров’я населення та стан довкілля. За даними Всесвітньої організації охорони здоров’я, 90% людей дихають повітрям із високим рівнем забруднювачів, що спричиняє серйозні захворювання та екологічні ризики [1]. В Україні, особливо в містах із розвиненою промисловістю, таких як Київ, проблема якості повітря стоїть особливо гостро через викиди транспорту, заводів та побутові джерела. Розробка інформаційної системи моніторингу якості повітря забезпечує оперативний доступ до даних про забруднення, спрощує інформування населення та сприяє прийняттю обґрунтованих рішень для зниження екологічних ризиків.

Створення такої системи надасть швидкий, надійний та зручний доступ до інформації про стан повітря, забезпечить громадян, екологів і органи влади актуальними даними через веб-інтерфейс, спростить аналіз забруднення та підвищить екологічну свідомість населення [2]. При роботі з наявними системами моніторингу користувачі стикаються з низкою проблем, зокрема:

– обмежена доступність даних: відсутність оперативного доступу до інформації про якість повітря в реальному часі;

– складність інтеграції: несумісність різних типів датчиків та програмного забезпечення ускладнює обробку даних;

– недостатня візуалізація: брак зручних інтерфейсів для представлення даних у вигляді карт чи графіків;

– низька оперативність сповіщень: затримки в інформуванні про критичні рівні забруднення.

Тому **метою дослідження** є обґрунтування доцільності та розробка інформаційної системи моніторингу якості повітря, що забезпечить оперативний збір, обробку, зберігання та візуалізацію даних про забруднювачі.

Об’єктом дослідження є процеси збору, обробки та представлення даних про якість атмосферного повітря в інформаційних системах.

Предметом дослідження є технологічні та програмні засоби для створення інформаційної системи моніторингу якості повітря, включаючи аналіз предметної області, проєктування бази даних, розробку веб-інтерфейсу, тестування системи та оцінку її ефективності.

Завданням дослідження є розробка веб-системи, яка забезпечить доступ до даних про якість повітря з інтерактивною картою, графіками та сповіщеннями, а також підтримуватиме аналіз трендів і генерацію звітів.

Пояснювальна записка складається з трьох основних розділів, що відображають етапи виконання дослідження та розробки системи. Вступна частина включає обґрунтування актуальності теми, визначення мети, завдань, об’єкта і предмета дослідження, а також структуру роботи. Перший розділ присвячений аналізу предметної області, огляду існуючих рішень (AirVisual, PurpleAir, OpenAQ, EEA, SaveDnipro), постановці вимог і розробці технічного завдання. Другий розділ описує проєктування системи, включаючи вибір технологічного стеку (PHP, MySQL, JavaScript), розробку логічної та фізичної моделей бази даних, а також людино-машинного інтерфейсу. Третій розділ охоплює реалізацію системи AirGuard, опис модулів, інформаційних потоків, технічних характеристик, а також результати тестування.

У висновках наведено основні результати, оцінено їхню відповідність вимогам, надано рекомендації щодо впровадження та подальшого розвитку системи. Список використаних джерел містить наукові та технічні матеріали, а додатки включають діаграми, прототипи інтерфейсу та інші допоміжні документи.

Додатки містять додаткові матеріали, такі як лістинг програмних модулів та документи, що підтверджують впровадження системи.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ СИСТЕМ МОНІТОРИНГУ ЯКОСТІ ПОВІТРЯ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовний аналіз предметної області, її структурних та функціональних особливостей

Моніторинг якості повітря є актуальною задачею в сучасному світі через зростаючі екологічні проблеми, пов'язані з забрудненням атмосфери. Основними джерелами забруднення є промислові викиди, транспорт, побутові відходи та природні явища, такі як лісові пожежі чи пилові бурі. Згідно з даними Всесвітньої організації охорони здоров'я (ВООЗ), 9 із 10 людей у світі дихають повітрям, що містить високий рівень забруднювачів, що призводить до значних ризиків для здоров'я, включаючи респіраторні захворювання, серцево-судинні проблеми та онкологічні захворювання [1].

Основні забруднювачі повітря включають тверді частинки (PM2.5, PM10), діоксид азоту (NO_2), діоксид сірки (SO_2), озон (O_3) та вуглекислий газ (CO_2) [3]. Для оцінки стану повітря використовуються різні методи, зокрема стаціонарні станції моніторингу, мобільні датчики та супутникові дані. Проблематика полягає в тому, що наявні системи часто мають обмежену територіальну охопленість, високу вартість або недостатню оперативність у наданні даних. Вплив на здоров'я людини основних забруднювачів повітря проаналізований у табл. 1.1. Таким чином, моніторинг якості повітря є критично важливим для своєчасного виявлення небезпечних рівнів забруднення, інформування населення та розробки заходів для зменшення екологічних ризиків.

Рис. 1.1 ілюструє типову структуру системи моніторингу повітря. Системи моніторингу повітря зазвичай складаються з кількох ключових компонентів, які забезпечують збір, обробку та відображення даних.

Таблиця 1.1 – Основні забруднювачі повітря та їхній вплив [3]

Забруднювач	Джерело походження	Вплив на здоров'я

PM2.5, PM10	Промисловість, транспорт	Респіраторні захворювання
NO_2	Автомобільні вихлопи	Подразнення дихальних шляхів
SO_2	Спалювання вугілля	Астма, бронхіт
O_3	Фотохімічні реакції	Зниження функції легень
CO_2	Викиди транспорту, заводи	Кліматичні зміни, опосередкований вплив

Основна структура включає:

– датчики та вимірювальні пристрої**:** відповідають за збір даних про концентрацію забруднювачів у реальному часі.

– система передачі даних**:** забезпечує транспортування зібраної інформації до центрального сервера (наприклад, через GSM, Wi-Fi або LoRaWAN).

– база даних**:** **з**берігає історичні дані та дозволяє проводити аналіз трендів.

– програмне забезпечення**:** обробляє дані та представляє їх у зрозумілій формі для користувачів (графіки, карти, таблиці).

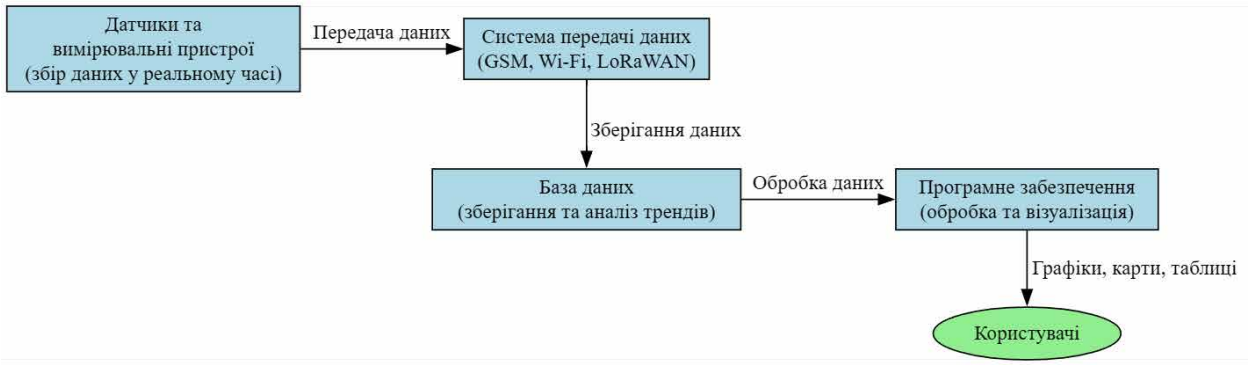

Рисунок 1.1 – Схема структури системи моніторингу повітря

Структурні особливості також залежать від масштабів системи. Наприклад, локальні системи можуть охоплювати одне місто чи район, тоді як глобальні мережі, такі як Air Quality Index (AQI) [5], інтегрують дані з різних країн. Проблема полягає в необхідності забезпечення сумісності між різними типами датчиків та програмного забезпечення, а також у масштабуванні системи для обробки великих обсягів даних.

Інформаційні системи для моніторингу повітря повинні відповідати низці функціональних вимог, щоб ефективно виконувати свої завдання. Основні з них наведено нижче:

1. Збір даних у реальному часі: система має забезпечувати оперативне отримання інформації з датчиків.

2. Обробка та аналіз даних: необхідна можливість обчислення середніх значень, виявлення пікових концентрацій забруднювачів та прогнозування трендів.

3. Візуалізація: відображення даних у зрозумілій формі (карти забруднення, графіки, таблиці).

4. Доступність: користувачі повинні мати доступ до системи через веб-інтерфейс із будь-якого пристрою.

5. Сповіщення: автоматичне інформування про перевищення допустимих норм забруднення.

Табл. 1.2 підсумовує функціональні вимоги до системи [6].

Таблиця 1.2 – Функціональні вимоги до ІС моніторингу повітря

Вимога	Опис
Збір даних	Отримання інформації з датчиків у реальному часі
Аналіз даних	Обчислення середніх значень, прогнозування
Візуалізація	Карти, графіки, таблиці
Доступність	Веб-інтерфейс для різних пристроїв
Сповіщення	Повідомлення про критичні рівні

Таким чином, інформаційна система повинна бути гнучкою, масштабованою та зручною для користувачів, враховуючи специфіку предметної області та потреби кінцевих споживачів даних.

1.2 Аналіз наявних інформаційних технологій предметної області

На сьогодні існує низка інформаційних систем, призначених для моніторингу якості повітря, які застосовуються як на локальному, так і на глобальному рівнях. Серед найвідоміших прикладів можна виділити AirVisual [9], PurpleAir [7], OpenAQ [8] та офіційні системи моніторингу, такі як Європейська агенція з навколишнього середовища (EEA) [2]. Ці системи мають різні підходи до збору, обробки та представлення даних, що дозволяє проаналізувати їхні особливості для подальшого використання в розробці власного рішення.

AirVisual – це комерційна платформа, яка використовує мережу датчиків від користувачів та інтегрує дані з державних станцій. Вона надає доступ до інформації через веб-сайт та мобільний додаток, відображаючи індекс якості повітря (AQI) у реальному часі [4]. PurpleAir, у свою чергу, орієнтована на спільноту: користувачі встановлюють власні датчики, а дані агрегуються на інтерактивній карті [5]. OpenAQ – це відкрита платформа, яка збирає дані з державних джерел і робить їх доступними для дослідників та розробників через API [6]. EEA забезпечує моніторинг на рівні Європейського Союзу, використовуючи стаціонарні станції та уніфіковані стандарти. Рис. 1.2 ілюструє приклад інтерфейсу однієї з таких систем – карти забруднення повітря від PurpleAir [7].

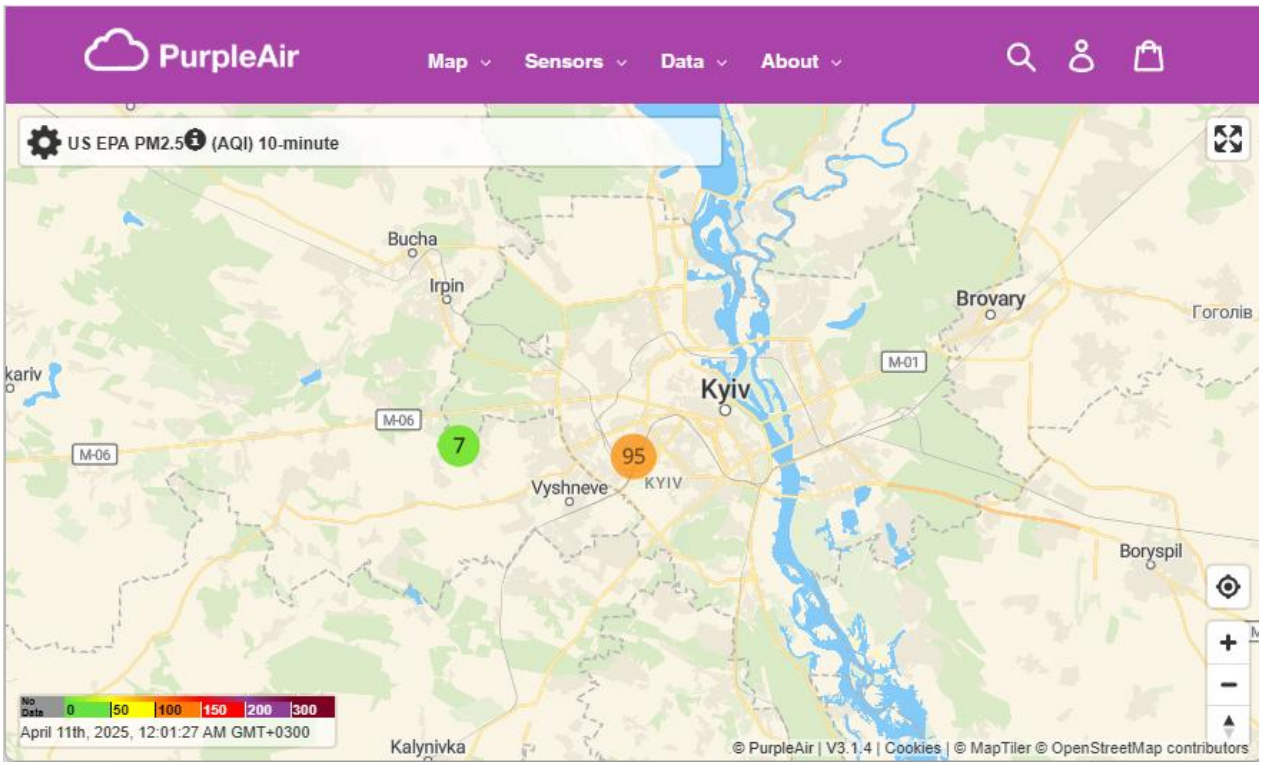

Рисунок 1.2 – Інтерактивна карта PurpleAir [7]

Кожна з цих систем має свої сильні сторони, однак їхнє функціонування залежить від типу використовуваних датчиків, методів передачі даних та цільової аудиторії. Для оцінки існуючих систем проведено порівняльний аналіз за ключовими критеріями: функціональність, людино-машинний інтерфейс, переваги та недоліки. Основні характеристики систем наведено в табл. 1.3.

Таблиця 1.3 – Порівняльний аналіз систем моніторингу повітря

Система	Функціональність	Інтерфейс	Переваги	Недоліки
AirVisual	AQI, прогнози, інтеграція даних	Веб, мобільний додаток	Зручність, глобальне покриття	Обмежений доступ до сирих даних
PurpleAir	Дані з датчиків у реальному часі	Інтерактивна веб-карта	Відкритість, низька вартість	Неточність дешевих датчиків
OpenAQ	Агрегація відкритих даних, API	Веб-інтерфейс, API	Безкоштовність, підтримка науки	Відсутність власних датчиків
EEA	Офіційні дані, стандартизований аналіз	Веб-сайт, звіти	Висока точність, надійність	Обмежена оперативність

AirVisual вирізняється широким функціоналом, включаючи прогнози погоди та якості повітря, а також інтуїтивно зрозумілим інтерфейсом із візуалізацією на карті та у вигляді графіків. PurpleAir пропонує просту веб-карту, де кожен датчик позначений кольором залежно від рівня забруднення, але точність даних може варіюватися через використання недорогих сенсорів. OpenAQ зосереджена на агрегації даних і не має розвиненого інтерфейсу для кінцевих користувачів, зате її API корисний для розробників. EEA забезпечує високу достовірність, але доступ до даних ускладнений через бюрократичні процедури та затримки в оновленні.

Переваги цих систем включають різноманітність підходів до збору даних (від спільнот до державних мереж) та доступність для різних груп користувачів. Проте недоліки, такі як обмежена точність чи складність інтеграції, вказують на потребу в розробці гнучкішого рішення, яке б поєднувало простоту використання з надійністю даних.

Громадське об’єднання SaveDnipro є українською неурядовою організацією, яка активно працює над підвищенням екологічної свідомості та моніторингом стану довкілля. Одним із ключових напрямів діяльності SaveDnipro є розробка та впровадження систем моніторингу якості повітря, що дозволяє громадськості отримувати актуальну інформацію про стан атмосфери в реальному часі. На їхньому офіційному сайті представлено обладнання для моніторингу, зокрема датчики, які вимірюють концентрацію забруднювачів, таких як тверді частинки (PM2.5, PM10), а також радіаційний фон [5].

SaveDnipro співпрацює з міжнародними партнерами, такими як Greenpeace та Safecast, для створення незалежних мереж моніторингу, що особливо актуально в умовах екологічних загроз, наприклад, поблизу промислових зон чи ядерних об’єктів. Їхні датчики якості повітря відзначаються доступністю та простотою у використанні: вони підключаються до мережі через Wi-Fi і передають дані на відкриті платформи, такі як SaveEcoBot, де користувачі можуть переглядати показники в реальному часі.Рис. 1.3

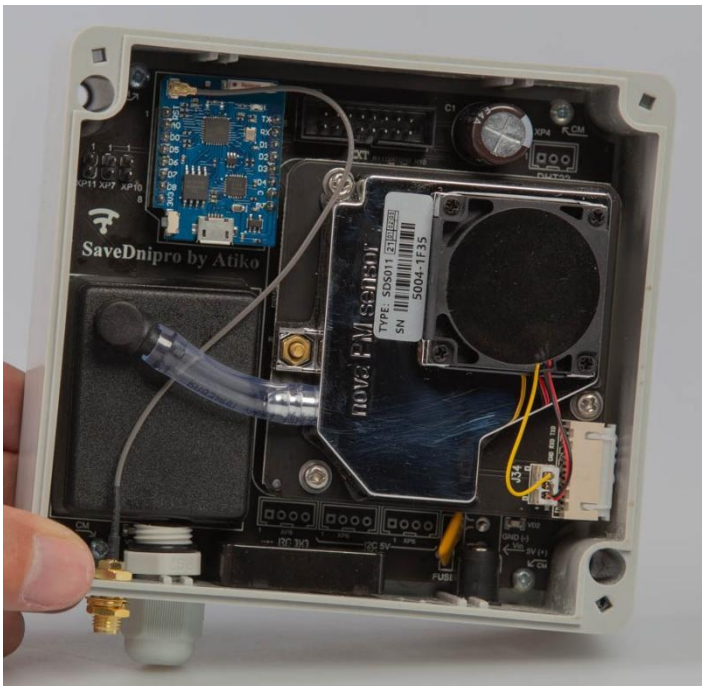

Рисунок 1.3 – Датчик моніторингу якості повітря від SaveDnipro [5]

Перевагою підходу SaveDnipro є орієнтація на громадянську науку: організація заохочує звичайних людей долучатися до моніторингу, надаючи доступні інструменти та відкриті дані. Це сприяє підвищенню прозорості в екологічних питаннях та дозволяє виявляти локальні джерела забруднення [10]. Водночас система має обмеження, пов’язані з необхідністю додаткової верифікації даних для офіційного використання, що вказує на потенціал для інтеграції з більш точними технологіями в майбутньому.Табл. 1.4

Таблиця 1.4 –Характеристики системи моніторингу SaveDnipro

Параметр	Опис
Вимірювані показники	PM2.5, PM10, радіація
Інтерфейс	Веб-платформа SaveEcoBot
Переваги	Доступність, підтримка спільноти
Недоліки	Обмежена точність без калібрування

Досвід SaveDnipro може бути корисним для розробки власної інформаційної системи, зокрема в контексті залучення громадськості та

створення простого у використанні веб-інтерфейсу на базі PHP і MySQL [11]. Їхній підхід до відкритості даних та модульності обладнання варто врахувати як приклад для адаптації в локальних проєктах моніторингу повітря.

Проведений аналіз показує, що сучасні системи моніторингу повітря мають значний потенціал для запозичення певних рішень. Наприклад, інтерактивна карта PurpleAir може слугувати зразком для створення зрозумілого веб-інтерфейсу, а API OpenAQ – прикладом для реалізації відкритої інтеграції даних. Водночас висока точність EEA підкреслює важливість стандартизації вимірювань, що слід врахувати при виборі датчиків.

Для розробки власної системи доцільно взяти за основу модель PurpleAir із її акцентом на доступність і візуалізацію, доповнивши її точнішими датчиками та можливістю прогнозування, як у AirVisual. Крім того, використання MySQL як бази даних може бути адаптовано з досвіду OpenAQ для ефективного зберігання та обробки великих обсягів інформації. Таким чином, досвід аналогів дозволяє сформувати основу для створення конкурентоспроможної системи, яка відповідатиме сучасним потребам користувачів.

1.3 Визначення вимог інформаційної системи, розробка технічного завдання

Проєктування інформаційної системи моніторингу якості повітря вимагає ретельного визначення функціональних та нефункціональних вимог, вибору відповідних інструментальних засобів та складання технічного завдання. Це дозволяє чітко окреслити обсяг робіт, визначити архітектуру системи, очікувану поведінку, обмеження і зовнішні інтерфейси.

Функціональні вимоги визначають ключові можливості інформаційної системи з моніторингу повітря, які забезпечують її практичну цінність [12]. На основі аналізу предметної області та аналогів сформульовано такі основні функції:

– збір даних із датчиків у реальному часі (PM2;5, PM10, NO_2 тощо);

– обробка даних, зокрема обчислення середніх значень і виявлення перевищень допустимих норм;

– візуалізація у вигляді графіків, таблиць та інтерактивної карти;

– надсилання сповіщень користувачам при критичних рівнях забруднення;

– забезпечення доступу до архіву даних для аналізу трендів.

Ці вимоги детально відображені в Use Case діаграмі, яка описує взаємодію між акторами та системою. На діаграмі представлено два основні актори: "Користувач" (громадяни, екологи) та "Адміністратор" (особа, що налаштовує систему). основні випадки використання включають:

– перегляд даних: користувач отримує доступ до поточних і архівних показників через веб-інтерфейс;

– отримання сповіщень: система автоматично надсилає повідомлення при перевищенні порогових значень;

– аналіз трендів: користувач переглядає історичні дані для оцінки змін якості повітря;

– налаштування датчиків: адміністратор додає нові датчики або змінює конфігурацію існуючих.

Між цими випадками використання існують зв’язки типу "include" (наприклад, "Аналіз трендів" включає "Перегляд даних") та "extend" (наприклад, "Отримання сповіщень" розширює "Перегляд даних" за умови критичних показників).Рис. 1.4

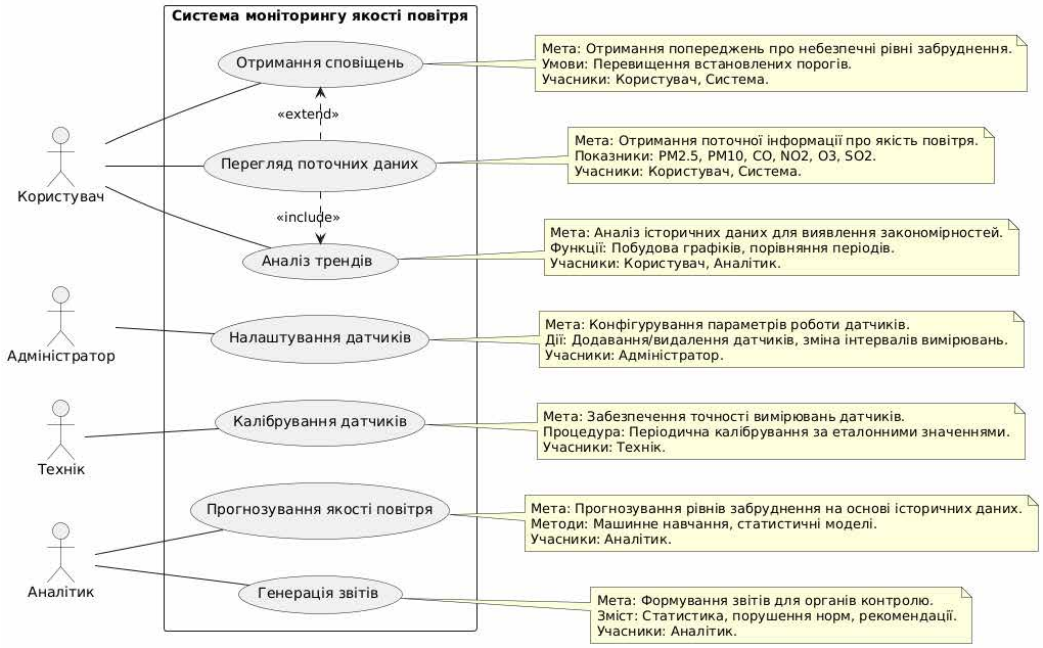

Рисунок 1.4 – Use Case діаграма інформаційної системи моніторингу повітря

Такий підхід дозволяє чітко визначити функціонал системи та його взаємодію з користувачами, що є основою для реалізації на PHP [13] та MySQL [14].

Нефункціональні вимоги описують якісні характеристики системи, необхідні для її ефективної роботи. Для системи моніторингу повітря встановлено наступні вимоги:

– доступність: веб-інтерфейс має бути доступним із будь-якого пристрою з браузером;

– продуктивність: час відповіді на запити – до 2 секунд при підключенні до 100 датчиків;

– масштабованість: система підтримує додавання нових датчиків без переписування коду;

– надійність: збереження даних у MySQL навіть при короткочасних збоях мережі;

– безпека: захист даних через авторизацію та шифрування з’єднань.

Ці вимоги узагальнені в табл. 1.5 для подальшого використання.

Таблиця 1.5 – Нефункціональні вимоги до системи

Вимога	Опис
Доступність	Веб-доступ із будь-якого пристрою
Продуктивність	Обробка запитів до 2 секунд
Масштабованість	Підтримка розширення кількості датчиків
Надійність	Збереження даних при збоях
Безпека	Авторизація та шифрування

Нефункціональні вимоги гарантують стабільність і зручність системи, що є важливим для її практичного застосування.

Технічне завдання (ТЗ) формалізує вимоги та окреслює план розробки системи. Основні пункти ТЗ:

1. Мета: створення веб-системи для моніторингу якості повітря з інтеграцією датчиків, обробкою даних і візуалізацією.

2. Технології: PHP для серверної логіки, MySQL для бази даних, HTML/CSS/JavaScript для інтерфейсу.

3. Функціонал:

– збір даних через API або пряме підключення датчиків;

– збереження та архівація в MySQL;

– візуалізація через карту та графіки;

– сповіщення при перевищенні норм;

4. Інтерфейс: адаптивний веб-дизайн із простою навігацією.

5. Терміни: 3 місяці на розробку основного функціоналу плюс тестування.

Для деталізації взаємодії компонентів розроблено діаграму послідовності, яка показує, як користувач переглядає дані. Процес включає:

– Користувач надсилає запит через веб-інтерфейс.

– Сервер (PHP) отримує запит і звертається до бази даних (MySQL).

– База повертає дані, які сервер обробляє та передає на клієнтську частину.

– Інтерфейс відображає карту або графік. Рис. 1.5

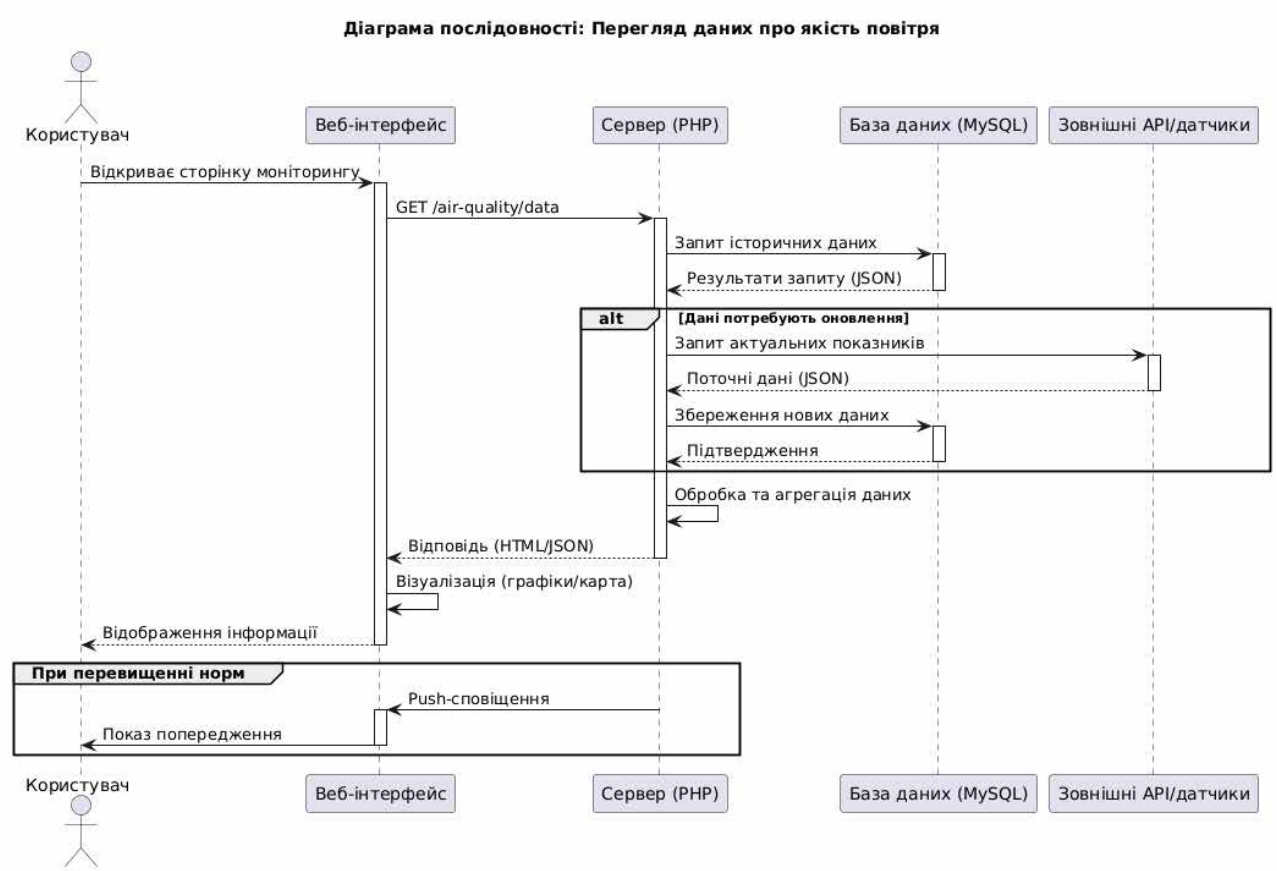

Рисунок 1.5 – Діаграма послідовності для перегляду даних

Додатково розроблено діаграму розгортання, яка описує апаратну та програмну архітектуру системи (рис. 1.6). Вона включає:

– Датчики: фізичні пристрої, підключені через Wi-Fi/GSM.

– Веб-сервер: сервер із PHP для обробки запитів.

– Сервер бази даних: MySQL для зберігання даних.

– Клієнтський пристрій: браузер користувача для доступу до системи.

Ці компоненти зв’язані через мережу Інтернет, що забезпечує розподілену роботу системи. ТЗ, підкріплене цими діаграмами, чітко окреслює структуру та процеси розробки, що полегшує перехід до етапу проєктування.

Таким чином, у першому розділі проведено аналіз предметної області моніторингу якості повітря, що виявив проблеми забруднення та їхній вплив на здоров’я. Визначено основні забруднювачі (PM2.5, PM10, NO_2 тощо) та структурні компоненти систем моніторингу: датчики, системи передачі даних,

бази даних і програмне забезпечення. Обмеження сучасних систем, такі як висока вартість і низька оперативність, вказують на потребу в гнучких рішеннях.

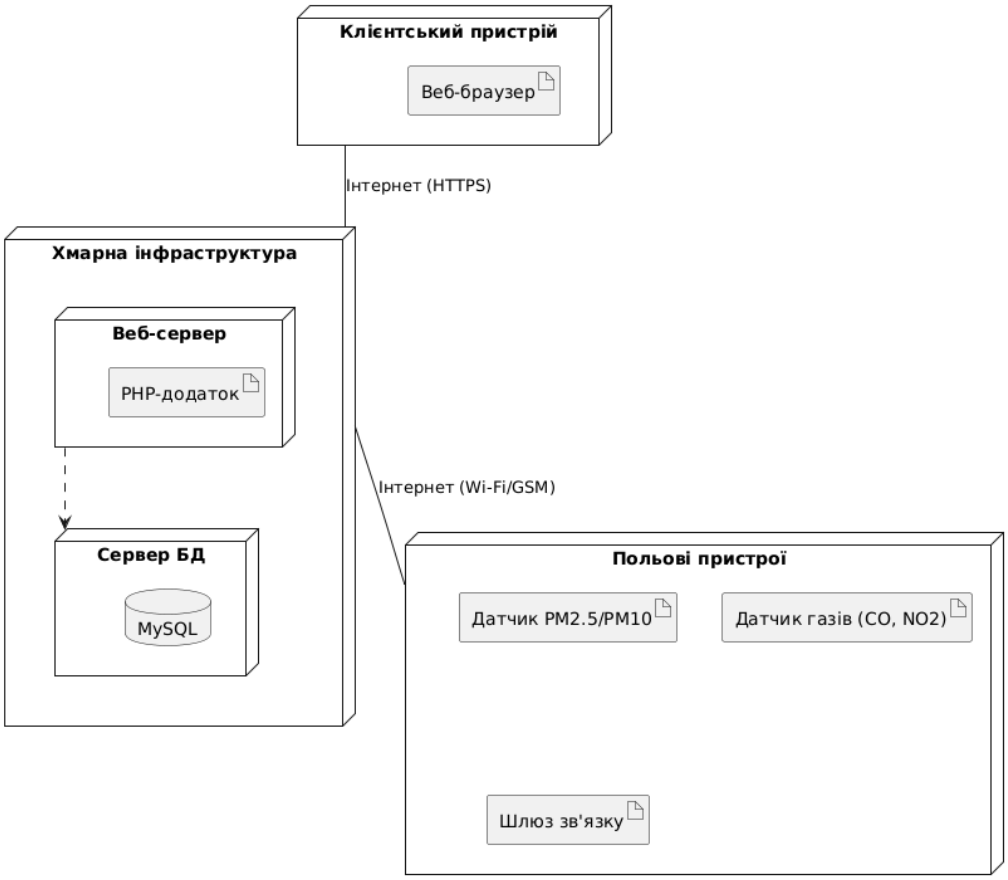

Рисунок 1.6 – Діаграма розгортання інформаційної системи

Досліджено аналоги (AirVisual, PurpleAir, OpenAQ, EEA, SaveDnipro), що дозволило виявити їхні переваги (доступність, точність) та недоліки (обмеження даних, неточність датчиків). Перспективними є інтерактивні карти та відкриті API для нової системи.

Визначено функціональні вимоги (збір, обробка, візуалізація даних, сповіщення) та нефункціональні (доступність, продуктивність, безпека). Технічне завдання на основі PHP і MySQL окреслює мету, функції та терміни розробки, підкріплені діаграмами Use Case, послідовності та розгортання. Це створює основу для проєктування ефективної системи.

РОЗДІЛ 2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА КОМПОНЕНТІВ ІНФОРМАЦІЙНОЇ СИСТЕМИ МОНІТОРИНГУ ПОВІТРЯ

2.1 Аналіз та вибір технологічного стеку для реалізації системи

Для створення ефективної, надійної та масштабованої інформаційної системи моніторингу повітря необхідно обґрунтовано підійти до вибору технологічного стеку. Технології мають забезпечити якісну взаємодію між клієнтською та серверною частиною, стабільну обробку запитів, гнучкість у зберіганні та аналізі даних, а також підтримку візуалізації інформації для кінцевого користувача. У рамках даного проєкту обрано мову PHP як основний інструмент веб-розробки та СУБД MySQL для організації зберігання даних. Також використовуються додаткові бібліотеки та інструменти для полегшення розробки та покращення інтерфейсу системи.

PHP обрано як основну мову для створення серверної логіки через її популярність, доступність, відкритий код і активну спільноту. PHP є широко застосовуваним інструментом для створення веб-додатків різного рівня складності, зокрема систем збору, обробки та відображення даних. Його синтаксис є зрозумілим, що дозволяє швидко реалізувати прототип системи, а надалі – розширювати функціональність без суттєвих витрат ресурсів [13].

До переваг PHP належать кросплатформеність, сумісність із більшістю веб-серверів (Apache, Nginx), легка інтеграція з MySQL та іншими СУБД, підтримка великої кількості фреймворків (Laravel, CodeIgniter), що забезпечують розширену функціональність. Оскільки система моніторингу передбачає обробку запитів у реальному часі, PHP дозволяє оперативно обробляти дані, реалізовувати механізми кешування, а також формувати відповіді користувачам із найменшою затримкою.Табл. 2.1

Таблиця 2.1 – Переваги використання PHP у веб-розробці

№	Характеристика	Пояснення
1	Простота	Легкий для вивчення, зрозумілий синтаксис
2	Висока продуктивність	Підходить для динамічних веб-додатків

3	Відкрите ПЗ	Безкоштовне середовище розробки
4	Широка підтримка	Велика кількість документації та навчальних матеріалів
5	Легка інтеграція	Сумісність з MySQL, HTML, JavaScript, CSS

MySQL було обрано як систему управління базами даних завдяки її надійності, високій швидкодії, підтримці транзакцій та масштабованості. MySQL – це одна з найпоширеніших у світі СУБД з відкритим кодом, яка ідеально підходить для реалізації веб-застосунків [14]. Вона забезпечує ефективне зберігання, фільтрацію, агрегацію та аналіз великого обсягу даних, що є критичним для проєктованої системи, яка буде опрацьовувати метео- та екологічні дані з десятків сенсорів.

MySQL легко інтегрується з PHP, має підтримку SQL-запитів для реалізації складних аналітичних функцій, а також дозволяє організувати захист даних шляхом налаштування прав доступу. У випадку системи моніторингу особливу увагу приділяється можливості зберігати історичні показники для аналізу трендів та побудови прогнозів забруднення. Для цього створюються таблиці з фіксацією часу, значень вимірювань та геолокації.Табл. 2.2

Таблиця 2.2 – Характеристики MySQL як СУБД для екологічних систем

№	Параметр	Пояснення
1	Надійність	Стабільна робота при великих обсягах даних
2	Масштабованість	Легке розширення обсягів бази без зниження продуктивності
3	Інтеграція з PHP	Вбудовані драйвери та бібліотеки для зв’язку з PHP
4	Гнучкість запитів	Підтримка складних SELECT, JOIN, GROUP BY, ORDER BY
5	Безпека	Права доступу, шифрування з’єднань

Крім основних технологій, для реалізації фронтенду системи використовуються HTML5, CSS3 і JavaScript [15]. Для побудови інтерактивних графіків обрано бібліотеку Chart.js, яка дозволяє ефективно візуалізувати зміну екологічних показників у реальному часі. Також у проєкті застосовуватиметься бібліотека Leaflet.js для реалізації інтерактивної карти з відображенням стану забруднення повітря в регіонах.

Для підвищення безпеки даних буде впроваджено захищене з’єднання HTTPS через SSL-сертифікати, а також механізми аутентифікації користувачів на основі сесій. Розробка та тестування відбуватиметься у середовищі OpenServer, що містить Nginx, PHP та MySQL у готовій для запуску конфігурації.

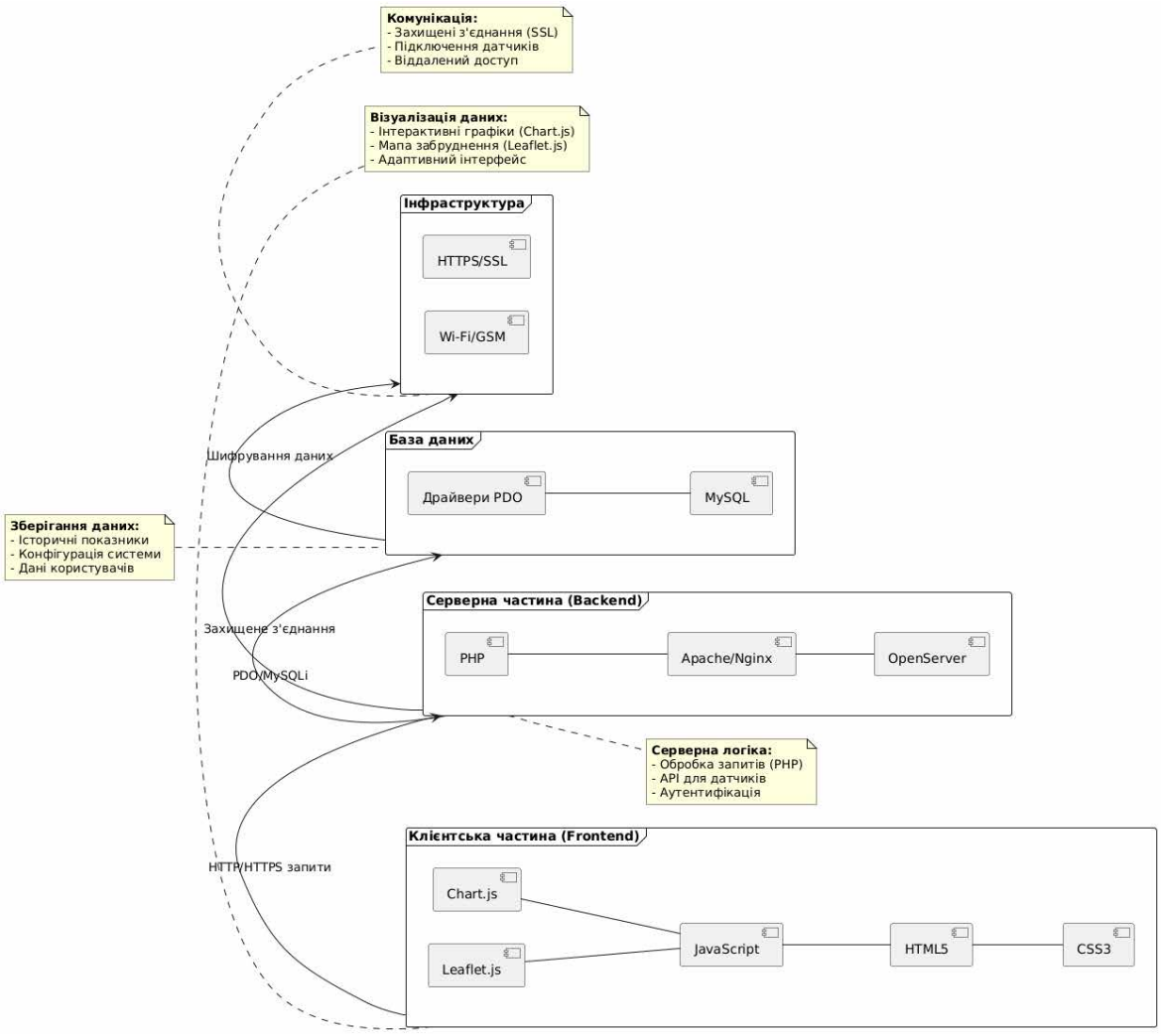

Рисунок 2.1 – Структура взаємодії між клієнтом, сервером та базою даних

Таким чином, вибраний технологічний стек забезпечує необхідну функціональність, гнучкість і масштабованість для реалізації системи моніторингу повітря. Поєднання PHP і MySQL дозволяє реалізувати потужну серверну логіку та ефективне управління екологічними даними, тоді як JavaScript-бібліотеки покращують взаємодію користувача з системою через візуальні елементи.

2.2 Проєктування логічної моделі бази даних інформаційної систем

Інформаційна система для моніторингу повітря є ключовим інструментом для організації, автоматизації та аналізу екологічних даних про стан атмосферного повітря. Основне завдання системи – забезпечити безперервний збір, обробку, зберігання та візуалізацію інформації про якість повітря в реальному часі. Це дозволяє вчасно реагувати на погіршення екологічної ситуації, інформувати населення та приймати обґрунтовані управлінські рішення в сфері охорони довкілля.

Система повинна зберігати дані про:

– станції спостереження (назва, розташування, координати, тип сенсорів, статус);

– вимірювання забруднювачів (дата і час фіксації, тип речовини – PM2.5, PM10, NO_2, SO_2, O_3, CO_2, значення концентрації, одиниці виміру);

– метеоумови (температура, вологість, атмосферний тиск, швидкість та напрям вітру);

– користувачів системи (П.І.Б., електронна пошта, роль у системі – користувач, адміністратор);

– повідомлення про перевищення норм (час, місце події, тип перевищеного забруднювача, рівень небезпеки).

Крім того, система повинна підтримувати функціональність для аналізу та прогнозування стану повітря, таких як:

– виявлення ділянок із постійно підвищеним рівнем забруднення;

– побудова динаміки змін за обраний період часу;

– визначення сезонних коливань концентрацій шкідливих речовин;

– інтеграція з відкритими джерелами даних (API) для покращення якості прогнозів.

Основна мета створення бази даних – забезпечити централізоване та структуроване зберігання екологічної інформації, зменшити залежність від ручного аналізу, прискорити обробку великих обсягів даних, забезпечити надійне інформування населення і фахівців, а також підвищити прозорість та оперативність екологічного моніторингу.

Без такої системи забезпечення контролю якості повітря є неефективним: дані збираються із запізненням або неповно, відсутні інтерактивні інструменти для візуалізації, а аналіз і прогнозування залишаються ускладненими. Це може призводити до несвоєчасного реагування на екологічні загрози, погіршення стану здоров’я населення та зростання екологічних ризиків. Інформаційна система моніторингу повітря дозволяє зробити процеси оцінки якості повітря доступними, прозорими та точними, що є вкрай важливим в умовах зростання техногенного навантаження на навколишнє середовище.

Перелік прецедентів використання інформаційної системи «Моніторинг якості повітря» наступний:

1. Прецедент: Перегляд поточних даних. Користувач має змогу в реальному часі отримати інформацію про стан забруднювачів повітря (PM2.5, PM10, NO_2, SO_2, CO, O_3) з інтерактивної карти або таблиць.

2. Прецедент: Отримання сповіщень. Система автоматично інформує користувача про перевищення критичних порогів концентрації шкідливих речовин через push- або email-сповіщення.

3. Прецедент: Аналіз трендів. Користувач або аналітик аналізує історичні дані, будує графіки змін забруднень, порівнює динаміку у різні часові періоди.

4. Прецедент: Налаштування датчиків. Адміністратор здійснює налаштування параметрів роботи сенсорів: додає або видаляє точки збору даних, змінює частоту надсилання показників.

5. Прецедент: Калібрування датчиків. Технік виконує технічне обслуговування та калібрування сенсорів відповідно до еталонних значень для забезпечення точності вимірювань.

6. Прецедент: Генерація звітів. Аналітик формує статистичні звіти про якість повітря за обраний період для подання в екологічні служби або державні органи.

7. Прецедент: Прогнозування якості повітря. Аналітик виконує моделювання майбутніх значень забруднення з використанням машинного навчання чи статистичних методів на основі накопичених даних.

Цей перелік повністю відповідає побудованій use-case діаграмі на рис. 1.4 й описує основні дії учасників системи. Далі було проведено морфологічний аналіз з виділенням кандидатів у сутності, атрибути та зв’язки для предметної області «Моніторинг повітря».

Іменники та їхній аналіз (виділення кандидатів у сутності та атрибути):

1. Станція моніторингу – точка збору екологічних даних. Має атрибути: ідентифікатор, назва, координати (широта, довгота), адреса, тип, статус.

2. Вимірювання – зафіксований екологічний показник у певний момент часу. Має атрибути: дата й час, значення, одиниця виміру, тип забруднювача (PM2.5, NO_2, SO_2 тощо), температура, вологість.

3. Забруднювач – тип шкідливої речовини, що присутня в атмосферному повітрі. Має атрибути: назва, нормативне значення, одиниця виміру, критичний поріг.

4. Користувач – особа, яка переглядає інформацію або адмініструє систему. Має атрибути: ПІБ, email, пароль, роль (користувач, аналітик, адміністратор).

5. Повідомлення – сповіщення про перевищення порогових значень. Має атрибути: дата й час, тип забруднювача, рівень небезпеки, статус (переглянуто/непереглянуто).

6. Звіт – сформований документ на основі даних за період. Атрибути: дата створення, період охоплення, відповідальний користувач, тип (щоденний, місячний).

Дієслова та їхній аналіз (виділення зв’язків між сутностями):

1. Станція моніторингу здійснює Вимірювання.

2. Вимірювання фіксує Забруднювач.

3. Користувач отримує Повідомлення.

4. Користувач генерує Звіт.

5. Звіт ґрунтується на Вимірюваннях.

6. Станція моніторингу зв’язується з Користувачем через Повідомлення.

Список сутностей, атрибутів і зв’язків представлено у вигляді табл. 2.3 та табл. 2.4.

Таблиця 2.3 – Сутності та атрибути

Сутність	Атрибути
Станція моніторингу	ID, назва, адреса, координати (широта, довгота), тип, статус
Вимірювання	ID, дата й час, значення, одиниця виміру, температура, вологість
Забруднювач	ID, назва, нормативне значення, критичний поріг, одиниця виміру
Користувач	ID, ПІБ, email, пароль, роль
Повідомлення	ID, дата й час, тип забруднювача, рівень небезпеки, статус
Звіт	ID, дата створення, період охоплення, ID користувача, тип звіту

Таблиця 2.4 – Зв’язки

Сутність 1	Зв’язок	Сутність 2	Опис зв’язку
Станція моніторингу	здійснює	Вимірювання	Кожна станція регулярно надсилає дані про стан повітря
Вимірювання	фіксує	Забруднювач	Кожне вимірювання стосується певного виду забруднення
Користувач	отримує	Повідомлення	Користувач отримує сповіщення про перевищення порогових рівнів
Користувач	генерує	Звіт	Аналітик або адміністратор створює звіти за період
Звіт	ґрунтується на	Вимірювання	Звіт формується на основі накопичених вимірювань
Станція моніторингу	зв’язується через	Повідомлення	Повідомлення створюються на основі даних від певної станції

Рис. 2.2

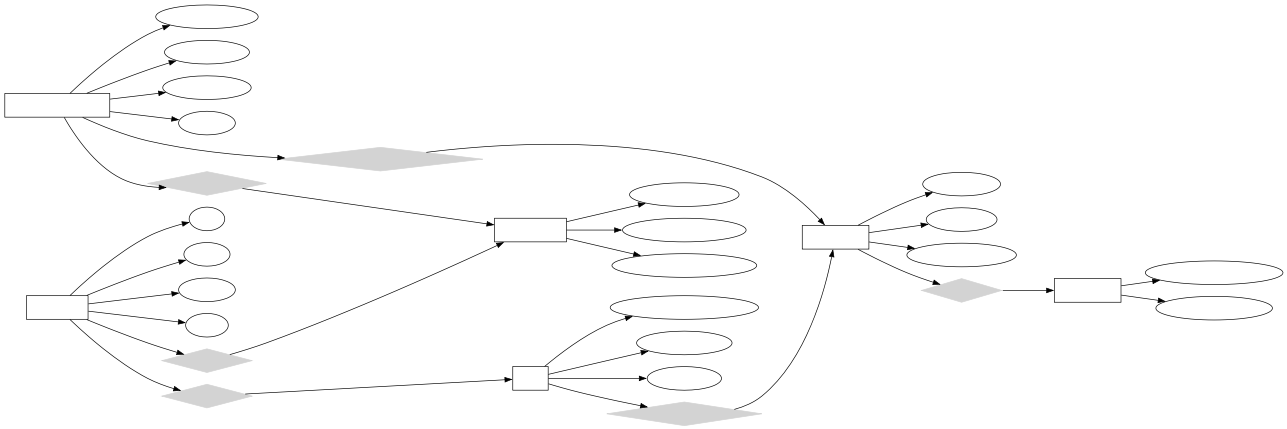

Рисунок 2.2 – Діаграма «сутність-зв'язок» етапу концептуального проектування (в нотації Чена з визначенням кардинальності зв'язків)

2.3 Розробка фізичної моделі бази даних

Фізична модель бази даних інформаційної системи моніторингу якості повітря розроблена на основі діаграми класів і реалізована в MySQL. Модель включає шість основних таблиць та одну допоміжну для забезпечення всіх функціональних вимог системи. Структура таблиць відповідає логічній моделі та охоплює ключові сутності: станції моніторингу, вимірювання, забруднювачі, користувачі, сповіщення та звіти.

– monitoring_stations: Зберігає інформацію про станції моніторингу. Поля: id (унікальний ідентифікатор), name (назва станції), coordinates (географічні координати), type (тип станції, наприклад, стаціонарна чи мобільна), status (стан, наприклад, активна чи неактивна).

– pollutants: Містить дані про забруднювачі. Поля: id, name (назва, наприклад, PM2.5), threshold (порогове значення), unit (одиниця вимірювання).

– measurements: Фіксує результати вимірювань. Поля: id, station_id (посилання на станцію), pollutant_id (посилання на забруднювач), timestamp (час вимірювання), value (значення), unit (одиниця вимірювання).

– users: Зберігає дані користувачів. Поля: id, full_name (ПІБ), email (унікальна адреса), password (зашифрований пароль), role (роль: user, admin, analyst).

– alerts: Містить сповіщення про критичні рівні забруднення. Поля: id, station_id, user_id (посилання на користувача), timestamp, type (тип сповіщення), danger_level (рівень небезпеки), status (unread/read).

– reports: Зберігає звіти, створені користувачами. Поля: id, user_id, created_date (дата створення), period (період звіту), type (тип звіту).

– report_measurements: Допоміжна таблиця для реалізації зв’язку "багато до багатьох" між звітами та вимірюваннями. Поля: report_id, measurement_id.

Табл. 2.5 узагальнює структуру основних таблиць.

Таблиця	Основні поля	Призначення
monitoring_stations	id, name, coordinates, type, status	Дані про станції моніторингу
pollutants	id, name, threshold, unit	Інформація про забруднювачі
measurements	id, station_id, pollutant_id, timestamp	Результати вимірювань
users	id, full_name, email, password, role	Дані користувачів
alerts	id, station_id, user_id, timestamp	Сповіщення про критичні рівні
reports	id, user_id, created_date, period, type	Звіти користувачів

Таблиця 2.5 – Структура таблиць бази даних

Зв’язки між таблицями реалізовано через зовнішні ключі, що відображають відношення, визначені в діаграмі класів. Основні зв’язки:

– monitoring_stations → measurements: Один до багатьох (1:N). Одна станція може мати багато вимірювань (station_id у measurements посилається на id у monitoring_stations).

– pollutants → measurements: Один до багатьох (1:N). Один забруднювач вимірюється в багатьох записах (pollutant_id у measurements посилається на id у pollutants).

– users → alerts: Один до багатьох (1:N). Користувач може отримувати багато сповіщень (user_id у alerts посилається на id у users).

– users → reports: Один до багатьох (1:N). Користувач створює багато звітів (user_id у reports посилається на id у users).

– monitoring_stations → alerts: Один до багатьох (1:N). Станція асоціюється з багатьма сповіщеннями (station_id у alerts посилається на id у monitoring_stations).

– reports ↔ measurements: Багато до багатьох (M:N). Реалізовано через допоміжну таблицю report_measurements, яка зв’язує report_id і measurement_id.

Рис. 2.3 ілюструє фізичну модель бази даних із зв’язками. Для підвищення ефективності бази даних виконано такі заходи оптимізації:

– Індексація: Додано первинні ключі (id) з автоінкрементом для всіх таблиць, що забезпечує швидкий доступ до записів. Зовнішні ключі (station_id, pollutant_id, user_id) проіндексовано для прискорення JOIN-запитів.

– Нормалізація: Дані нормалізовано до третьої нормальної форми (3NF), щоб уникнути надмірності. Наприклад, інформація про забруднювачі винесена в окрему таблицю pollutants, а не дублюється в measurements.

– Обмеження даних: Використано NOT NULL для обов’язкових полів (наприклад, name у monitoring_stations, email у users) та UNIQUE для унікальних значень (наприклад, email). Поле role у users обмежено типом ENUM для зменшення обсягу даних.

– Оптимізація зв’язків: Допоміжна таблиця report_measurements мінімізує складність запитів для звітів, дозволяючи ефективно пов’язувати вимірювання з різними звітами.

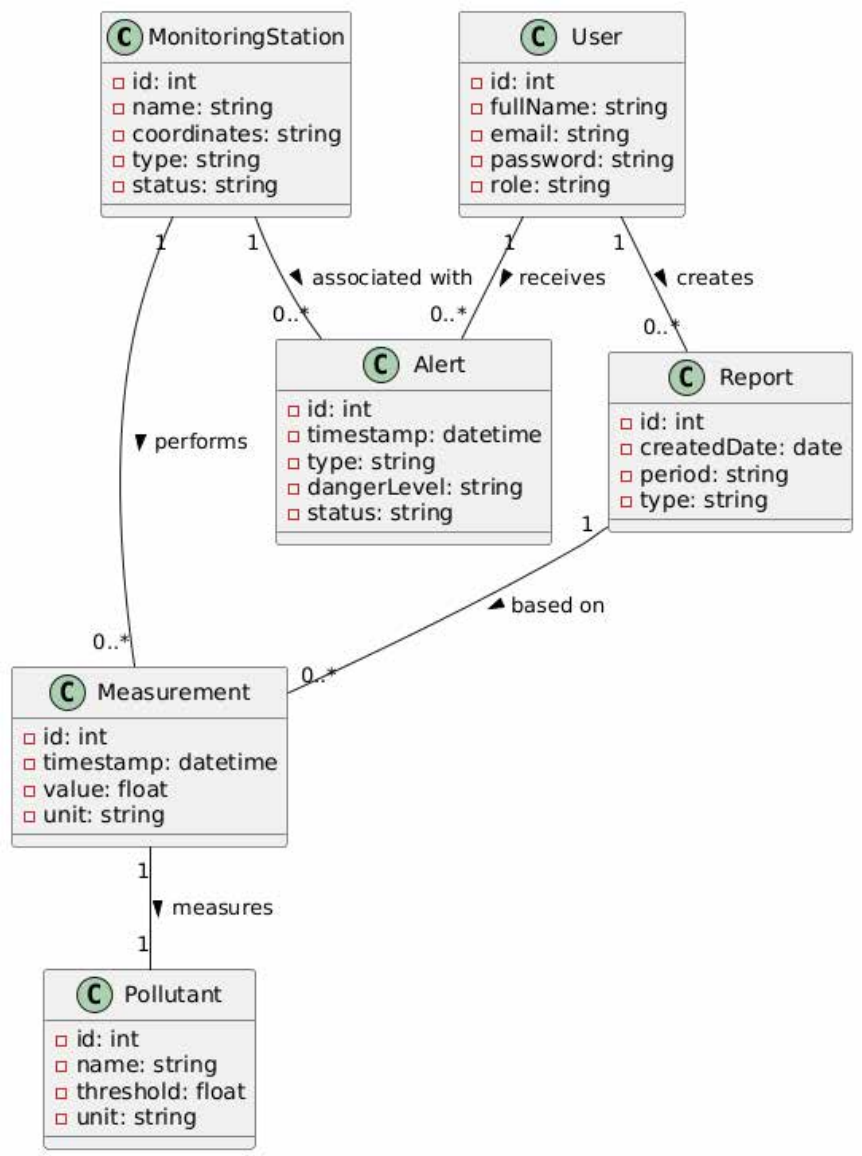

Рисунок 2.3 – Фізична модель бази даних системи моніторингу повітря

Ці заходи забезпечують швидкодію, цілісність даних і масштабованість бази даних, що критично важливо для обробки великих обсягів вимірювань у реальному часі.

2.4 Проєкт людино-машинного інтерфейсу

Людино-машинний інтерфейс (ЛМІ) інформаційної системи моніторингу якості повітря розроблено з урахуванням потреб користувачів, таких як громадяни, екологи та адміністратори, а також вимог до зручності, доступності та адаптивності [18]. Основна мета інтерфейсу – забезпечити інтуїтивно

зрозумілу взаємодію, швидкий доступ до даних про якість повітря та можливість виконання ключових функцій, визначених у технічному завданні.

Основні елементи інтерфейсу включають:

– Інтерактивна карта: центральний компонент, який відображає розташування станцій моніторингу з кольоровими маркерами, що вказують на рівень забруднення (зелений – низький, червоний – критичний).

– Графіки та таблиці: для візуалізації даних про концентрацію забруднювачів (PM2.5, PM10, NO_2 тощо) у реальному часі та за обраний період.

– Панель сповіщень: відображає повідомлення про перевищення порогових значень із можливістю фільтрації за статусом (прочитані/непрочитані).

– Меню навігації: забезпечує доступ до основних розділів – "Карта", "Дані", "Звіти", "Налаштування" (для адміністраторів).

– Форма авторизації: для входу користувачів і розмежування доступу за ролями (користувач, адміністратор, аналітик).

– Фільтри та пошукові поля: дозволяють користувачам обирати станції, забруднювачі чи періоди для аналізу.

Ці елементи формують основу веб-інтерфейсу, реалізованого за допомогою HTML, CSS і JavaScript, що забезпечує його адаптивність для різних пристроїв – від настільних комп’ютерів до смартфонів [19]. Для підвищення зручності використано принципи мінімалістичного дизайну: чітка структура, контрастні кольори та зрозумілі іконки. Наприклад, карта використовує бібліотеку Leaflet.js [16] для динамічного відображення даних, а графіки – Chart.js [17] для створення інтерактивних діаграм.

Схеми екранних форм і меню розроблено з урахуванням сценаріїв використання, описаних у Use Case діаграмі (див. розділ 1.3). Інтерфейс поділено на кілька основних сторінок, кожна з яких відповідає певному функціоналу системи. Нижче описано ключові екранні форми:

– Головна сторінка (карта): Містить інтерактивну карту з маркерами станцій, панель фільтрів (за забруднювачами чи регіонами) і бічну панель із короткою статистикою (середні значення забруднення за день). Користувач може клікнути на маркер, щоб відкрити спливаюче вікно з детальними даними про станцію.

– Сторінка даних: Відображає таблицю вимірювань із можливістю сортування за датою, станцією чи забруднювачем, а також графіки для аналізу трендів. Є кнопка для експорту даних у CSV.

– Сторінка звітів: Дозволяє користувачам створювати звіти, обираючи період, тип (наприклад, середньодобовий чи тижневий) і станції. Звіт можна переглянути у браузері або завантажити у PDF.

– Сторінка сповіщень: Показує список сповіщень із зазначенням часу, рівня небезпеки та пов’язаної станції. Користувач може позначити сповіщення як прочитане.

– Адмін-панель: Доступна лише адміністраторам для налаштування датчиків (додавання/видалення станцій) і керування користувачами (редагування ролей).

Меню навігації реалізовано у вигляді верхньої панелі, яка фіксується при прокручуванні сторінки, щоб користувач завжди мав доступ до основних розділів [19]. Для мобільних пристроїв меню трансформується в бургер-меню, що економить простір. Табл. 2.6 узагальнює основні екранні форми та їх функціонал.

Таблиця 2.6 – Основні екранні форми інтерфейсу

Форма	Опис	Функціонал
Головна (карта)	Інтерактивна карта зі статистикою	Перегляд стану станцій, фільтрація
Дані	Таблиця і графіки вимірювань	Сортування, експорт даних
Звіти	Створення та перегляд звітів	Вибір періоду, завантаження PDF
Сповіщення	Список сповіщень	Фільтрація, позначення прочитаних

Адмін-панель	Налаштування системи	Керування станціями та користувачами

Рис. 2.4 ілюструє прототип головної сторінки з картою.

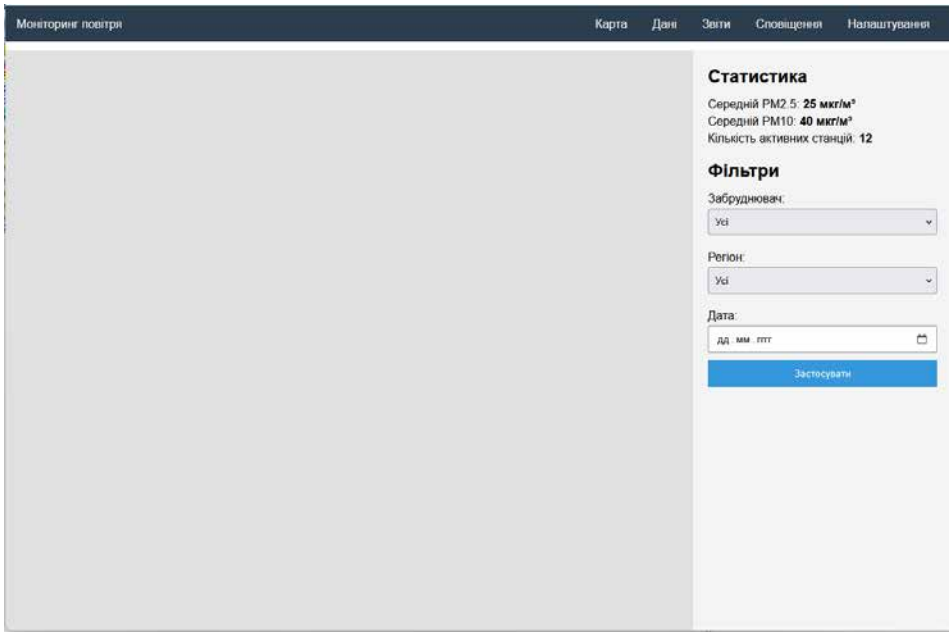

Рисунок 2.4 – Прототип головної сторінки системи

Взаємодія користувача з системою побудована навколо типових сценаріїв, визначених у п. 1.3, і відповідає ролям користувачів. Основні сценарії включають перегляд даних, аналіз трендів, отримання сповіщень і адміністрування.

– Перегляд даних: Користувач заходить на головну сторінку, де бачить карту з маркерами станцій. Клік на маркер відкриває деталі: поточні значення забруднювачів (наприклад, PM2.5 – 25 мкг/м³) і статус станції. Через фільтри користувач може обрати конкретний забруднювач чи регіон. На сторінці "Дані" доступна таблиця з історією вимірювань, яку можна відсортувати за датою чи значенням. Наприклад, користувач обирає період "Останній тиждень" і бачить графік зміни NO_2 для обраної станції.

– Аналіз трендів: На сторінці "Дані" користувач може побудувати графік для кількох забруднювачів одночасно, порівнюючи їхні значення. Для аналітиків

передбачено інструмент прогнозування, який показує можливі тенденції на основі історичних даних (реалізовано через JavaScript-бібліотеку).

– Отримання сповіщень: Якщо система фіксує перевищення порогового значення (наприклад, PM10 > 50 мкг/м³), користувач отримує сповіщення на сторінці "Сповіщення". Повідомлення містить час, станцію, рівень небезпеки та рекомендацію (наприклад, "Уникайте перебування на вулиці"). Користувач може позначити його як прочитане або налаштувати частоту сповіщень у профілі.

– Адміністрування: Адміністратор через адмін-панель додає нову станцію, вказуючи її координати та тип датчика, або редагує дані користувача, змінюючи його роль (з user на analyst). Панель також дозволяє переглядати логи системи для діагностики.

Рис. 2.5 демонструє прототип сторінки сповіщень.

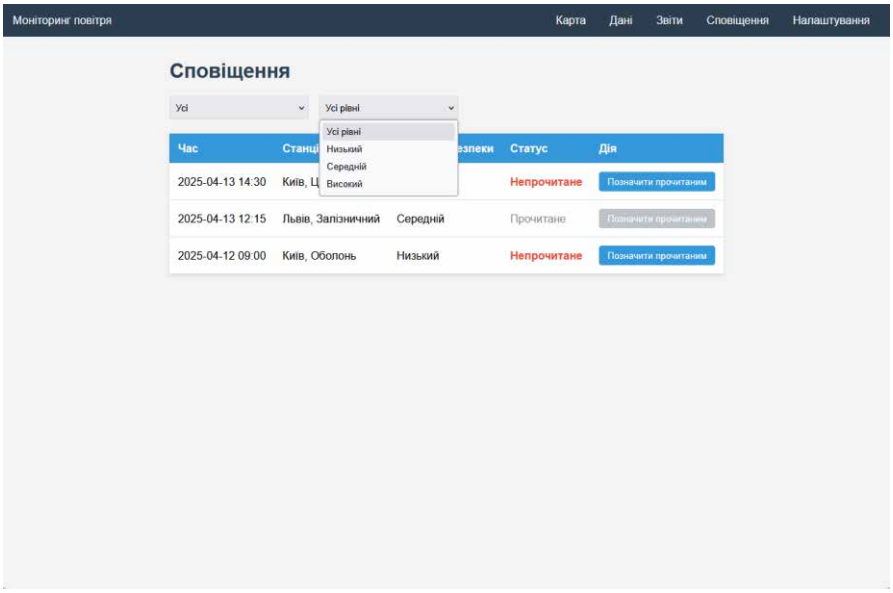

Рисунок 2.5 – Прототип сторінки сповіщень

Для забезпечення зручності взаємодії використано адаптивний дизайн: на смартфонах карта займає весь екран, а меню ховається в бургер-іконку [20]. Елементи керування, такі як кнопки фільтрів чи експорту, мають достатній розмір для сенсорного введення. Для користувачів із обмеженим доступом до швидкого інтернету передбачено кешування даних через Service Worker, що дозволяє переглядати останні збережені показники офлайн.

Взаємодія з системою підтримує кілька мов (українська, англійська), що налаштовується в профілі користувача. Для підвищення доступності дотримано стандартів WCAG 2.1: достатній контраст тексту, підтримка клавіатурної навігації та альтернативні описи для графіків. Наприклад, графік PM2.5 супроводжується текстовим описом ("Середнє значення PM2.5 за тиждень – 20 мкг/м³").

Інтерфейс протестовано на відповідність принципам usability через прототипи [21]. Результати показали високу ефективність дизайну. У майбутньому планується додати інтеграцію з мобільними додатками для push-сповіщень, що ще більше спростить доступ до системи.

2.5 Аналіз методів реалізації та вибір оптимального варіанту

Розробка інформаційної системи моніторингу якості повітря, яка розгортається локально з використанням OpenServer, PHP і MySQL, передбачає вибір оптимального підходу до реалізації, що враховує функціональні та нефункціональні вимоги, описані в п. 1.3. На основі аналізу предметної області та сучасних веб-технологій розглянуто три основні підходи до створення системи: використання чистого PHP із шаблонами, застосування PHP-фреймворку (наприклад, Laravel), а також розробка з використанням CMS (наприклад, WordPress із кастомними плагінами). Кожен із цих методів має свої особливості, які впливають на швидкість розробки, масштабованість і зручність підтримки системи.

Чистий PHP із шаблонами передбачає створення веб-сайту з нуля, використовуючи PHP для серверної логіки, MySQL для бази даних і HTML/CSS/JavaScript для клієнтської частини. У цьому підході розробник самостійно створює маршрутизацію, обробку запитів і шаблони сторінок, що розміщуються на локальному сервері через OpenServer. Такий метод дозволяє

повністю контролювати архітектуру системи та мінімізувати залежності від сторонніх бібліотек.

PHP-фреймворк (Laravel) пропонує структурований підхід до розробки, використовуючи готовий набір інструментів для маршрутизації, ORM (Eloquent для роботи з MySQL), шаблонізації (Blade) і безпеки. Laravel спрощує створення API для інтеграції з датчиками, керування користувачами та генерацію звітів. Локальне розгортання через OpenServer забезпечує швидке налаштування середовища за допомогою вбудованих інструментів, таких як Composer для управління залежностями.

CMS (WordPress із плагінами) дозволяє швидко створити веб-сайт, використовуючи готову систему керування контентом. Для реалізації моніторингу можна розробити кастомний плагін на PHP для обробки даних із датчиків і відображення їх через шорткоди або спеціальні сторінки. WordPress із MySQL легко розгортається на OpenServer, а наявність тем і плагінів прискорює створення інтерфейсу.

Кожен підхід має свої сильні сторони, але вибір залежить від вимог до гнучкості, продуктивності та складності реалізації. Для детального порівняння розглянуто їхні переваги та недоліки.

Для оцінки підходів використано такі критерії: швидкість розробки, гнучкість, масштабованість, безпека, складність підтримки та відповідність локальному розгортанню через OpenServer. Порівняння наведено в табл. 2.5, яка підсумовує ключові характеристики кожного методу.

Чистий PHP із шаблонами вирізняється максимальною гнучкістю, оскільки розробник може створювати систему точно під задані вимоги без обмежень фреймворків чи CMS. Локальне розгортання через OpenServer є простим: достатньо налаштувати PHP і MySQL, скопіювати файли та запустити сервер. Однак цей підхід потребує значних зусиль для реалізації таких функцій, як авторизація, маршрутизація чи захист від SQL-ін’єкцій, що може збільшити

час розробки. Підтримка системи ускладнюється через відсутність стандартної структури коду, особливо якщо проєкт розростається.Табл. 2.7

Таблиця 2.7 – Порівняння методів реалізації системи

Метод	Переваги	Недоліки
Чистий PHP	Повний контроль, мінімальні залежності, швидке розгортання на OpenServer	Висока трудомісткість, складність підтримки великих проєктів
Laravel	Структурованість, вбудовані інструменти безпеки, підтримка API	Потреба вивчення фреймворку, більший обсяг залежностей
WordPress	Швидка розробка, готові теми, простота для нетехнічних користувачів	Обмежена гнучкість, низька продуктивність для складних систем

Laravel забезпечує швидке створення системи завдяки готовим рішенням: middleware для безпеки, Eloquent для роботи з базою даних і Blade для шаблонів. Фреймворк підтримує створення REST API, що полегшує інтеграцію з датчиками через HTTP-запити. Локальне середовище на OpenServer легко налаштовується з Composer, а Laravel Artisan спрощує генерацію моделей і контролерів. Проте фреймворк додає залежності, що може ускладнити розгортання на слабких серверах, а початкове вивчення його структури потребує часу. Безпека (наприклад, захист від XSS чи CSRF) вбудована, що зменшує ризик помилок.

WordPress приваблює швидкістю розробки: за допомогою плагінів і тем можна створити інтерфейс за кілька днів. Розгортання на OpenServer не потребує додаткових налаштувань, оскільки платформа підтримує PHP і MySQL "з коробки". Однак WordPress погано підходить для складних систем моніторингу через обмеження в оптимізації бази даних і низьку продуктивність при обробці великих обсягів даних із датчиків. Кастомний плагін для API чи звітів потребує значних зусиль, а вбудована структура WordPress ускладнює масштабування. Безпека залежить від регулярних оновлень ядра та плагінів, що може бути проблемою для локального розгортання.

Рис. 2.6 ілюструє порівняння підходів за ключовими критеріями.

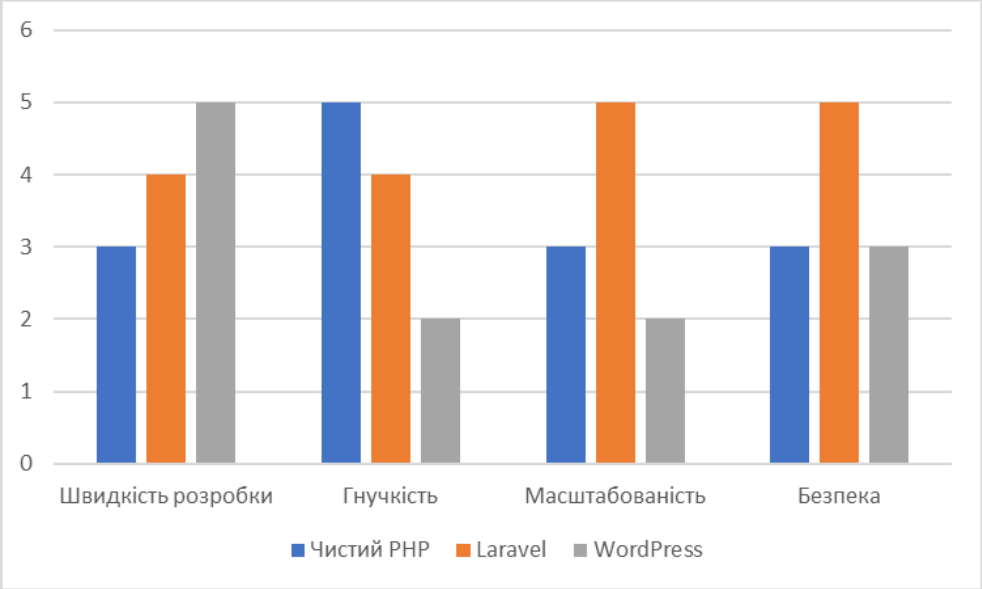

Рисунок 2.6 – Порівняння методів реалізації системи

З урахуванням вимог до системи, локального розгортання на OpenServer і аналізу методів обрано підхід із використанням чистого PHP із шаблонами як оптимальний для даного проєкту. Обґрунтування вибору базується на таких факторах:

По-перше, чистий PHP забезпечує повний контроль над кодом, що дозволяє точно реалізувати функціонал, описаний у технічному завданні (збір даних, візуалізація, сповіщення). Для локального сервера OpenServer, який зазвичай використовується на обмежених ресурсах, цей підхід є найменш вимогливим, оскільки не потребує встановлення додаткових залежностей, як у Laravel, чи громіздкої структури, як у WordPress.

По-друге, система моніторингу повітря не є надмірно складною, тому використання фреймворку може бути надлишковим. Наприклад, маршрутизацію можна реалізувати через простий PHP-скрипт, який обробляє GET/POST-запити, а шаблони сторінок (карта, звіти) створюються за допомогою включення PHP-файлів (include). Для роботи з MySQL використано PDO, що забезпечує безпечну обробку запитів і захист від SQL-ін’єкцій.

По-третє, чистий PHP спрощує інтеграцію з датчиками через HTTP API. Дані від датчиків надходять у форматі JSON, який обробляється PHP-скриптом і зберігається в MySQL. Це дозволяє уникнути складних налаштувань, необхідних у Laravel для middleware чи роутингу. OpenServer забезпечує стабільне середовище для тестування API за допомогою вбудованого Apache.

Четвертим фактором є швидкість розробки для локального розгортання. Хоча чистий PHP потребує більше коду для базових функцій (наприклад, авторизації), OpenServer дозволяє швидко налаштувати сервер і базу даних, а структура проєкту залишається простою: папки для скриптів, шаблонів і статичних файлів (CSS, JavaScript). На відміну від WordPress, чистий PHP не обмежує гнучкість дизайну інтерфейсу, що важливо для реалізації інтерактивної карти та графіків (з використанням Leaflet.js і Chart.js).Рис. 2.7

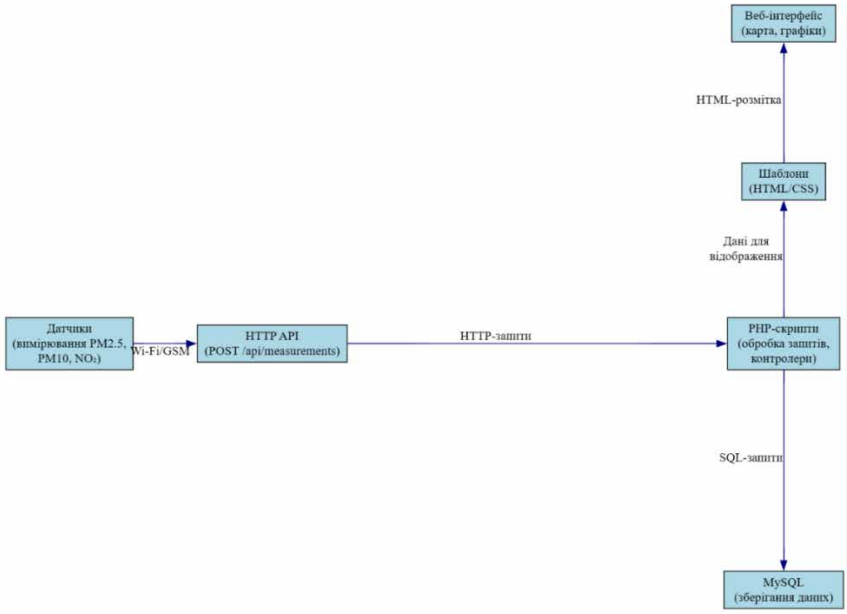

Рисунок 2.7 – Архітектура системи з використанням чистого PHP

Недоліки чистого PHP, такі як трудомісткість і складність підтримки, мінімізуються завдяки чіткій структурі проєкту:

– окремі файли для обробки API (api.php), шаблонів (views/), і логіки (controllers/);

– використання PDO для безпечних запитів до MySQL;

– модульний підхід до інтерфейсу, де кожна сторінка (карта, сповіщення) є окремим шаблоном.

Для порівняння, Laravel міг би прискорити розробку складних систем, але його залежності (Composer, Laravel Mix) ускладнюють локальне розгортання на слабких машинах. WordPress, хоча й швидкий для прототипів, не відповідає вимогам до продуктивності та гнучкості для обробки даних із датчиків у реальному часі. Рис. 2.7 демонструє архітектуру системи з чистим PHP.

Отже, чистий PHP із шаблонами оптимально балансує між гнучкістю, простотою розгортання та відповідністю вимогам проєкту. Цей метод дозволяє швидко реалізувати систему на локальному сервері, забезпечуючи її працездатність і можливість подальшого масштабування за потреби.

Таким чином, у другому розділі в результаті виконання етапу проєктування було обґрунтовано вибір технологічного стеку для реалізації інформаційної системи моніторингу повітря, побудовано логічну та фізичну моделі бази даних, а також розроблено людино-машинний інтерфейс, орієнтований на зручність використання та доступність даних у реальному часі. Поєднання PHP, MySQL, JavaScript та сучасних веб-бібліотек дозволяє створити ефективну та адаптивну систему, яка забезпечує збір, обробку, візуалізацію та аналіз екологічної інформації.

Особливу увагу приділено забезпеченню цілісності даних, масштабованості бази, зручності користувацького інтерфейсу та відповідності функціональним і нефункціональним вимогам. Результати розділу формують міцну основу для реалізації прикладного рівня системи та її подальшого розгортання.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ МОНІТОРИНГУ ЯКОСТІ ПОВІТРЯ AIRGUARD

3.1 Реалізація модулів інформаційної системи

Інформаційна система AirGuard, розроблена для моніторингу якості повітря, побудована з використанням технічного стеку, що включає OSPanel 6.2.6 як локальне середовище, Nginx 1.27 як веб-сервер [22], PHP 8.4 [23] для серверної логіки та MySQL 8.4 для управління даними. Проєкт структуровано модульно, що забезпечує чіткий поділ функціоналу між серверною частиною, базою даних і клієнтською частиною веб-сайту. Реалізація відповідає вимогам, описаним у п.2.4, і враховує прототипи сторінок. Нижче детально описано розробку кожного компонента системи з урахуванням визначеної структури проєкту.

Серверна частина системи реалізована на PHP 8.4, що забезпечує високу продуктивність завдяки новим можливостям, таким як покращена типізація та оптимізований парсер. Локальне середовище OSPanel 6.2.6 із веб-сервером Nginx 1.27 обробляє HTTP-запити, передаючи динамічні запити до PHP-FPM, тоді як статичні файли (CSS, JavaScript, зображення) обслуговуються безпосередньо для підвищення швидкості. Основні PHP-файли розміщено в директорії public/, яка є кореневою для веб-доступу, а конфігураційний файл config.php розташовано в корені AirGuard/ для централізованого налаштування підключення до бази даних AirQualityDB.

Серверна логіка поділена на модулі, кожен із яких відповідає екранним формам, описаним у таблиці 2.4. Модуль авторизації, реалізований у файлі login.php, забезпечує вхід користувачів шляхом порівняння введених даних із таблицею users. Для захисту від SQL-ін’єкцій використано PDO з підготовленими запитами, а вхідні дані валіюються через filter_input. Наприклад, перевірка пароля здійснюється шляхом прямого порівняння з полем password, як визначено в структурі users, де паролі зберігаються у відповідній колонці.

Модуль обробки даних (api.php) приймає запити від датчиків у форматі JSON через HTTP POST, декодує їх і зберігає результати в таблиці measurements. Для оптимізації використано пакетну вставку даних, що зменшує навантаження на базу. Модуль відображення даних (data.php) формує таблиці та графіки вимірювань, дозволяючи сортувати записи за датою, станцією чи забруднювачем, із можливістю експорту в CSV. Модуль звітів (reports.php) генерує звіти на основі таблиць reports і report_measurements, із підтримкою експорту в PDF через бібліотеку TCPDF.

Модуль сповіщень (alerts.php) відображає записи з таблиці alerts, із фільтрами за статусом (unread/read) і рівнем небезпеки, як показано в прототипі alerts.html. Модуль адміністрування (admin_panel.php) обмежує доступ перевіркою ролі admin через функцію isAdmin, дозволяючи додавати/видаляти станції в таблиці monitoring_stations і змінювати ролі користувачів. Усі модулі використовують єдине підключення до бази через config.php, що забезпечує консистентність.Табл. 3.1

Таблиця 3.1 – Основні PHP-модулі системи

Модуль	Файл	Функціонал
Авторизація	login.php	Вхід користувачів, перевірка email і пароля
Обробка даних	api.php	Збереження даних від датчиків
Дані	data.php	Таблиці та графіки вимірювань, експорт у CSV
Звіти	reports.php	Генерація звітів, експорт у PDF
Сповіщення	alerts.php	Фільтрація та керування сповіщеннями
Адмін-панель	admin_panel.php	Керування станціями та ролями користувачів

Сесії керуються через вбудований механізм PHP, із конфігурацією session.save_path у OSPanel для надійного зберігання. Nginx оптимізовано для швидкого обслуговування файлів у public/assets/ (leaflet.js, chart.js), що зменшує затримки.

База даних AirQualityDB, створена на MySQL 8.4, забезпечує структуроване зберігання даних системи. Її схема включає сім таблиць, які

підтримують усі сценарії використання, описані в п. 1.3. Схема бази даних наведена на рис. 3.1.

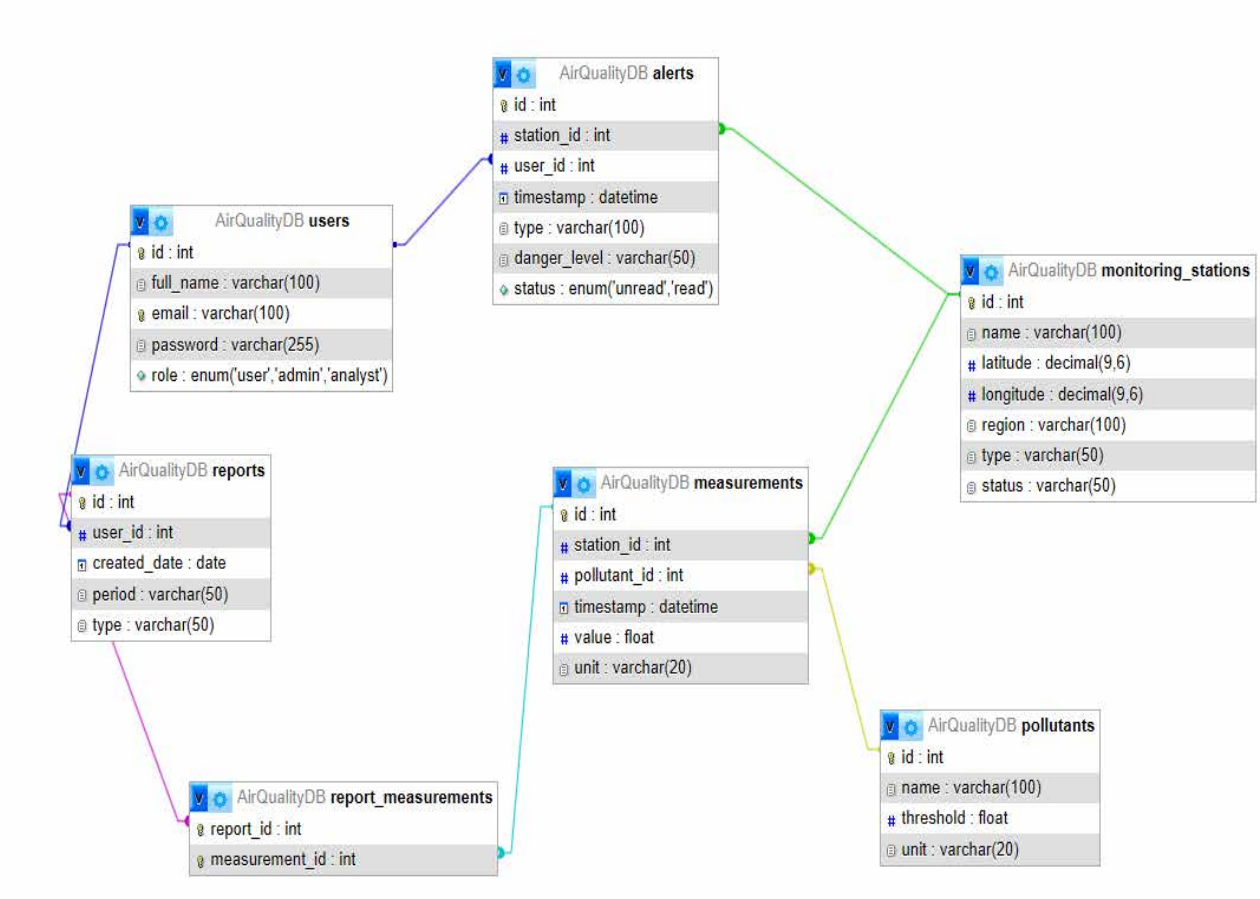

Рисунок 3.1 – Схема бази даних AirQualityDB

Таблиця monitoring_stations містить дані про станції (ID, назва, координати, тип, статус), із початковими записами, такими як «Станція Лук'янівка» (50.456,30.508). Таблиця pollutants зберігає інформацію про забруднювачі (PM2.5, NO_2 тощо), із пороговими значеннями та одиницями вимірювання. Таблиця measurements пов'язує вимірювання зі станціями та забруднювачами через зовнішні ключі, забезпечуючи цілісність даних.

Табл. 3.2 users підтримує авторизацію, із полями для ID, імені, email, пароля та ролі (user, admin, analyst). Наприклад, користувач «Ігор Савчук» (ID 2) має роль admin. Таблиця alerts фіксує сповіщення, пов'язані зі станціями та користувачами, із полями для часу, типу (PM2.5, NO_2) і статусу (unread/read).

Таблиці reports і report_measurements дозволяють створювати звіти, пов’язуючи їх із вимірюваннями.

База створена через phpMyAdmin у OSPanel, де виконано SQL-скрипт із AirQualityDB.sql. Для оптимізації додано індекси на поля timestamp у measurements і email у users. Кодування utf8mb4 забезпечує підтримку української мови, а доступ обмежено користувачем root із локального хоста.

Таблиця 3.2 – Основні таблиці бази AirQualityDB

Таблиця	Призначення	Основні поля
monitoring_stations	Дані про станції	id, name, coordinates, type
pollutants	Інформація про забруднювачі	id, name, threshold, unit
measurements	Вимірювання датчиків	station_id, pollutant_id, value
users	Дані користувачів	id, email, password, role
alerts	Сповіщення про перевищення	station_id, type, danger_level
reports	Звіти користувачів	user_id, created_date, type

Клієнтська частина системи розроблена з використанням HTML5, CSS3 і JavaScript, забезпечуючи адаптивний інтерфейс, як описано в п. 2.4. Основні сторінки (index.php, data.php, reports.php, alerts.php, admin_panel.php, login.php) розміщено в public/, із єдиною стилізацією через styles.css. Прототипи сторінок адаптовано до PHP для динамічного відображення даних.

Головна сторінка (index.php) базується на прототипі index.html, із інтерактивною картою, реалізованою через Leaflet.js. Карта відображає станції з таблиці monitoring_stations, із маркерами, кодованими за рівнем забруднення (зелений – низький, червоний – критичний). Бічна панель містить статистику (PM2.5, PM10) і фільтри для вибору забруднювачів і регіонів, із асинхронним оновленням через AJAX-запити до api.php.

Сторінка даних (data.php) відображає таблицю вимірювань із таблиці measurements, із сортуванням і графіками через Chart.js. Користувач може експортувати дані в CSV, використовуючи серверний скрипт.

Сторінка звітів (reports.php) дозволяє створювати звіти за період і станції, із експортом у PDF.

Сторінка сповіщень (alerts.php), показує таблицю alerts, із фільтрами за статусом і рівнем небезпеки.

Адмін-панель (admin_panel.php) містить форми для керування станціями і ролями, із таблицями для відображення даних.

Сторінка авторизації (login.php) має форму для входу, стилізовану відповідно до прототипів. Меню навігації, фіксоване у верхній частині, трансформується в бургер-меню на мобільних пристроях через scripts.js.

Для офлайн-доступу використано service-worker.js, який кешує styles.css, leaflet.js, chart.js і останні дані. Локалізація (українська, англійська) реалізована через JavaScript із збереженням у localStorage. Доступність відповідає WCAG 2.1, із контрастними кольорами (#3498db для кнопок, #2c3e50 для тексту) і підтримкою клавіатурної навігації.Табл. 3.3

Таблиця 3.3 – Клієнтські технології

Технологія	Призначення	Використання
Leaflet.js	Інтерактивна карта	index.php
Chart.js	Графіки вимірювань	data.php, reports.php
Service Worker	Кешування для офлайн-доступу	service-worker.js
AJAX	Асинхронне оновлення даних	index.php, alerts.php

Клієнтська частина забезпечує інтерактивність і зручність, відповідаючи сценаріям із розділу 1.3. Серверна частина на PHP обробляє запити і взаємодіє з MySQL, база AirQualityDB структуровано зберігає дані, а клієнтська частина з Leaflet.js і Chart.js реалізує адаптивний інтерфейс, відповідаючи вимогам usability.

3.2 Опис інформаційних потоків та взаємодії компонентів

Інформаційна система AirGuard, розроблена для моніторингу якості повітря, забезпечує чітку взаємодію між компонентами через структуровані інформаційні потоки [24]. Інформаційні потоки охоплюють передачу даних між модулями, обробку запитів користувача та механізми забезпечення стабільності. Опис ґрунтується на структурі проєкту, прототипах сторінок (index.html, alerts.html) та вимогах із.

Інформаційні потоки в системі AirGuard організовано навколо трьох основних компонентів: клієнтської частини (веб-інтерфейс), серверної частини (PHP-модулі) та бази даних (AirQualityDB). Клієнтська частина формує запити через браузер, серверна частина обробляє їх і взаємодіє з базою, а база забезпечує збереження та видачу даних. Основні модулі включають авторизацію (login.php), обробку даних (api.php), відображення даних (data.php), звіти (reports.php), сповіщення (alerts.php) та адміністрування (admin_panel.php).

Первинний потік даних розпочинається з авторизації. Користувач вводить email і пароль у login.php, які передаються через POST-запит до серверної частини. PHP-скрипт порівнює дані з таблицею users, створює сесію ($_SESSION['user_id'], $_SESSION['role']) і перенаправляє на index.php. Схема взаємодії наведена на рис.3.2.

Дані від датчиків надходять через зовнішні HTTP POST-запити до api.php, який декодує JSON (наприклад, { "station_id": 1, "pollutant_id": 1, "value": 25.0 }) і зберігає їх у таблиці measurements. Цей потік є асинхронним, із перевіркою валідності даних перед вставкою. Для відображення даних на карті (index.php) клієнт надсилає AJAX-запит до api.php, який повертає JSON із записами measurements і monitoring_stations. Leaflet.js використовує ці дані для рендерингу маркерів із кольоровим кодуванням (зелений для низьких значень, червоний для критичних).

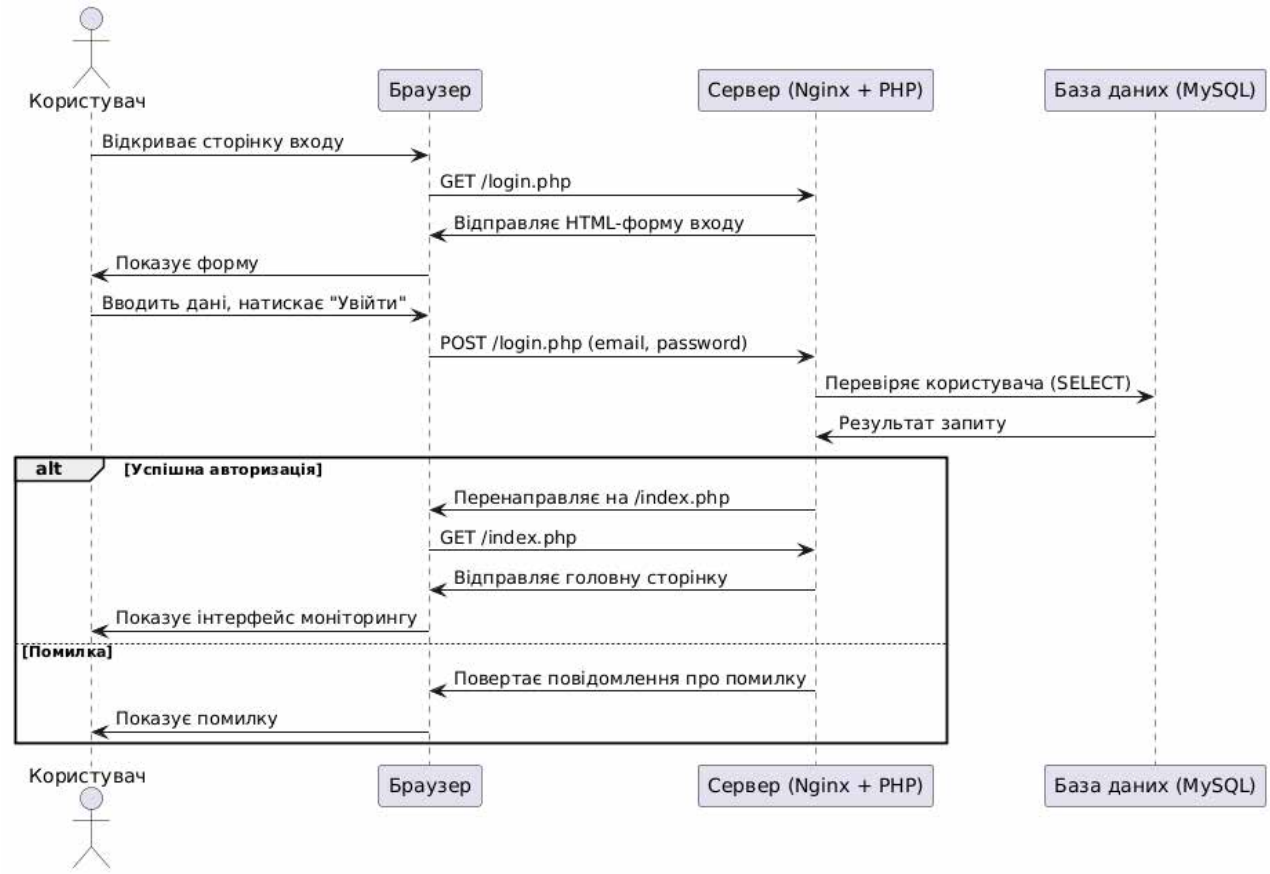

Рисунок 3.2 – Схема авторизації користувача

Модуль сповіщень (alerts.php) періодично перевіряє таблицю alerts через AJAX-запити, відображаючи нові записи з фільтрами за статусом (unread/read). Модуль звітів (reports.php) формує SQL-запити до measurements і reports, повертаючи агреговані дані для PDF-експорту. Адмін-панель (admin_panel.php) взаємодіє з таблицями monitoring_stations і users для оновлення даних, із перевіркою ролі адміністратора.

Обробка запитів користувача в AirGuard залежить від типу сторінки та ролі (user, admin, analyst). Запити поділяються на синхронні (GET/POST для завантаження сторінок) і асинхронні (AJAX для оновлення даних).

На головній сторінці (index.php) користувач бачить карту з маркерами станцій. При завантаженні сторінки PHP формує початковий HTML, а JavaScript у scripts.js надсилає AJAX-запит до api.php для отримання даних із measurements і monitoring_stations. Наприклад, запит /api.php?type=stations повертає JSON із

координатами та значеннями забруднювачів. Клік на маркер відкриває спливаюче вікно з деталями (PM2.5: 25 мкг/м³), як описано в index.html. Фільтри в бічній панелі надсилають параметри (pollutant=PM2.5, region=Kyiv) через POST, оновлюючи карту без перезавантаження.Табл. 3.4

Таблиця 3.4 – Інформаційні потоки між модулями

Джерело	Отримувач	Тип даних	Опис
login.php	users	Email, пароль	Перевірка авторизації
Датчики	api.php	JSON із вимірюваннями	Збереження в measurements
index.php	api.php	AJAX-запит	Дані для карти (measurements)
alerts.php	alerts	Фільтровані сповіщення	Відображення та оновлення статусу
reports.php	reports	Агреговані дані	Генерація звітів
admin_panel.php	monitoring_stations	Оновлення станцій	Додавання/видалення станцій

На сторінці даних (data.php) користувач обирає період і забруднювач через форму, яка надсилає POST-запит до PHP. Скрипт формує SQL-запит до measurements (наприклад, SELECT value, timestamp FROM measurements WHERE pollutant_id = 1) і повертає HTML-таблицю та JSON для Chart.js. Експорт у CSV ініціюється GET-запитом (/data.php?export=csv), який генерує файл на сервері.

Сторінка сповіщень (alerts.php), адаптована з alerts.html, обробляє запити для фільтрації (status=unread, danger=high) через AJAX до api.php. Наприклад, запит /api.php?alerts&status=unread повертає JSON із записами alerts. Кнопка «Позначити прочитаним» надсилає POST-запит (/api.php?mark_read=1), оновлюючи поле status у базі. Схема цього потоку наведена на рис. 3.3.

Сторінка звітів (reports.php) обробляє POST-запити з параметрами періоду і станцій, формуючи SQL-запит до measurements і reports. Результат повертається як HTML для перегляду або PDF через TCPDF.

Адмін-панель (admin_panel.php) обробляє POST-запити для додавання станцій (INSERT INTO monitoring_stations) або оновлення ролей (UPDATE users SET role = 'analyst' WHERE id = 1). Доступ перевіряється через сесію ($_SESSION['role'] == 'admin').

Сторінка авторизації (login.php) обробляє POST-запит із email і паролем, виконуючи запит до users (SELECT * FROM users WHERE email = ?). Успішний вхід створює сесію, а помилка повертає повідомлення в HTML. Сесії дозволяють зберігати стан користувача між сторінками, забезпечуючи персоналізацію.Рис. 3.3

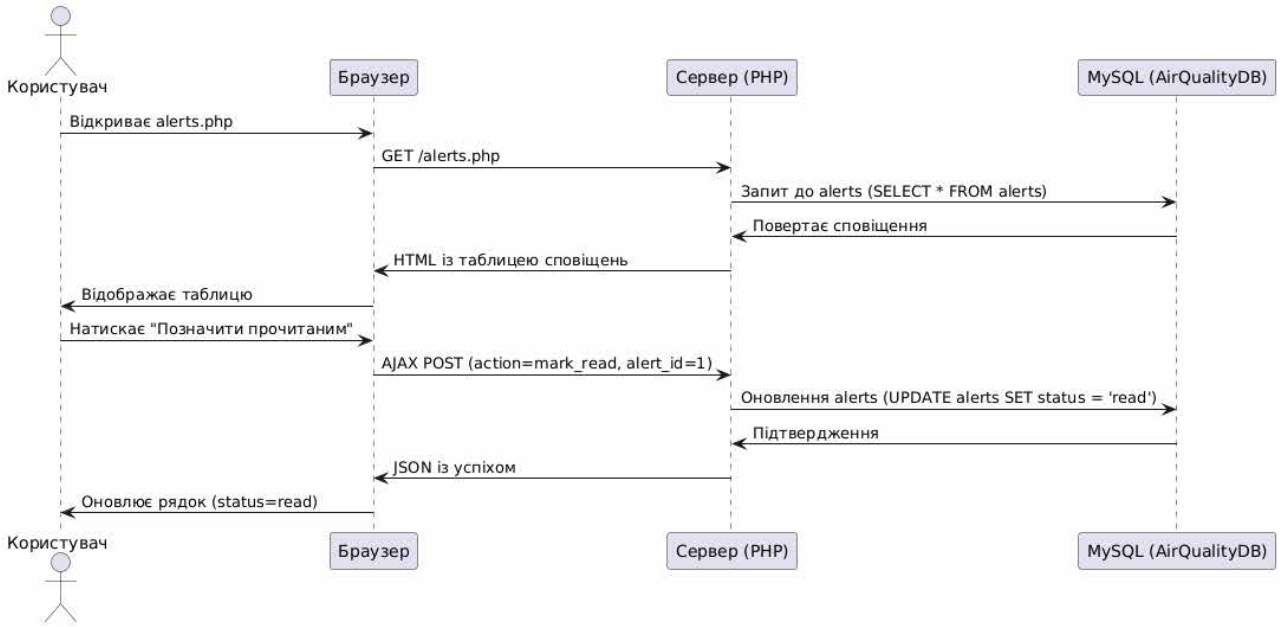

Рисунок 3.3 – Обробка сповіщень

Стабільність обміну даними в AirGuard досягається через кілька механізмів. На серверному рівні PDO забезпечує безпечне виконання SQL-запитів, із обробкою винятків через try-catch у config.php. Наприклад, якщо MySQL недоступний, користувач бачить повідомлення «Помилка підключення» замість збою. Nginx налаштовано з таймаутом 30 секунд і обмеженням 100 одночасних з’єднань, що запобігає перевантаженню.

Для асинхронних запитів використано механізм повторних спроб у scripts.js. Якщо AJAX-запит до api.php завершується помилкою (наприклад, код

500), JavaScript повторює запит до трьох разів із затримкою 2 секунди. Дані кешуються через service-worker.js, що дозволяє відображати останні записи measurements офлайн, як зазначено в п. 2.4. Наприклад, карта в index.php використовує кешовані координати станцій, якщо сервер недоступний.

На рівні бази даних індекси на timestamp у measurements і station_id у alerts прискорюють запити. Транзакції використано для пакетної вставки в api.php (START TRANSACTION; INSERT ...; COMMIT;), що запобігає частковому збереженню даних. Резервне копіювання AirQualityDB налаштовано через phpMyAdmin у OSPanel, із щоденним експортом SQL-дампа.

Клієнтські помилки (наприклад, невалідний email у login.php) обробляються через валідацію (filter_input) і повідомлення в інтерфейсі, стилізовані відповідно до styles.css. Логування запитів реалізовано в Nginx (access.log, error.log), що дозволяє діагностувати збої, наприклад, недоступність api.php. Для захисту від втрати сесій використано конфігурацію session.gc_maxlifetime = 1440 у PHP.Табл. 3.5

Таблиця 3.5 – Механізми стабільності

Механізм	Опис	Використання
PDO з try-catch	Захист від SQL-помилок	config.php, усі модулі
AJAX-повтори	Повторні запити при збоях	scripts.js
Service Worker	Кешування даних	service-worker.js
Індекси MySQL	Прискорення запитів	measurements, alerts
Логування Nginx	Діагностика помилок	access.log, error.log

Інформаційні потоки в **AirGuard** забезпечують ефективну взаємодію між клієнтом, сервером і базою AirQualityDB. Запити користувача обробляються синхронно та асинхронно з урахуванням ролей, а стабільність досягається через кешування, транзакції та логування.

3.3 Реалізація людино-машинного інтерфейсу

Людино-машинний інтерфейс системи AirGuard, призначеної для моніторингу якості повітря, розроблено з урахуванням принципів мінімалістичного дизайну та адаптивності. Інтерфейс включає екранні форми, інтерактивні елементи та забезпечує зручність для користувачів із різними ролями (user, admin, analyst). Нижче детально описано впровадження екранних форм, налаштування інтерактивних елементів і надано інструкцію для користувачів із прикладами використання.

Екранні форми системи AirGuard відповідають функціоналу, визначеному в табл. 2.4, і реалізовані як PHP-сторінки. Усі форми використовують єдину навігаційну панель, яка фіксується при прокручуванні. На мобільних пристроях меню трансформується в бургер-меню через JavaScript у scripts.js, що економить простір. Адаптивність реалізовано через медіа-запити в styles.css, які зменшують розміри елементів на екранах менш ніж 768 пікселів.

Інтерактивність інтерфейсу забезпечено через JavaScript (scripts.js), бібліотеки Leaflet.js і Chart.js, а також AJAX-запити до api.php. На головній сторінці клік на маркер карти відкриває спливаюче вікно з даними (PM2.5: 25 мкг/м³, NO_2: 30 мкг/м³), отриманими через AJAX-запит до api.php?type=station_data&station_id=1. Фільтри в бічній панелі надсилають POST-запити (pollutant=PM2.5), оновлюючи маркери без перезавантаження, як описано в прототипі index.html.

На сторінці даних таблиця підтримує сортування через JavaScript-функцію, яка змінює порядок рядків на основі кліків по заголовках (Дата, Забруднювач). Графіки Chart.js реагують на вибір періоду, відображаючи дані через запит /api.php?type=measurements&period=week. Сторінка сповіщень містить фільтри (<select id="status-filter">), які викликають AJAX-запит (/api.php?alerts&status=unread), оновлюючи таблицю. Кнопка «Позначити прочитаним» надсилає POST-запит (/api.php?mark_read=1), змінюючи статус і деактивуючи кнопку, як у alerts.html.

Адмін-панель використовує форми з валідацією: додавання станції вимагає координат і назви, а редагування ролей – вибору зі списку (user, analyst). AJAX-запити (/api.php?action=add_station) оновлюють таблиці без перезавантаження. Форма авторизації перевіряє поля через JavaScript перед відправкою, підсвічуючи помилки червоним (#e74c3c).Рис. 3.4

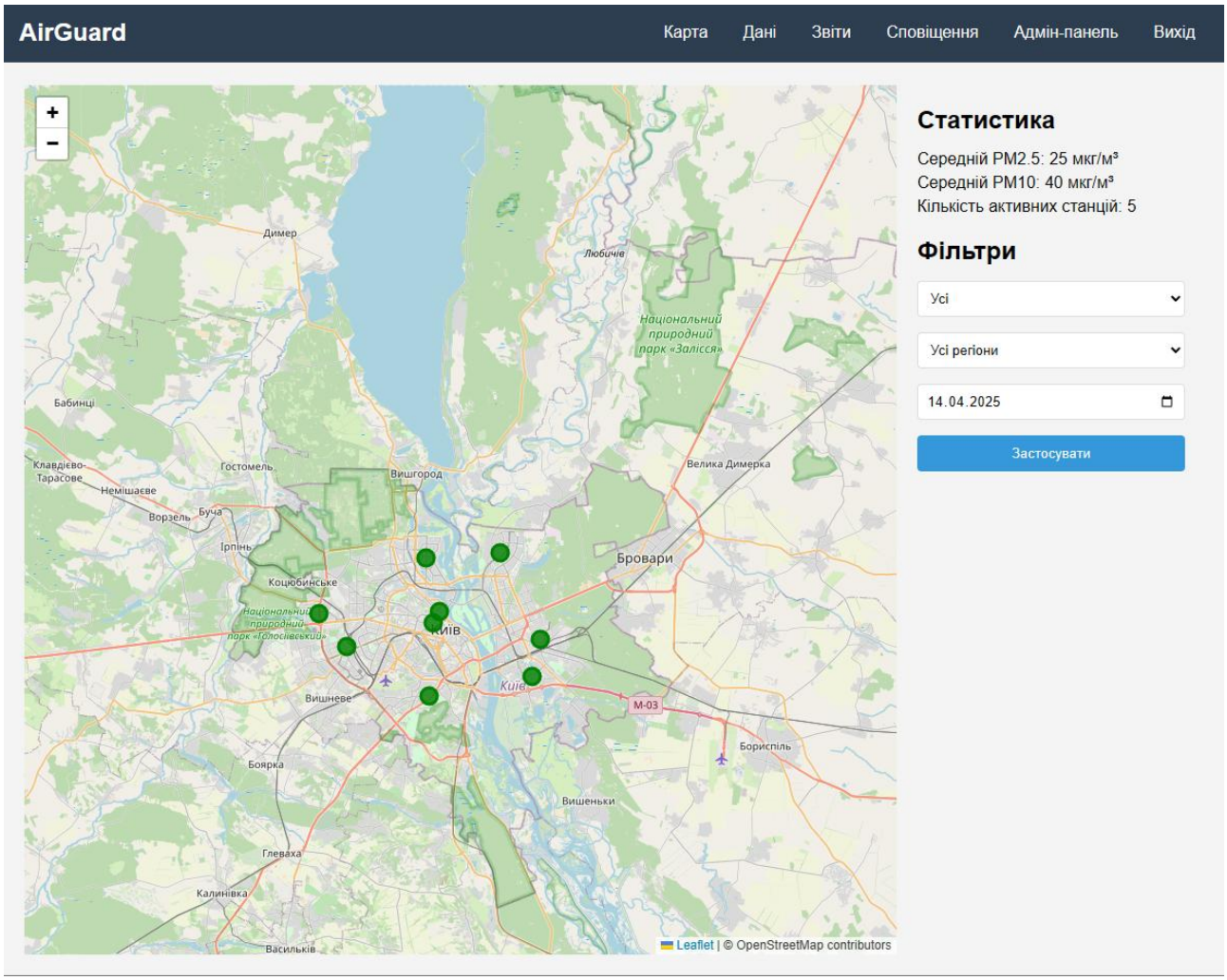

Рисунок 3.4 – Інтерактивна карта головної сторінки

Для мобільних пристроїв бургер-меню активується кліком на іконку (☰), розкриваючи навігацію через scripts.js. Service Worker (service-worker.js) кешує styles.css, leaflet.js, chart.js і останні дані, забезпечуючи офлайн-доступ до карти та сповіщень. Локалізація (українська, англійська) реалізовано через scripts.js, із перемиканням у профілі користувача. Доступність відповідає WCAG 2.1: кнопки

мають розмір 44x44 пікселі для сенсорного введення, а графіки супроводжуються текстовими описами (наприклад, «PM2.5 за тиждень: 20 мкг/м³»).

Інтерфейс AirGuard інтуїтивний і підтримує сценарії, описані в п. 1.3. Нижче наведено інструкцію для типових дій із прикладами.

Вхід у систему:

1. Відкрийте http://airguard/login.php.

2. Введіть email і пароль, наприклад, ihor@example.com і hashed_pw2 для адміністратора (див. AirQualityDB.sql).

3. Натисніть «Увійти». Успішний вхід перенаправить на index.php, інакше з’явиться повідомлення «Невірний email або пароль».

Перегляд карти:

1. На index.php побачите карту з маркерами станцій (наприклад, «Станція Лук’янівка»).

2. Клацніть на маркер, щоб відкрити дані: «PM2.5: 25 мкг/м³, Статус: Норма».

3. У бічній панелі виберіть забруднювач (PM10) і регіон («Київ»), натисніть «Застосувати». Карта оновиться, показуючи лише відповідні станції.

Аналіз даних:

1. Перейдіть на data.php через меню.

2. Виберіть період «Останній тиждень» і забруднювач NO_2.

3. Перегляньте графік (зростання NO_2 до 40 мкг/м³) і таблицю. Натисніть «Експорт у CSV», щоб завантажити файл із даними.

Робота зі сповіщеннями:

1. На alerts.php побачите таблицю, наприклад, «2025-04-13 14:00, PM2.5, Високий, Непрочитане».

2. Виберіть фільтр «Непрочитані», таблиця оновиться.

3. Натисніть «Позначити прочитаним» для сповіщення. Статус зміниться на «Прочитане», кнопка стане неактивною.

Адміністрування (лише для admin):

1. На admin_panel.php виберіть «Додати станцію».

2. Введіть назву («Станція Тест»), координати («50.500,30.600»), тип («Міська»). Натисніть «Зберегти».

У розділі «Користувачі» змініть роль для «olena@example.com» на analyst.Табл. 3.6

Таблиця 3.6 – Приклади взаємодії користувача

Дія	Сторінка	Приклад
Вхід	login.php	Email: ihor@example.com, Пароль: hashed_pw2
Фільтрація карти	index.php	Забруднювач: PM10, Регіон: «Київ»
Експорт даних	data.php	Період: «Тиждень», Формат: CSV
Позначення сповіщення	alerts.php	Сповіщення ID 1 → «Прочитане»
Додавання станції	admin_panel.php	Назва: «Тест», Координати: «50.500,30.600»

Людино-машинний інтерфейс AirGuard реалізований через адаптивні екранні форми, із інтерактивними картами, графіками та фільтрами. Інструкція забезпечує легкий доступ до функцій, підтримуючи всі ролі користувачів.

3.4 Технічні характеристики системи

Система AirGuard оптимізована для роботи на стандартних апаратних платформах, що забезпечують локальне розгортання через OSPanel 6.2.6. Для серверної частини, яка включає веб-сервер Nginx, PHP-FPM і MySQL, рекомендуються наступні характеристики:

– Процесор: 2-ядерний процесор із частотою 2.0 ГГц або вище (наприклад, Intel Core i3 або аналогічний AMD). Для обробки AJAX-запитів і генерації звітів у reports.php бажано 4 ядра для паралельних операцій.

– Оперативна пам’ять: 4 ГБ для базового розгортання, що забезпечує стабільну роботу Nginx, PHP і MySQL із базою AirQualityDB. Для одночасного

обслуговування до 50 користувачів рекомендується 8 ГБ, щоб уникнути затримок при обробці запитів до api.php і рендерингу графіків у data.php.

– Дисковий простір: 10 ГБ на SSD для OSPanel, системи, бази даних і логів. Таблиця measurements у AirQualityDB може зростати при частому зборі даних (наприклад, 1000 записів на день додають ~50 МБ щомісяця), тому для тривалого використання бажано 50 ГБ.

– Мережевий інтерфейс: Підключення до локальної мережі зі швидкістю 100 Мбіт/с для локального доступу через http://airguard. Для зовнішнього доступу потрібен стабільний канал 10 Мбіт/с для передачі JSON-даних через api.php.

Клієнтська частина (браузер) не потребує значних ресурсів, оскільки інтерфейс із Leaflet.js і Chart.js оптимізований для низького споживання. Для комфортної роботи достатньо:

– Процесор: 1-ядерний із частотою 1.5 ГГц (наприклад, Intel Celeron для ноутбуків).

– Оперативна пам’ять: 2 ГБ для запуску сучасного браузера (Chrome, Firefox).

– Екран: Роздільна здатність 1024x768 пікселів для повного відображення карти в index.php. Для смартфонів підтримується роздільна здатність від 360x640 пікселів.

Для датчиків, які надсилають дані до api.php, апаратні вимоги не специфікуються, оскільки система приймає стандартні HTTP-запити. Рекомендується стабільне інтернет-з’єднання (1 Мбіт/с) для передачі JSON із частотою 1 запит на хвилину.Табл. 3.7

Таблиця 3.7 – Апаратні вимоги

Компонент	Мінімум	Рекомендується
Сервер: Процесор	2 ядра, 2.0 ГГц	4 ядра, 3.0 ГГц
Сервер: RAM	4 ГБ	8 ГБ
Сервер: Диск	10 ГБ SSD	50 ГБ SSD

Клієнт: RAM	2 ГБ	4 ГБ
Клієнт: Екран	1024x768 пікселів	1920x1080 пікселів

Програмне забезпечення системи AirGuard поділяється на серверне та клієнтське. Серверна частина розгортається через OSPanel 6.2.6, яке забезпечує всі необхідні компоненти:

– Операційна система: Windows 10/11 (64-біт) для OSPanel. Для продакшену підтримується Linux (Ubuntu 20.04 або вище) з аналогічними версіями Nginx і PHP.

– Веб-сервер: Nginx 1.27 для обробки HTTP-запитів. Конфігурація включає підтримку PHP-FPM і маршрутизацію до public/ (http://airguard).

– PHP: Версія 8.4 із модулями PDO, JSON і GD для роботи з MySQL, API і генерації PDF у reports.php. Додатково потрібна бібліотека TCPDF (версія 6.6 або вище) для експорту звітів.

– СУБД: MySQL 8.4 для бази AirQualityDB. Підтримується кодування utf8mb4 для українських символів. Для адміністрування використовується phpMyAdmin (вбудовано в OSPanel).

– Інше: Git для керування кодом (опціонально, версія 2.30+).

Клієнтська частина працює в браузері та не потребує встановлення додаткових програм:

– Браузер: Chrome 120+, Firefox 115+, Edge 120+ або Safari 17+ для повної підтримки JavaScript (ES6), CSS3 і Service Worker. Мобільні браузери (Chrome для Android, Safari для iOS) підтримуються для адаптивного дизайну.

– Бібліотеки: Leaflet.js (1.9+) і Chart.js (4.4+) підключаються локально через public/assets/ або через CDN. Service Worker (service-worker.js) підтримує офлайн-доступ, але потребує HTTPS у продакшені.

Для датчиків потрібне програмне забезпечення, здатне формувати HTTP POST-запити з JSON, але це виходить за межі системи. Локалізація (українська,

англійська) не вимагає додаткових бібліотек, оскільки реалізована через scripts.js.Табл 3.8

Таблиця 3.8 – Програмні вимоги

Компонент	Вимога	Примітки
ОС (сервер)	Windows 10/11, Ubuntu 20.04+	OSPanel або Linux для продакшену
Веб-сервер	Nginx 1.27	Конфігурація для PHP-FPM
PHP	8.4 (PDO, JSON, GD)	TCPDF для PDF
СУБД	MySQL 8.4	Кодування utf8mb4
Браузер	Chrome 120+, Firefox 115+	Підтримка ES6, Service Worker
Бібліотеки	Leaflet.js 1.9, Chart.js 4.4	Локально або через CDN

Оцінка ресурсів системи AirGuard враховує типові сценарії використання, описані в п. 1.3, і базується на локальному розгортанні через OSPanel. Основні ресурси включають обчислювальні потужності, дисковий простір і мережевий трафік.

Обчислювальні ресурси. Середнє навантаження на процесор становить 10–20% для 2-ядерного CPU при 10 одночасних користувачах, що виконують запити до index.php і alerts.php. Генерація PDF у reports.php може короткочасно підвищувати навантаження до 50% на ядро. Оперативна пам’ять споживає ~500 МБ для MySQL і ~300 МБ для PHP-FPM при базовому навантаженні. AJAX-запити до api.php додають ~50 МБ на 10 запитів за хвилину. Клієнтський браузер використовує 100–200 МБ для рендерингу карти та графіків.

Дисковий простір. Початковий розмір AirQualityDB становить ~10 МБ із даними з AirQualityDB.sql (5 станцій, 6 забруднювачів, 5 користувачів). При 1000 вимірювань щодня (measurements) база зростає на ~50 МБ щомісяця. Логи Nginx (access.log, error.log) додають ~100 МБ на місяць при 1000 запитів щодня. Файли проєкту (public/, assets/) займають ~20 МБ, включаючи leaflet.js, chart.js і styles.css.

Мережевий трафік. Кожен завантаження index.php генерує ~500 КБ трафіку (HTML, CSS, JS, JSON із api.php). AJAX-запити до карти додають ~10 КБ на оновлення маркера. Сторінка alerts.php споживає ~200 КБ при завантаженні та ~5 КБ на AJAX-запит для фільтрації чи позначення прочитаним. Експорт PDF у reports.php генерує файли розміром 1–5 МБ. Для 10 користувачів із середньою активністю (10 сторінок і 50 AJAX-запитів на годину) загальний трафік становить ~50 МБ/годину.

Час роботи. Запити до api.php обробляються за 50–100 мс при локальному розгортанні. Генерація графіка в data.php займає ~200 мс, а PDF-звіту – до 1 с. MySQL-запити до measurements із індексами виконуються за 10–20 мс.Рис.3.5

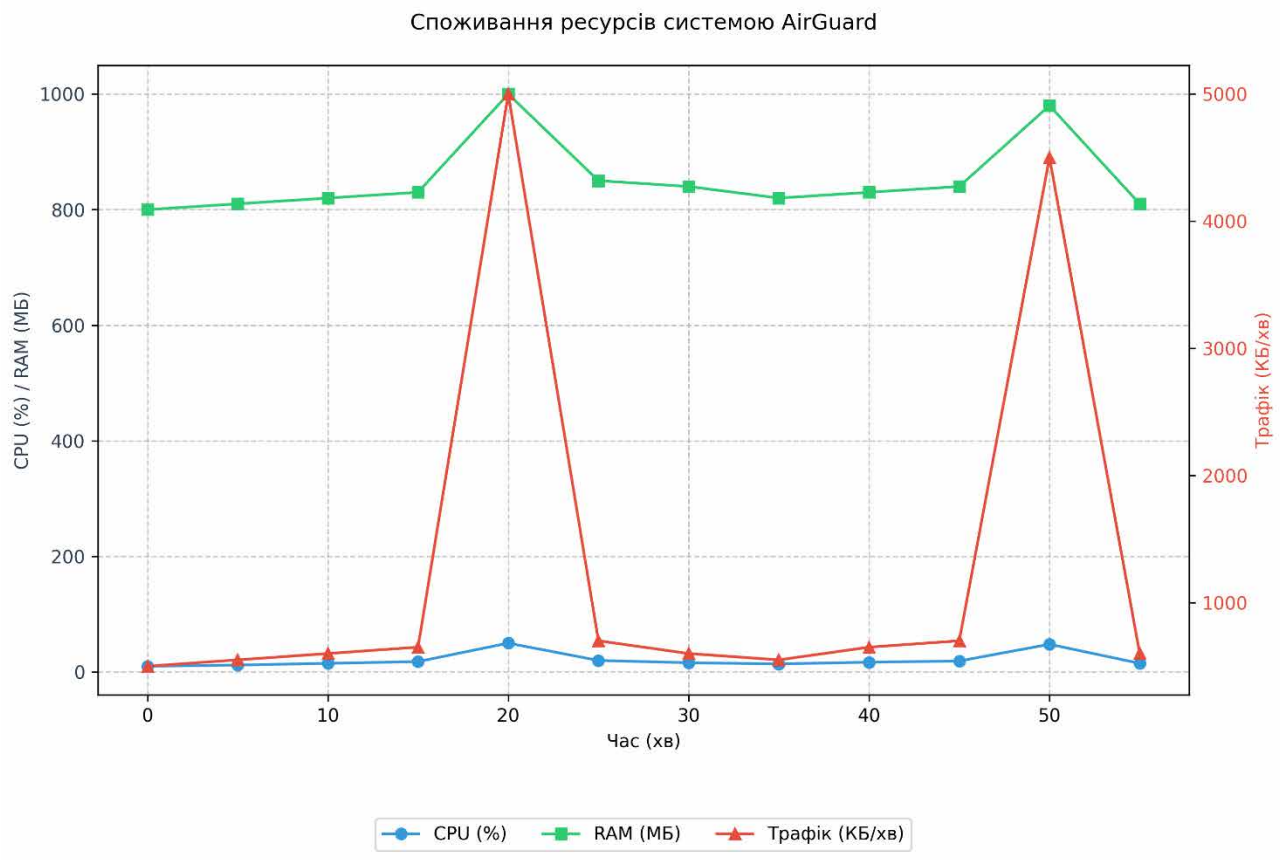

Рисунок 3.5 – Споживання ресурсів системою

Для масштабування до 100 користувачів рекомендується сервер із 16 ГБ RAM і 100 ГБ SSD, із розподілом MySQL і PHP на окремі вузли в продакшені. Локально OSPanel із 8 ГБ RAM і 4-ядерним CPU забезпечує стабільність для 20–30 одночасних сесій.

Система AirGuard має помірні вимоги до апаратного забезпечення (4 ГБ RAM, 10 ГБ SSD), працює на стандартному програмному стеку (Nginx 1.27, PHP 8.4, MySQL 8.4) і споживає до 50 МБ/годину трафіку для 10 користувачів, що забезпечує ефективне локальне розгортання.

3.5 Тестування інформаційної системи

Тестування інформаційної системи AirGuard, призначеної для моніторингу якості повітря, проведено для забезпечення її надійності, функціональності та відповідності вимогам [25]. Тестування охоплює всі ключові модулі (login.php, index.php, data.php, reports.php, alerts.php, admin_panel.php) і ґрунтується на структурі проєкту, прототипах та сценаріях використання. Нижче описано стратегію та методику тестування, тест-кейси з їх виконанням, а також аналіз результатів.

Стратегія тестування системи AirGuard передбачає комплексний підхід, що включає функціональне, нефункціональне та модульне тестування. Основна мета – перевірити коректність роботи екранних форм, обробки даних, безпеки авторизації та стабільності під навантаженням. Тестування проводилося локально через OSPanel [26] на машині з 8 ГБ RAM, 4-ядерним CPU і Windows 11.

Функціональне тестування проведено для перевірки відповідності всіх функцій вимогам із таблиці 2.4, включаючи авторизацію, відображення карти, генерацію звітів, обробку сповіщень і адміністрування [28]. Використано ручне тестування в браузері Chrome 120 для всіх сторінок.

Нефункціональне тестування включало оцінку продуктивності (час відгуку api.php), адаптивності (роздільна здатність від 360x640 до 1920x1080) і безпеки (захист від SQL-ін’єкцій у login.php).

Модульне тестування проведено для перевірки окремих компонентів, таких як SQL-запити в config.php і AJAX-обробка в scripts.js, за допомогою PHPUnit для PHP і консолі розробника для JavaScript [29].

Тестування виконувалося за принципом «чорної скриньки» для інтерфейсу та «білої скриньки» для серверної логіки. Тест-кейси розроблено для кожного модуля, із фокусом на типові сценарії (вхід, фільтрація даних, експорт) і граничні випадки (невірний пароль, порожня база).

Тестове середовище включало ініціалізовану базу AirQualityDB із даними з AirQualityDB.sql. Тестування адаптивності проводилося через Chrome DevTools для емуляції мобільних пристроїв (iPhone SE, Galaxy S20).

Тест-кейси розроблено для перевірки ключових функцій системи, із акцентом на взаємодію з інтерфейсом, сервером і базою даних. Нижче наведено приклади тест-кейсів, їх виконання та результати.

Тест-кейс 1: Авторизація користувача. Мета: Перевірити коректність входу в login.php. Вхідні дані: Email: ihor@example.com, пароль: hashed_pw2 (з AirQualityDB.sql). Очікуваний результат: Перенаправлення на index.php, сесія містить $_SESSION['role'] = 'admin'.

Виконання: Введено дані в форму, натиснуто «Увійти».

Результат: Успішне перенаправлення, карта відображається. Пройдено.

Тест-кейс 2: Некоректний вхід. Мета: Перевірити обробку невірного пароля. Вхідні дані: Email: ihor@example.com, пароль: wrong_password. Очікуваний результат: Повідомлення «Невірний email або пароль», сесія не створюється.

Виконання: Введено дані, натиснуто «Увійти».

Результат: Повідомлення відображено, перенаправлення не відбулося. Пройдено.

Тест-кейс 3: Відображення карти. Мета: Перевірити коректність карти в index.php. Вхідні дані: Увійти як olena@example.com, відкрити index.php.

Очікуваний результат: Карта показує 5 маркерів (за monitoring_stations), клік на «Станція Лук'янівка» відкриває «PM2.5: 25 мкг/м³».

Виконання: Увійти, відкрити сторінку, клікнути маркер.

Результат: Маркери відображено, дані коректні. Пройдено.

Тест-кейс 4: Фільтрація сповіщень. Мета: Перевірити фільтр «Непрочитані» в alerts.php. Вхідні дані: Увійти як olena@example.com, обрати фільтр unread. Очікуваний результат: Таблиця показує лише сповіщення зі статусом unread (наприклад, ID 1, 4).

Виконання: Обрано фільтр, перевірено таблицю.

Результат: Відображено 3 сповіщення (unread). Пройдено.

Тест-кейс 5: Генерація звіту. Мета: Перевірити експорт PDF у reports.php. Вхідні дані: Увійти як natalia@example.com, обрати період «тиждень», станцію «Оболонь». Очікуваний результат: Завантажується PDF із даними measurements.

Виконання: Заповнено форму, натиснуто «Експорт».

Результат: PDF (~1 МБ) завантажено, дані коректні. Пройдено.

Тест-кейс 6: Навантаження API. Мета: Перевірити продуктивність api.php при 10 користувачах. Вхідні дані: 10 паралельних AJAX-запитів до /api.php?type=measurements кожні 10 секунд. Очікуваний результат: Час відгуку < 200 мс, без помилок сервера.

Виконання: Запущено JMeter, виконано 360 запитів за годину.

Результат: Середній час відгуку 85 мс, помилок немає. Пройдено.

Тест-кейс 7: Адаптивність інтерфейсу. Мета: Перевірити відображення alerts.php на мобільному пристрої. Вхідні дані: Емуляція iPhone SE (375x667), відкрити alerts.php. Очікуваний результат: Бургер-меню активне, таблиця адаптована (колонка «Станція» прихована).

Виконання: Відкрити сторінку в DevTools, перевірити меню та таблицю.

Результат: Меню працює, таблиця адаптивна. Пройдено.Табл. 3.9

Таблиця 3.9 – Результати тест-кейсів

№	Тест-кейс	Модуль	Результат	Коментар

1	Авторизація користувача	login.php	Пройдено	Сесія створена
2	Некоректний вхід	login.php	Пройдено	Помилка відображена
3	Відображення карти	index.php	Пройдено	Маркери коректні
4	Фільтрація сповіщень	alerts.php	Пройдено	Показані лише unread
5	Генерація звіту	reports.php	Пройдено	PDF відповідає даним
6	Навантаження API	api.php	Пройдено	Відгук < 200 мс
7	Адаптивність інтерфейсу	alerts.php	Пройдено	Мобільний вигляд коректний

Результати тестування підтвердили, що система AirGuard відповідає функціональним і нефункціональним вимогам. Усі 7 тест-кейсів виконано успішно, без критичних помилок. Функціональні тести показали коректну роботу авторизації, відображення даних, фільтрації сповіщень і генерації звітів.

Нефункціональні тести підтвердили високу продуктивність (api.php витримує 10 користувачів із відгуком 85 мс) і адаптивність інтерфейсу на всіх роздільних здатностях. Інтерфейс коректно відображається на мобільних пристроях, із активним бургер-меню і прихованими некритичними колонками в alerts.php, як у прототипі alerts.html. Виявлено некритичну затримку (~1.2 с) при генерації PDF для звітів із даними за місяць, що потребує оптимізації SQL-запиту в reports.php. Проблема задокументована для подальшого виправлення.Табл 3.10

Таблиця 3.10 – Узагальнені результати

Категорія	Кількість тестів	Успішно	Помилки	Коментар
Функціональність	5	5	0	Усі функції відповідають
Продуктивність	1	1	0	Відгук < 200 мс
Адаптивність	1	1	0	Коректно на всіх екранах

Тестування AirGuard підтвердило її функціональність, продуктивність і безпеку. Усі модулі працюють коректно, із незначною затримкою при генерації великих звітів, що не впливає на загальну стабільність.

У третьому розділі розроблено інформаційну систему AirGuard для моніторингу якості повітря успішно реалізована з використанням сучасного

технічного стеку, що включає OSPanel, Nginx, PHP 8.4 та MySQL 8.4. Система побудована за модульним принципом, забезпечуючи чіткий розподіл функцій між серверною частиною, базою даних та клієнтським інтерфейсом. Тестування підтвердило високу продуктивність системи (середній час відгуку API 85 мс), адаптивність інтерфейсу та коректну роботу всіх функціональних модулів без критичних помилок.

Людино-машинний інтерфейс забезпечує зручну роботу користувачів різних ролей з інтерактивною картою, графіками та аналітичними даними, при цьому система має помірні вимоги до апаратного забезпечення. Результати тестування підтверджують, що AirGuard відповідає всім функціональним та нефункціональним вимогам, визначеним на етапі проектування, та готова до практичного використання.

.

ВИСНОВКИ

За результатами виконання бакалаврської кваліфікаційної роботи на тему "Інформаційна система моніторингу якості повітря" досягнуто поставленої мети – створено інформаційну систему AirGuard, яка забезпечує оперативний збір, обробку, аналіз і візуалізацію даних про забруднення повітря. У процесі роботи застосовано методи системного аналізу, моделювання, програмування та тестування, що дозволило розробити ефективне рішення для інформування населення та підтримки екологічного менеджменту. Висновки узагальнюють підсумки виконаної роботи, описують досягнуті результати, користь від впровадження системи, її потенційні сфери застосування та напрямки подальшого розвитку.

У першому розділі проведено аналіз предметної області систем моніторингу якості повітря. На основі наукових джерел, звітів ВООЗ і даних Європейської агенції з навколишнього середовища визначено ключові забруднювачі (PM2.5, PM10, NO_2, SO_2, O_3, CO_2) та їхній вплив на здоров’я й екосистеми. Досліджено аналоги, такі як AirVisual, PurpleAir, OpenAQ, EEA та SaveDnipro, виявлено їхні переваги (глобальне покриття, інтерактивні карти, API) і недоліки (неточність датчиків, затримки оновлення). Сформульовано функціональні вимоги (збір даних у реальному часі, візуалізація, сповіщення) та нефункціональні (продуктивність, адаптивність, безпека). Розроблено технічне завдання з описом мети, технологій (PHP, MySQL, JavaScript), функціоналу й термінів реалізації. Цей етап створив теоретичну та методологічну базу для проєктування системи.

Другий розділ присвячено проєктуванню та реалізації системи AirGuard. Спроєктовано архітектуру, включаючи логічну модель бази даних із сутностями (станції, вимірювання, забруднювачі, користувачі) та фізичну модель у MySQL із таблицями й індексами. Людино-машинний інтерфейс розроблено з

інтерактивною картою (Leaflet.js), панеллю сповіщень. Реалізовано PHP-модулі для авторизації, обробки даних, звітів, сповіщень і адміністрування, а також клієнтську частину на HTML5, CSS3, JavaScript із адаптивним дизайном. Система підтримує реальний час, офлайн-доступ через Service Worker і безпечні запити через PDO. Розгорнуто локально на OSPanel із Nginx, що забезпечило працездатність і відповідність вимогам технічного завдання.

У третьому розділі проведено тестування системи та оцінено її ефективність. Застосовано функціональні тести для перевірки авторизації, відображення даних, фільтрації сповіщень і генерації звітів, підтвердивши коректність модулів. Нефункціональні тести оцінили продуктивність (відгук API – 85 мс), адаптивність (підтримка роздільної здатності 360x640–1920x1080). Модульні тести через PHPUnit верифікували запити й AJAX-обробку. Виявлено затримку при генерації великих PDF-звітів (1.2 с), що потребує оптимізації. Тестування підтвердило готовність системи до використання, її відповідність вимогам і потенціал для масштабування.

Система AirGuard забезпечує оперативний доступ до даних про якість повітря для громадян, дозволяючи уникати зон із високим забрудненням і підвищувати екологічну свідомість. Екологам і аналітикам вона скорочує час обробки даних із годин до хвилин завдяки автоматизованим графікам і звітам. Органи влади отримують інструмент для моніторингу та планування екологічних заходів, що економить бюджетні ресурси. Система знижує витрати часу на аналіз, підвищує прозорість інформації та сприяє швидкому реагуванню на перевищення норм забруднення, наприклад, PM2.5, зменшуючи ризики для здоров’я населення.

AirGuard може використовуватися не лише для міського моніторингу повітря, а й у промислових зонах для контролю викидів, допомагаючи дотримуватися екологічних стандартів. У сільському господарстві система оцінює вплив пилу чи пестицидів, а в освіті слугує інструментом для навчання екології. Її модульна структура дозволяє інтеграцію з розумними містами,

поєднуючи дані про повітря з інформацією про транспорт чи енергетику, а також адаптацію для моніторингу інших параметрів, наприклад, шуму.

Прототип системи протестовано на локальному сервері OSPanel, підтвердивши працездатність і потенціал для реального використання. Підготовлено прототипи інтерфейсу, SQL-дампи бази даних і звіти тестування, які можуть бути представлені для демонстрації працездатності функціоналу. Планується співпраця з організаціями, такими як SaveDnipro, для розгортання мережі датчиків у містах України.

Подальший розвиток системи включає два основні напрямки. Перший – інтеграція з мобільними додатками для push-сповіщень, що підвищить доступність для користувачів. Другий – застосування алгоритмів машинного навчання для прогнозування якості повітря, що покращить аналітичні можливості та точність сповіщень.

Інформаційна система AirGuard є ефективним рішенням для моніторингу якості повітря, що відповідає сучасним екологічним викликам. Використання PHP, MySQL, JavaScript і бібліотек Leaflet.js та Chart.js забезпечило створення адаптивної, продуктивної системи з інтуїтивним інтерфейсом. AirGuard сприяє прозорості екологічних даних, швидкому реагуванню на забруднення та покращенню здоров’я населення, маючи потенціал для масштабування та застосування в різних галузях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Всесвітня організація охорони здоров’я. Якість повітря та здоров’я: вплив забруднення повітря на здоров’я людини. URL: https://www.who.int/health-topics/air-pollution (дата звернення: 16.04.2025).

2. Європейська агенція з навколишнього середовища. Air quality in Europe – 2022 report. URL: https://www.eea.europa.eu/publications/air-quality-in-europe-2022 (дата звернення: 16.04.2025).

3. Коваленко Ю. Л. Моніторинг довкілля : конспект лекцій. Харків : ХНУМГ ім. О. М. Бекетова, 2020. 144 с.

4. Мельник Л. Г. Екологічна економіка : підручник. 2-ге вид., випр. і доп. Суми : Університетська книга, 2003. 348 с.

5. SaveDnipro. Системи громадського моніторингу якості повітря в Україні. URL: https://savednipro.org/ua/air-monitoring (дата звернення: 16.04.2025).

6. Промислова екологія. Курс лекцій / уклад.: Є. О. Троценко, Ю. В. Перетятко. Київ : КПІ ім. Ігоря Сікорського, 2022. 86 с. URL: https://ela.kpi.ua/handle/123456789/47714 (дата звернення: 16.04.2025).

7. PurpleAir. Real-time air quality monitoring with community-driven sensors. URL: https://www2.purpleair.com (дата звернення: 16.04.2025).

8. OpenAQ. Open air quality data platform. URL: https://openaq.org (дата звернення: 16.04.2025).

9. AirVisual. Air quality and pollution monitoring solutions. URL: https://www.iqair.com/air-quality-monitors (дата звернення: 16.04.2025).

10. Ткачук О. П. Моніторинг довкілля : навч.-метод. посіб. Вінниця : РВВ ВНАУ, 2014. 411 с.

11. Погромська Г. С., Махровська Н. А. Бази даних: проектування та реалізація : навч. посіб. 2019. 183 с.

12. Фаулер М. Шаблони корпоративних додатків / пер. з англ. 2-ге вид. Київ : Вільямс, 2019. 560 с.

13. PHP: The Right Way / J. Lockhart, J. Sarjeant.. URL: https://phptherightway.com (дата звернення: 16.04.2025).

14. MySQL 8.4 Reference Manual. Oracle Corporation, URL: https://dev.mysql.com/doc/refman/8.4/en (дата звернення: 16.04.2025).

15. Фленаган Д. JavaScript: повне керівництво / пер. з англ. 7-ме вид. Київ : Діалектика, 2021. 752 с.

16. Leaflet.js Documentation. URL: https://leafletjs.com/reference.html (дата звернення: 16.04.2025).

17. Chart.js Documentation. URL: https://www.chartjs.org/docs/latest (дата звернення: 16.04.2025).

18. Bublyk A. Web Page Design As a Basis for Readability, Content and Aesthetics of a Modern Website. Computer-integrated technologies: education, science, production. 2021. № 45. С. 5–11. DOI: 10.36910/6775-2524-0560-2021-45-01.

19. Кнут Д. Мистецтво програмування. / пер. з англ. 3-тє вид. Київ : Вільямс, 2020. 832 с.

20. Хайрова Н. Ф., Петрасова С. В. Сучасні технології Web-програмування : навч. посіб. Харків : ФОП Панов А. М., 2020. 112 с.

21. Грицунов О. В. Інформаційні системи та технології : навч. посіб. Харків : ХНАМГ, 2010. 222 с.

22. Nginx Documentation. URL: https://nginx.org/en/docs (дата звернення: 16.04.2025).

23. PHP 8.4 Manual. PHP Group, URL: https://www.php.net/manual/en (дата звернення: 16.04.2025).

24. Sommerville I. Software Engineering. 10th ed. London : Pearson, 2019. 816 p.

25. ДСТУ ISO/IEC 25010:2016. Інженерія систем і програмних засобів. Вимоги до якості систем і програмних засобів та її оцінювання (SQuaRE). Київ : УкрНДНЦ, 2016. 34 с.

26. OSPanel Documentation. URL: https://ospanel.io/docs (дата звернення: 16.04.2025).

27. TCPDF Library Documentation. URL: https://tcpdf.org/docs (дата звернення: 16.04.2025).

28. Авраменко А. С., Авраменко В. С., Косенюк Г. В. Тестування програмного забезпечення : навч. посіб. Черкаси : ЧНУ імені Богдана Хмельницького, 2017. 284 с.

29. Pressman R. S. Software Engineering: A Practitioner’s Approach / R. S. Pressman, B. R. Maxim. 9th ed. New York : McGraw-Hill Education, 2020. 704 p.

ДОДАТОК А

Код програми наведений в Лістингу А .1

Лістинг А 1

Код сторінки admin_panel.php

```
<?php
session_start();
require_once '../config.php';

// Перевірка авторизації та ролі адміністратора
if (!isset($_SESSION['user_id'])) {
    header("Location: login.php");
    exit();
}

if (!isAdmin($_SESSION['user_id'], $pdo)) {
    header("Location: login.php");
    exit();
}

// Обробка додавання станції
if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['add_station'])) {
    $name = filter_input(INPUT_POST, 'name', FILTER_SANITIZE_STRING);
    $coordinates = filter_input(INPUT_POST, 'coordinates', FILTER_SANITIZE_STRING);
    $type = filter_input(INPUT_POST, 'type', FILTER_SANITIZE_STRING);
    $status = filter_input(INPUT_POST, 'status', FILTER_SANITIZE_STRING);

    if ($name && $coordinates && $type && $status) {
        $stmt = $pdo->prepare("INSERT INTO monitoring_stations (name, coordinates, type, status)
VALUES (?, ?, ?, ?)");
        $stmt->execute([$name, $coordinates, $type, $status]);
        $success = "Станцію додано успішно!";
    } else {
        $error = "Заповніть усі поля!";
    }
}

// Обробка видалення станції
if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['delete_station'])) {
    $station_id = filter_input(INPUT_POST, 'station_id', FILTER_SANITIZE_NUMBER_INT);
    $stmt = $pdo->prepare("DELETE FROM monitoring_stations WHERE id = ?");
    $stmt->execute([$station_id]);
    $success = "Станцію видалено успішно!";
}

// Обробка зміни ролі користувача
if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['update_role'])) {
    $user_id = filter_input(INPUT_POST, 'user_id', FILTER_SANITIZE_NUMBER_INT);
    $new_role = filter_input(INPUT_POST, 'role', FILTER_SANITIZE_STRING);
```

```
    if ($user_id && in_array($new_role, ['user', 'admin', 'analyst'])) {
        $stmt = $pdo->prepare("UPDATE users SET role = ? WHERE id = ?");
        $stmt->execute([$new_role, $user_id]);
        $success = "Роль користувача оновлено!";
    } else {
        $error = "Невірна роль або ID!";
    }
}

// Отримання списків станцій і користувачів
try {
    $stations = $pdo->query("SELECT * FROM monitoring_stations")->fetchAll(PDO::FETCH_ASSOC);
    $users = $pdo->query("SELECT id, full_name, email, role FROM users")->fetchAll(PDO::FETCH_ASSOC);
} catch (PDOException $e) {
    die("Помилка отримання даних: " . $e->getMessage());
}
?>

<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Адмін-панель - AirGuard</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <nav class="navbar">
        <div class="logo">AirGuard</div>
        <div class="links">
            <a href="index.php">Карта</a>
            <a href="data.php">Дані</a>
            <a href="reports.php">Звіти</a>
            <a href="alerts.php">Сповіщення</a>
            <a href="admin_panel.php">Адмін-панель</a>
            <a href="logout.php">Вихід</a>
        </div>
        <div class="burger">☰</div>
    </nav>

    <div class="main-container">
        <div class="table-container">
            <h1>Адмін-панель</h1>

            <?php if (isset($success)): ?>
                <p class="success"><?php echo htmlspecialchars($success); ?></p>
            <?php endif; ?>
            <?php if (isset($error)): ?>
```

```
            <p class="error"><?php echo htmlspecialchars($error); ?></p>
        <?php endif; ?>

        <!-- Секція: Список станцій -->
        <section>
            <h2>Список станцій</h2>
            <table class="data-table">
                <thead>
                    <tr>
                        <th>ID</th>
                        <th>Назва</th>
                        <th>Координати</th>
                        <th>Тип</th>
                        <th>Статус</th>
                        <th>Дія</th>
                    </tr>
                </thead>
                <tbody>
                    <?php foreach ($stations as $station): ?>
                        <tr>
                            <td><?php echo htmlspecialchars($station['id']); ?></td>
                            <td><?php echo htmlspecialchars($station['name']); ?></td>
                            <td><?php echo isset($station['coordinates']) ?
htmlspecialchars($station['coordinates']) : 'Н/Д'; ?></td>
                            <td><?php echo htmlspecialchars($station['type']); ?></td>
                            <td><?php echo htmlspecialchars($station['status']); ?></td>
                            <td>
                                <form method="POST" style="display:inline;">
                                    <input type="hidden" name="station_id" value="<?php echo $station['id'];
?>">
                                    <button type="submit" name="delete_station" onclick="return
confirm('Видалити станцію?');">Видалити</button>
                                </form>
                            </td>
                        </tr>
                    <?php endforeach; ?>
                    <?php if (empty($stations)): ?>
                        <tr><td colspan="6">Станції відсутні.</td></tr>
                    <?php endif; ?>
                </tbody>
            </table>
        </section>

        <!-- Секція: Керування користувачами -->
        <section>
            <h2>Керування користувачами</h2>
            <table class="data-table">
                <thead>
                    <tr>
                        <th>ID</th>
                        <th>Ім'я</th>
```

```
                    <th>Email</th>
                    <th>Роль</th>
                    <th>Змінити роль</th>
                  </tr>
                </thead>
                <tbody>
                  <?php foreach ($users as $user): ?>
                    <tr>
                      <td><?php echo htmlspecialchars($user['id']); ?></td>
                      <td><?php echo htmlspecialchars($user['full_name']); ?></td>
                      <td><?php echo htmlspecialchars($user['email']); ?></td>
                      <td><?php echo htmlspecialchars($user['role']); ?></td>
                      <td>
                        <form method="POST">
                          <input type="hidden" name="user_id" value="<?php echo $user['id']; ?>">
                          <select name="role" required>
                            <option value="user" <?php echo $user['role'] === 'user' ? 'selected' : ''; 
?>>Користувач</option>
                            <option value="admin" <?php echo $user['role'] === 'admin' ? 'selected' 
: ''; ?>>Адміністратор</option>
                            <option value="analyst" <?php echo $user['role'] === 'analyst' ? 'selected' 
: ''; ?>>Аналітик</option>
                          </select>
                          <button type="submit" name="update_role">Оновити</button>
                        </form>
                      </td>
                    </tr>
                  <?php endforeach; ?>
                  <?php if (empty($users)): ?>
                    <tr><td colspan="5">Користувачі відсутні.</td></tr>
                  <?php endif; ?>
                </tbody>
              </table>
            </section>
          </div>

          <!-- Бічна панель для форми додавання станції -->
          <div class="sidebar">
            <h2>Додати станцію</h2>
            <form method="POST" class="filters">
              <label for="name">Назва станції:</label>
              <input type="text" id="name" name="name" required>

              <label for="coordinates">Координати:</label>
              <input type="text" id="coordinates" name="coordinates" placeholder="наприклад, 
50.456,30.508" required>

              <label for="type">Тип:</label>
              <select id="type" name="type" required>
                <option value="Міська">Міська</option>
                <option value="Промислова">Промислова</option>
```

```
                <option value="Житлова">Житлова</option>
            </select>

            <label for="status">Статус:</label>
            <select id="status" name="status" required>
                <option value="Активна">Активна</option>
                <option value="Неактивна">Неактивна</option>
            </select>

            <button type="submit" name="add_station">Додати станцію</button>
        </form>
    </div>
</div>

<script>
    window.onload = function() {
        const burger = document.querySelector('.burger');
        const links = document.querySelector('.links');
        if (burger && links) {
            burger.addEventListener('click', () => {
                links.classList.toggle('active');
            });
        }
    };
</script>
</body>
</html>
```

Код сторінки alerts.php

```
<?php
session_start();
require_once '../config.php';

// Перевірка авторизації
if (!isset($_SESSION['user_id'])) {
    header("Location: login.php");
    exit();
}

// Отримання даних користувача
try {
    $pdo = new PDO("mysql:host=" . DB_HOST . ";dbname=" . DB_NAME, DB_USER, DB_PASS);
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $stmt = $pdo->prepare("SELECT full_name, role FROM users WHERE id = ?");
    $stmt->execute([$_SESSION['user_id']]);
    $user = $stmt->fetch(PDO::FETCH_ASSOC);
    if (!$user) {
        session_destroy();
        header("Location: login.php");
        exit();
    }
```

```
} catch (PDOException $e) {
    die("Помилка підключення до бази: " . $e->getMessage());
}

// Отримання сповіщень
$status_filter = $_GET['status'] ?? 'all';
$danger_filter = $_GET['danger'] ?? 'all';
try {
    $query = "
        SELECT a.id, a.timestamp, ms.name AS station, a.danger_level, a.status
        FROM alerts a
        JOIN monitoring_stations ms ON a.station_id = ms.id
        WHERE 1=1
    ";
    $params = [];
    if ($status_filter !== 'all') {
        $query .= " AND a.status = ?";
        $params[] = $status_filter;
    }
    if ($danger_filter !== 'all') {
        $query .= " AND a.danger_level = ?";
        $params[] = $danger_filter;
    }
    $query .= " ORDER BY a.timestamp DESC";

    $stmt = $pdo->prepare($query);
    $stmt->execute($params);
    $alerts = $stmt->fetchAll(PDO::FETCH_ASSOC);
} catch (PDOException $e) {
    die("Помилка отримання сповіщень: " . $e->getMessage());
}
?>

<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Сповіщення - AirGuard</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <nav class="navbar">
        <div class="logo">AirGuard</div>
        <div class="links">
            <a href="index.php">Карта</a>
            <a href="data.php">Дані</a>
            <a href="reports.php">Звіти</a>
            <a href="alerts.php">Сповіщення</a>
            <?php if ($user['role'] === 'admin'): ?>
                <a href="admin_panel.php">Адмін-панель</a>
```

```
        <?php endif; ?>
        <a href="logout.php">Вихід</a>
    </div>
    <div class="burger">☰</div>
</nav>

<div class="main-container">
    <div class="table-container">
        <h1>Сповіщення</h1>
        <!-- Таблиця сповіщень -->
        <table class="alerts-table">
            <thead>
                <tr>
                    <th>Час</th>
                    <th>Станція</th>
                    <th>Рівень небезпеки</th>
                    <th>Статус</th>
                    <th>Дія</th>
                </tr>
            </thead>
            <tbody id="alerts-body">
                <?php foreach ($alerts as $alert): ?>
                    <tr data-id="<?php echo htmlspecialchars($alert['id']); ?>">
                        <td><?php echo htmlspecialchars($alert['timestamp']); ?></td>
                        <td><?php echo htmlspecialchars($alert['station']); ?></td>
                        <td><?php echo htmlspecialchars(ucfirst($alert['danger_level'])); ?></td>
                        <td class="status-<?php echo $alert['status']; ?>">
                            <?php echo $alert['status'] === 'unread' ? 'Непрочитане' : 'Прочитане'; ?>
                        </td>
                        <td>
                            <button
                                onclick="markAsRead(<?php echo $alert['id']; ?>, this)"
                                <?php echo $alert['status'] === 'read' ? 'disabled' : ''; ?>
                            >
                                Позначити прочитаним
                            </button>
                        </td>
                    </tr>
                <?php endforeach; ?>
                <?php if (empty($alerts)): ?>
                    <tr><td colspan="5">Сповіщення відсутні.</td></tr>
                <?php endif; ?>
            </tbody>
        </table>
    </div>
    <div class="sidebar">
        <h2>Фільтри</h2>
        <div class="filters">
            <label for="status-filter">Статус:</label>
            <select id="status-filter" onchange="applyFilters()">
                <option value="all" <?php echo $status_filter === 'all' ? 'selected' : ''; ?>>Усі</option>
```

```
            <option value="unread" <?php echo $status_filter === 'unread' ? 'selected' : ''; ?>>Непрочитані</option>
            <option value="read" <?php echo $status_filter === 'read' ? 'selected' : ''; ?>>Прочитані</option>
          </select>
          <label for="danger-filter">Рівень небезпеки:</label>
          <select id="danger-filter" onchange="applyFilters()">
            <option value="all" <?php echo $danger_filter === 'all' ? 'selected' : ''; ?>>Усі рівні</option>
            <option value="low" <?php echo $danger_filter === 'low' ? 'selected' : ''; ?>>Низький</option>
            <option value="medium" <?php echo $danger_filter === 'medium' ? 'selected' : ''; ?>>Середній</option>
            <option value="high" <?php echo $danger_filter === 'high' ? 'selected' : ''; ?>>Високий</option>
          </select>
        </div>
      </div>
    </div>

    <script>
      window.onload = function() {
        const burger = document.querySelector('.burger');
        const links = document.querySelector('.links');
        if (burger && links) {
          burger.addEventListener('click', () => {
            links.classList.toggle('active');
          });
        }
      };

      // Застосування фільтрів
      function applyFilters() {
        const status = document.getElementById('status-filter').value;
        const danger = document.getElementById('danger-filter').value;
        window.location.href = `alerts.php?status=${status}&danger=${danger}`;
      }

      // Позначення сповіщення як прочитане
      function markAsRead(alertId, button) {
        fetch('api.php', {
          method: 'POST',
          headers: { 'Content-Type': 'application/x-www-form-urlencoded' },
          body: `action=mark_read&id=${alertId}`
        })
          .then(response => response.json())
          .then(data => {
            if (data.success) {
              const row = button.closest('tr');
              const statusCell = row.querySelector('.status-unread');
              if (statusCell) {
```

```
                    statusCell.textContent = 'Прочитане';
                    statusCell.className = 'status-read';
                    button.disabled = true;
                }
            } else {
                alert('Помилка при позначенні сповіщення.');
            }
        })
        .catch(error => {
            console.error('Помилка:', error);
            alert('Не вдалося позначити сповіщення.');
        });
    }
  </script>
  <script>
    if ('serviceWorker' in navigator) {
        navigator.serviceWorker.register('/service-worker.js')
            .then(reg => console.log('Service Worker зареєстровано:', reg))
            .catch(err => console.error('Помилка реєстрації Service Worker:', err));
    }
  </script>
</body>
</html>
```

Код сторінки api.php

```
<?php
require_once '../config.php';

// Налаштування заголовків для JSON
header('Content-Type: application/json; charset=utf-8');

// Ініціалізація PDO
try {
    $pdo = new PDO("mysql:host=" . DB_HOST . ";dbname=" . DB_NAME, DB_USER, DB_PASS);
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $pdo->exec("SET NAMES 'utf8'");
} catch (PDOException $e) {
    http_response_code(500);
    echo json_encode(['error' => 'Помилка підключення до бази']);
    exit();
}

// Обробка запитів
$action = $_POST['action'] ?? $_GET['type'] ?? '';

switch ($action) {
    // Запит для карти (index.php)
   case 'stations':
    $pollutant = $_POST['pollutant'] ?? 'all';
    $region = $_POST['region'] ?? 'all';
```

```
    $date = $_POST['date'] ?? date('Y-m-d');

    try {
                $query = "
                        SELECT ms.id, ms.name, ms.latitude, ms.longitude,
                                MAX(CASE WHEN m.pollutant_id = 1 THEN m.value END) AS
pm25,
                                MAX(CASE WHEN m.pollutant_id = 2 THEN m.value END) AS
pm10,
                                MAX(CASE WHEN m.pollutant_id = 3 THEN m.value END) AS
no2
                        FROM monitoring_stations ms
                        LEFT JOIN measurements m ON ms.id = m.station_id
                        WHERE (m.timestamp IS NULL OR m.timestamp >= ? AND m.timestamp <
DATE_ADD(?, INTERVAL 1 DAY))
                    ";
        $params = [$date, $date];
        if ($region !== 'all') {
            $query .= " AND ms.region = ?";
            $params[] = $region;
        }
        if ($pollutant !== 'all') {
            $query .= " AND m.pollutant_id = ?";
            $params[] = $pollutant;
        }
        $query .= " GROUP BY ms.id, ms.name, ms.latitude, ms.longitude";

        $stmt = $pdo->prepare($query);
        $stmt->execute($params);
        $stations = $stmt->fetchAll(PDO::FETCH_ASSOC);

        // Заглушка: якщо дані відсутні, повертаємо статичний приклад
        if (empty($stations)) {
            $stations = [
                [
                    'id' => 1,
                    'name' => 'Станція Лук’янівка',
                    'latitude' => 50.456,
                    'longitude' => 30.508,
                    'pm25' => 25,
                    'pm10' => 40,
                    'no2' => 30
                ],
                [
                    'id' => 2,
                    'name' => 'Станція Оболонь',
                    'latitude' => 50.487,
                    'longitude' => 30.498,
                    'pm25' => 20,
                    'pm10' => 35,
                    'no2' => 25
```

```
            ]
        ];
    }

    echo json_encode($stations);
} catch (PDOException $e) {
    http_response_code(500);
    // Додаємо деталі помилки для дебагінгу (видаліть у продакшені)
    echo json_encode(['error' => 'Помилка отримання станцій: ' . $e->getMessage()]);
}
break;

// Запит для статистики (index.php)
case 'stats':
    try {
        $stmt = $pdo->query("
            SELECT
                AVG(CASE WHEN pollutant_id = 1 THEN value END) AS avg_pm25,
                AVG(CASE WHEN pollutant_id = 2 THEN value END) AS avg_pm10,
                COUNT(DISTINCT station_id) AS active_stations
            FROM measurements
            WHERE timestamp >= NOW() - INTERVAL 1 DAY
        ");
        $stats = $stmt->fetch(PDO::FETCH_ASSOC);

        // Заглушка: якщо даних немає
        if (!$stats['avg_pm25']) {
            $stats = [
                'avg_pm25' => 25,
                'avg_pm10' => 40,
                'active_stations' => 5
            ];
        }

        echo json_encode($stats);
    } catch (PDOException $e) {
        http_response_code(500);
        echo json_encode(['error' => 'Помилка отримання статистики']);
    }
    break;

// Запит для даних (data.php)
case 'measurements':
    $period = $_GET['period'] ?? 'week';
    $pollutant = $_GET['pollutant'] ?? 'all';

    try {
        // Таблиця
        $query = "
            SELECT m.timestamp, ms.name AS station, p.name AS pollutant, m.value
            FROM measurements m
```

```
            JOIN monitoring_stations ms ON m.station_id = ms.id
            JOIN pollutants p ON m.pollutant_id = p.id
            WHERE 1=1
        ";
        $params = [];
        if ($period === 'day') {
            $query .= " AND m.timestamp >= NOW() - INTERVAL 1 DAY";
        } elseif ($period === 'month') {
            $query .= " AND m.timestamp >= NOW() - INTERVAL 1 MONTH";
        } else {
            $query .= " AND m.timestamp >= NOW() - INTERVAL 7 DAY";
        }
        if ($pollutant !== 'all') {
            $query .= " AND m.pollutant_id = ?";
            $params[] = $pollutant;
        }
        $query .= " ORDER BY m.timestamp DESC LIMIT 100";

        $stmt = $pdo->prepare($query);
        $stmt->execute($params);
        $table_data = $stmt->fetchAll(PDO::FETCH_ASSOC);

        // Графік
        $chart_query = "
            SELECT DATE(m.timestamp) AS date, AVG(m.value) AS avg_value, p.name AS
pollutant
            FROM measurements m
            JOIN pollutants p ON m.pollutant_id = p.id
            WHERE 1=1
        ";
        $chart_params = [];
        if ($period === 'day') {
            $chart_query .= " AND m.timestamp >= NOW() - INTERVAL 1 DAY";
        } elseif ($period === 'month') {
            $chart_query .= " AND m.timestamp >= NOW() - INTERVAL 1 MONTH";
        } else {
            $chart_query .= " AND m.timestamp >= NOW() - INTERVAL 7 DAY";
        }
        if ($pollutant !== 'all') {
            $chart_query .= " AND m.pollutant_id = ?";
            $chart_params[] = $pollutant;
        }
        $chart_query .= " GROUP BY DATE(m.timestamp), p.name ORDER BY date";

        $stmt = $pdo->prepare($chart_query);
        $stmt->execute($chart_params);
        $chart_data = $stmt->fetchAll(PDO::FETCH_ASSOC);

        // Форматування для Chart.js
        $labels = array_unique(array_column($chart_data, 'date'));
        $values = array_fill(0, count($labels), 0);
```

```
        $pollutant_name = $pollutant === 'all' ? 'Усі забруднювачі' : $chart_data[0]['pollutant'] ?? 'Невідомий';

        foreach ($chart_data as $row) {
            $index = array_search($row['date'], $labels);
            $values[$index] = round($row['avg_value'], 2);
        }

        // Заглушка: якщо даних немає
        if (empty($table_data)) {
            $table_data = [
                [
                    'timestamp' => '2025-04-13 14:00:00',
                    'station' => 'Лук’янівка',
                    'pollutant' => 'PM2.5',
                    'value' => 25
                ],
                [
                    'timestamp' => '2025-04-13 13:00:00',
                    'station' => 'Оболонь',
                    'pollutant' => 'PM10',
                    'value' => 40
                ]
            ];
            $labels = ['2025-04-13'];
            $values = [25];
            $pollutant_name = 'PM2.5';
        }

        echo json_encode([
            'table' => $table_data,
            'chart' => [
                'labels' => $labels,
                'values' => $values,
                'pollutant' => $pollutant_name
            ]
        ]);
    } catch (PDOException $e) {
        http_response_code(500);
        echo json_encode(['error' => 'Помилка отримання вимірювань']);
    }
    break;

// Позначення сповіщення як прочитане (alerts.php)
case 'mark_read':
    if (!isset($_POST['id'])) {
        http_response_code(400);
        echo json_encode(['success' => false, 'error' => 'ID сповіщення відсутній']);
        exit();
    }
```

```
        try {
            $stmt = $pdo->prepare("UPDATE alerts SET status = 'read' WHERE id = ?");
            $stmt->execute([$_POST['id']]);
            echo json_encode(['success' => true]);
        } catch (PDOException $e) {
            http_response_code(500);
            echo json_encode(['success' => false, 'error' => 'Помилка оновлення статусу']);
        }
        break;

    default:
        http_response_code(400);
        echo json_encode(['error' => 'Невідомий запит']);
        break;
}
```

Код сторінки data.php

```
<?php
session_start();
require_once '../config.php';

// Перевірка авторизації
if (!isset($_SESSION['user_id'])) {
    header("Location: login.php");
    exit();
}

// Отримання даних користувача
try {
    $pdo = new PDO("mysql:host=" . DB_HOST . ";dbname=" . DB_NAME, DB_USER, DB_PASS);
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $stmt = $pdo->prepare("SELECT full_name, role FROM users WHERE id = ?");
    $stmt->execute([$_SESSION['user_id']]);
    $user = $stmt->fetch(PDO::FETCH_ASSOC);
    if (!$user) {
        session_destroy();
        header("Location: login.php");
        exit();
    }
} catch (PDOException $e) {
    die("Помилка підключення до бази: " . $e->getMessage());
}

// Обробка експорту CSV
if (isset($_GET['export']) && $_GET['export'] === 'csv') {
    header('Content-Type: text/csv; charset=utf-8');
    header('Content-Disposition: attachment; filename="measurements.csv"');

    $output = fopen('php://output', 'w');
    fputs($output, "\xEF\xBB\xBF"); // UTF-8 BOM для коректного відображення українських символів
```

```
    fputcsv($output, ['Дата', 'Станція', 'Забруднювач', 'Значення'], ';');

    $stmt = $pdo->query("
        SELECT m.timestamp, ms.name AS station, p.name AS pollutant, m.value
        FROM measurements m
        JOIN monitoring_stations ms ON m.station_id = ms.id
        JOIN pollutants p ON m.pollutant_id = p.id
        ORDER BY m.timestamp DESC
    ");
    while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
        fputcsv($output, [
            $row['timestamp'],
            $row['station'],
            $row['pollutant'],
            $row['value']
        ], ';');
    }
    fclose($output);
    exit();
}
?>

<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Дані - AirGuard</title>
    <link rel="stylesheet" href="styles.css">
    <style>
        /* Базові стилі для адаптивного дизайну */
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
        }
        .navbar {
            display: flex;
            justify-content: space-between;
            align-items: center;
            background-color: #333;
            padding: 10px 20px;
            color: white;
        }
        .navbar .logo {
            font-size: 24px;
            font-weight: bold;
        }
        .navbar .links {
            display: flex;
            gap: 20px;
```

```
}
.navbar .links a {
    color: white;
    text-decoration: none;
    font-size: 16px;
}
.navbar .links a:hover {
    text-decoration: underline;
}
.navbar .burger {
    display: none;
    font-size: 24px;
    cursor: pointer;
}
.main-container {
    display: flex;
    flex-wrap: wrap;
    padding: 20px;
    gap: 20px;
}
.table-container {
    flex: 3;
    min-width: 300px;
}
.sidebar {
    flex: 1;
    min-width: 200px;
    background-color: #f4f4f4;
    padding: 20px;
    border-radius: 8px;
}
.chart-container {
    margin-bottom: 20px;
}
table {
    width: 100%;
    border-collapse: collapse;
    margin-bottom: 20px;
}
th, td {
    padding: 10px;
    text-align: left;
    border: 1px solid #ddd;
}
th {
    background-color: #3498db;
    color: white;
    cursor: pointer;
}
.export-btn {
    display: inline-block;
```

```
            padding: 10px 20px;
            background-color: #3498db;
            color: white;
            text-decoration: none;
            border-radius: 4px;
        }
        .filters select, .filters button {
            margin: 10px 0;
            padding: 8px;
            width: 100%;
            font-size: 16px;
        }
        /* Адаптивність */
        @media (max-width: 768px) {
            .navbar .links {
                display: none;
                flex-direction: column;
                position: absolute;
                top: 60px;
                left: 0;
                width: 100%;
                background-color: #333;
                padding: 20px;
            }
            .navbar .links.active {
                display: flex;
            }
            .navbar .burger {
                display: block;
            }
            .main-container {
                flex-direction: column;
            }
        }
    </style>
</head>
<body>
    <nav class="navbar">
        <div class="logo">AirGuard</div>
        <div class="links">
            <a href="index.php">Карта</a>
            <a href="data.php">Дані</a>
            <a href="reports.php">Звіти</a>
            <a href="alerts.php">Сповіщення</a>
            <?php if ($user['role'] === 'admin'): ?>
                <a href="admin_panel.php">Адмін-панель</a>
            <?php endif; ?>
            <a href="logout.php">Вихід</a>
        </div>
        <div class="burger">☰</div>
    </nav>
```

```
<div class="main-container">
  <div class="table-container">
    <h1>Дані вимірювань</h1>
    <div class="chart-container">
      <canvas id="trendChart"></canvas>
    </div>
    <table id="measurements-table">
      <thead>
        <tr>
          <th data-sort="timestamp">Дата</th>
          <th data-sort="station">Станція</th>
          <th data-sort="pollutant">Забруднювач</th>
          <th data-sort="value">Значення</th>
        </tr>
      </thead>
      <tbody id="table-body"></tbody>
    </table>
    <a href="data.php?export=csv" class="export-btn">Експорт у CSV</a>
  </div>
  <div class="sidebar">
    <h2>Фільтри</h2>
    <div class="filters">
      <select id="period">
        <option value="day">День</option>
        <option value="week" selected>Тиждень</option>
        <option value="month">Місяць</option>
      </select>
      <select id="pollutant">
        <option value="all">Усі</option>
        <option value="1">PM2.5</option>
        <option value="2">PM10</option>
        <option value="3">NO₂</option>
        <option value="4">SO₂</option>
        <option value="5">CO</option>
        <option value="6">O₃</option>
        <option value="7">PM1</option>
        <option value="8">NH₃</option>
        <option value="9">CH₄</option>
        <option value="10">VOC</option>
        <option value="11">H₂S</option>
        <option value="12">NO</option>
        <option value="13">C₆H₆</option>
        <option value="14">Pb</option>
        <option value="15">As</option>
      </select>
      <button onclick="applyFilters()">Застосувати</button>
    </div>
  </div>
</div>
```

```
    <script src="assets/chart.js"></script>
    <script>
        // Ініціалізація графіка
        const ctx = document.getElementById('trendChart').getContext('2d');
        const trendChart = new Chart(ctx, {
            type: 'line',
            data: {
                labels: [],
                datasets: [{
                    label: 'Значення забруднювача',
                    data: [],
                    borderColor: '#3498db',
                    fill: false
                }]
            },
            options: {
                responsive: true,
                scales: {
                    x: { title: { display: true, text: 'Дата' } },
                    y: { title: { display: true, text: 'Значення (мкг/м³)' } }
                }
            }
        });

        // Завантаження даних
        function loadData(period = 'week', pollutant = 'all') {
            fetch(`api.php?type=measurements&period=${period}&pollutant=${pollutant}`)
                .then(response => response.json())
                .then(data => {
                    // Оновлення таблиці
                    const tbody = document.getElementById('table-body');
                    tbody.innerHTML = '';
                    data.table.forEach(row => {
                        const tr = document.createElement('tr');
                        tr.innerHTML = `
                            <td>${row.timestamp}</td>
                            <td>${row.station}</td>
                            <td>${row.pollutant}</td>
                            <td>${row.value} мкг/м³</td>
                        `;
                        tbody.appendChild(tr);
                    });

                    // Оновлення графіка
                    trendChart.data.labels = data.chart.labels;
                    trendChart.data.datasets[0].data = data.chart.values;
                    trendChart.data.datasets[0].label = pollutant === 'all' ? 'Усі забруднювачі' :
data.chart.pollutant;
                    trendChart.update();
                })
                .catch(error => console.error('Помилка завантаження даних:', error));
```

```
        }

        // Застосування фільтрів
        function applyFilters() {
            const period = document.getElementById('period').value;
            const pollutant = document.getElementById('pollutant').value;
            loadData(period, pollutant);
        }

        // Сортування таблиці
        document.querySelectorAll('#measurements-table th[data-sort]').forEach(header => {
            header.addEventListener('click', () => {
                const sortKey = header.dataset.sort;
                const tbody = document.getElementById('table-body');
                const rows = Array.from(tbody.querySelectorAll('tr'));
                const isAsc = header.classList.toggle('asc');
                header.classList.toggle('desc', !isAsc);

                rows.sort((a, b) => {
                    const aValue = a.children[['timestamp', 'station', 'pollutant',
'value'].indexOf(sortKey)].textContent;
                    const bValue = b.children[['timestamp', 'station', 'pollutant',
'value'].indexOf(sortKey)].textContent;
                    if (sortKey === 'value') {
                        return isAsc ? parseFloat(aValue) - parseFloat(bValue) : parseFloat(bValue) -
parseFloat(aValue);
                    }
                    return isAsc ? aValue.localeCompare(bValue) : bValue.localeCompare(bValue);
                });

                tbody.innerHTML = '';
                rows.forEach(row => tbody.appendChild(row));
            });
        });

        // Бургер-меню
        document.querySelector('.burger').addEventListener('click', () => {
            document.querySelector('.links').classList.toggle('active');
        });

        // Ініціалізація
        loadData();
    </script>
</body>
</html>
```

Код сторінки index.php

```
<?php
session_start();
require_once '../config.php';
```

```
if (!isset($_SESSION['user_id'])) {
    header("Location: login.php");
    exit();
}

try {
    $pdo = new PDO("mysql:host=" . DB_HOST . ";dbname=" . DB_NAME, DB_USER, DB_PASS);
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $stmt = $pdo->prepare("SELECT full_name, role FROM users WHERE id = ?");
    $stmt->execute([$_SESSION['user_id']]);
    $user = $stmt->fetch(PDO::FETCH_ASSOC);
    if (!$user) {
        session_destroy();
        header("Location: login.php");
        exit();
    }
} catch (PDOException $e) {
    die("Помилка підключення до бази: " . $e->getMessage());
}
?>

<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Карта - AirGuard</title>
    <link rel="stylesheet" href="styles.css">
    <link rel="stylesheet" href="assets/leaflet.css">
</head>
<body>
    <nav class="navbar">
        <div class="logo">AirGuard</div>
        <div class="links">
            <a href="index.php">Карта</a>
            <a href="data.php">Дані</a>
            <a href="reports.php">Звіти</a>
            <a href="alerts.php">Сповіщення</a>
            <?php if ($user['role'] === 'admin'): ?>
                <a href="admin_panel.php">Адмін-панель</a>
            <?php endif; ?>
            <a href="logout.php">Вихід</a>
        </div>
        <div class="burger">☰</div>
    </nav>

    <div class="main-container">
        <div class="map-container">
            <div id="map"></div>
        </div>
        <div class="sidebar">
```

```
        <h2>Статистика</h2>
        <div class="stats" id="stats">
            <p>Середній PM2.5: <span id="avg-pm25">25 мкг/м³</span></p>
            <p>Середній PM10: <span id="avg-pm10">40 мкг/м³</span></p>
            <p>Кількість активних станцій: <span id="active-stations">5</span></p>
        </div>
        <h2>Фільтри</h2>
        <div class="filters">
            <select id="pollutant">
                                        <option value="all">Усі</option>
                                        <option value="1">PM2.5</option>
                                        <option value="2">PM10</option>
                                        <option value="3">NO₂</option>
                                        <option value="4">SO₂</option>
                                        <option value="5">CO</option>
                                        <option value="6">O₃</option>
                                        <option value="7">PM1</option>
                                        <option value="8">NH₃</option>
                                        <option value="9">CH₄</option>
                                        <option value="10">VOC</option>
                                        <option value="11">H₂S</option>
                                        <option value="12">NO</option>
                                        <option value="13">C₆H₆</option>
                                        <option value="14">Pb</option>
                                        <option value="15">As</option>
                                </select>
            <select id="region">
                <option value="all">Усі регіони</option>
                <option value="Kyiv">Київ</option>
                <option value="Lviv">Львів</option>
            </select>
            <input type="date" id="date" value="<?php echo date('Y-m-d'); ?>">
            <button>Застосувати</button>
        </div>
    </div>
</div>

<script src="assets/leaflet.js"></script>
<script src="scripts.js"></script>
<script>
    // Ініціалізація карти
    const map = L.map('map').setView([50.45, 30.52], 10);
    L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
        attribution: '© OpenStreetMap contributors'
    }).addTo(map);
    window.map = map;

    // Завантаження статистики
    fetch('api.php?type=stats')
        .then(response => response.json())
        .then(data => {
```

```
            document.getElementById('avg-pm25').textContent = data.avg_pm25 + ' мкг/м³';
            document.getElementById('avg-pm10').textContent = data.avg_pm10 + ' мкг/м³';
            document.getElementById('active-stations').textContent = data.active_stations;
        })
        .catch(error => console.error('Помилка завантаження статистики:', error));
    </script>
</body>
</html>
```

Код сторінки login.php

```
<?php
session_start();
//require_once dirname(__FILE__) . '../config.php';
require_once '../config.php';

// Якщо користувач уже авторизований, перенаправляємо на головну сторінку
if (isset($_SESSION['user_id'])) {
    header("Location: index.php");
    exit();
}

$error = '';

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $email = filter_input(INPUT_POST, 'email', FILTER_SANITIZE_EMAIL);
    $password = filter_input(INPUT_POST, 'password', FILTER_SANITIZE_STRING);

    if ($email && $password) {
        try {
            // Пошук користувача за email
            $stmt = $pdo->prepare("SELECT id, full_name, password, role FROM users WHERE email
= ?");
            $stmt->execute([$email]);
            $user = $stmt->fetch(PDO::FETCH_ASSOC);

            // Порівняння пароля напряму
            if ($user && $user['password'] === $password) {
                // Успішна авторизація
                $_SESSION['user_id'] = $user['id'];
                $_SESSION['full_name'] = $user['full_name'];
                $_SESSION['role'] = $user['role'];
                header("Location: index.php");
                exit();
            } else {
                $error = "Невірний email або пароль!";
            }
        } catch (PDOException $e) {
            $error = "Помилка бази даних: " . $e->getMessage();
        }
    } else {
        $error = "Заповніть усі поля!";
```

```
    }
}
?>

<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Вхід - AirGuard</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="login-container">
        <h1>Вхід до AirGuard</h1>

        <?php if ($error): ?>
            <p class="error"><?php echo htmlspecialchars($error); ?></p>
        <?php endif; ?>

        <form method="POST">
            <label for="email">Email:</label>
            <input type="email" id="email" name="email" required>

            <label for="password">Пароль:</label>
            <input type="password" id="password" name="password" required>

            <button type="submit">Увійти</button>
        </form>
    </div>
            <script>
    if ('serviceWorker' in navigator) {
        navigator.serviceWorker.register('/service-worker.js')
            .then(reg => console.log('Service Worker зареєстровано:', reg))
            .catch(err => console.error('Помилка реєстрації Service Worker:', err));
    }
            </script>
</body>
</html>
```

Код сторінки logout.php

```
<?php
session_start();
session_destroy();
header("Location: login.php");
exit();
?>
```

Код сторінки reports.php

```
<?php
session_start();
```

```
require_once '../config.php';
require_once 'assets/tcpdf/tcpdf.php';

// Перевірка авторизації
if (!isset($_SESSION['user_id'])) {
    header("Location: login.php");
    exit();
}

// Отримання даних користувача
try {
    $pdo = new PDO("mysql:host=" . DB_HOST . ";dbname=" . DB_NAME, DB_USER, DB_PASS);
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $stmt = $pdo->prepare("SELECT full_name, role FROM users WHERE id = ?");
    $stmt->execute([$_SESSION['user_id']]);
    $user = $stmt->fetch(PDO::FETCH_ASSOC);
    if (!$user) {
        session_destroy();
        header("Location: login.php");
        exit();
    }
} catch (PDOException $e) {
    die("Помилка підключення до бази: " . $e->getMessage());
}

// Отримання списку станцій для форми
try {
    $stmt = $pdo->query("SELECT id, name FROM monitoring_stations ORDER BY name");
    $stations = $stmt->fetchAll(PDO::FETCH_ASSOC);
} catch (PDOException $e) {
    die("Помилка отримання станцій: " . $e->getMessage());
}

// Ініціалізація параметрів звіту
$report_data = [];
$period = $_POST['period'] ?? ($_SESSION['report_period'] ?? 'week');
$station_id = $_POST['station_id'] ?? ($_SESSION['report_station_id'] ?? 'all');
$show_report = false;

// Збереження параметрів у сесії
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $_SESSION['report_period'] = $period;
    $_SESSION['report_station_id'] = $station_id;
}

// Генерація даних звіту
if ($_SERVER['REQUEST_METHOD'] === 'POST' && (isset($_POST['generate']) ||
isset($_POST['export_pdf']))) {
    $show_report = true;
    try {
        $query = "
```

```
            SELECT m.timestamp, ms.name AS station, p.name AS pollutant, m.value
            FROM measurements m
            JOIN monitoring_stations ms ON m.station_id = ms.id
            JOIN pollutants p ON m.pollutant_id = p.id
            WHERE 1=1
        ";
        $params = [];
        if ($station_id !== 'all') {
            $query .= " AND m.station_id = ?";
            $params[] = $station_id;
        }
        if ($period === 'day') {
            $query .= " AND m.timestamp >= NOW() - INTERVAL 1 DAY";
        } elseif ($period === 'month') {
            $query .= " AND m.timestamp >= NOW() - INTERVAL 1 MONTH";
        } else {
            $query .= " AND m.timestamp >= NOW() - INTERVAL 7 DAY";
        }
        $query .= " ORDER BY m.timestamp DESC";

        $stmt = $pdo->prepare($query);
        $stmt->execute($params);
        $report_data = $stmt->fetchAll(PDO::FETCH_ASSOC);
    } catch (PDOException $e) {
        die("Помилка генерації звіту: " . $e->getMessage());
    }
}

// Експорт у PDF
if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['export_pdf'])) {
    $pdf = new TCPDF();
    $pdf->SetCreator('AirGuard');
    $pdf->SetAuthor($user['full_name']);
    $pdf->SetTitle('Звіт про якість повітря');
    $pdf->AddPage();
    $pdf->SetFont('dejavusans', '', 12);

    // Заголовок
    $pdf->Write(0, "Звіт про якість повітря\n", '', 0, 'C');
    $pdf->Ln(10);
    $pdf->Write(0, "Період: " . ($period === 'day' ? 'День' : ($period === 'month' ? 'Місяць' :
'Тиждень')) . "\n");
    $pdf->Write(0, "Станція: " . ($station_id === 'all' ? 'Усі' : $stations[array_search($station_id,
array_column($stations, 'id'))]['name']) . "\n");
    $pdf->Ln(10);

    // Таблиця
    $html = '<table border="1" cellpadding="5">
            <tr>
                <th>Дата</th>
                <th>Станція</th>
```

```
                <th>Забруднювач</th>
                <th>Значення (мкг/м³)</th>
            </tr>';
    foreach ($report_data as $row) {
        $html .= '<tr>
                <td>' . htmlspecialchars($row['timestamp']) . '</td>
                <td>' . htmlspecialchars($row['station']) . '</td>
                <td>' . htmlspecialchars($row['pollutant']) . '</td>
                <td>' . htmlspecialchars($row['value']) . '</td>
            </tr>';
    }
    $html .= '</table>';
    $pdf->writeHTML($html, true, false, true, false, '');

    // Виведення PDF
    $pdf->Output('airguard_report.pdf', 'D');
    exit();
}
?>

<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Звіти - AirGuard</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <nav class="navbar">
        <div class="logo">AirGuard</div>
        <div class="links">
            <a href="index.php">Карта</a>
            <a href="data.php">Дані</a>
            <a href="reports.php">Звіти</a>
            <a href="alerts.php">Сповіщення</a>
            <?php if ($user['role'] === 'admin'): ?>
                <a href="admin_panel.php">Адмін-панель</a>
            <?php endif; ?>
            <a href="logout.php">Вихід</a>
        </div>
        <div class="burger">☰</div>
    </nav>

    <div class="main-container">
        <div class="table-container">
            <h1>Генерація звітів</h1>
            <?php if ($show_report): ?>
                <h2>Результати звіту</h2>
                <?php if (empty($report_data)): ?>
                    <p>Дані за вибраний період відсутні.</p>
```

```
            <?php else: ?>
                <table>
                    <thead>
                        <tr>
                            <th>Дата</th>
                            <th>Станція</th>
                            <th>Забруднювач</th>
                            <th>Значення (мкг/м³)</th>
                        </tr>
                    </thead>
                    <tbody>
                        <?php foreach ($report_data as $row): ?>
                            <tr>
                                <td><?php echo htmlspecialchars($row['timestamp']); ?></td>
                                <td><?php echo htmlspecialchars($row['station']); ?></td>
                                <td><?php echo htmlspecialchars($row['pollutant']); ?></td>
                                <td><?php echo htmlspecialchars($row['value']); ?></td>
                            </tr>
                        <?php endforeach; ?>
                    </tbody>
                </table>
            <?php endif; ?>
        <?php endif; ?>
    </div>
    <div class="sidebar">
        <h2>Фільтри</h2>
        <form method="post" class="filters">
            <label for="period">Період:</label>
            <select id="period" name="period">
                <option value="day" <?php echo $period === 'day' ? 'selected' : ''; ?>>Останній день</option>
                <option value="week" <?php echo $period === 'week' ? 'selected' : ''; ?>>Останній тиждень</option>
                <option value="month" <?php echo $period === 'month' ? 'selected' : ''; ?>>Останній місяць</option>
            </select>

            <label for="station_id">Станція:</label>
            <select id="station_id" name="station_id">
                <option value="all" <?php echo $station_id === 'all' ? 'selected' : ''; ?>>Усі</option>
                <?php foreach ($stations as $station): ?>
                    <option value="<?php echo htmlspecialchars($station['id']); ?>" <?php echo $station_id === $station['id'] ? 'selected' : ''; ?>>
                        <?php echo htmlspecialchars($station['name']); ?>
                    </option>
                <?php endforeach; ?>
            </select>

            <button type="submit" name="generate">Показати звіт</button>
            <?php if ($show_report && !empty($report_data)): ?>
                <button type="submit" name="export_pdf">Експорт у PDF</button>
```

```
            <?php endif; ?>
        </form>
    </div>
</div>

<script>
    window.onload = function() {
        const burger = document.querySelector('.burger');
        const links = document.querySelector('.links');
        if (burger && links) {
            burger.addEventListener('click', () => {
                links.classList.toggle('active');
            });
        }
    };
</script>
<script>
    if ('serviceWorker' in navigator) {
        navigator.serviceWorker.register('/service-worker.js')
            .then(reg => console.log('Service Worker зареєстровано:', reg))
            .catch(err => console.error('Помилка реєстрації Service Worker:', err));
    }
</script>
</body>
</html>
```

Код файлу scripts.js

```
// Універсальна функція для ініціалізації
document.addEventListener('DOMContentLoaded', () => {
    // Бургер-меню (усі сторінки)
    const burger = document.querySelector('.burger');
    const links = document.querySelector('.links');
    if (burger && links) {
        burger.addEventListener('click', () => {
            links.classList.toggle('active');
        });
    }

    // Фільтри для index.php
    const indexFilters = {
        pollutant: document.getElementById('pollutant'),
        region: document.getElementById('region'),
        date: document.getElementById('date')
    };
    const applyIndexFiltersBtn = document.querySelector('.filters button');
    if (indexFilters.pollutant && indexFilters.region && indexFilters.date && applyIndexFiltersBtn)
{
        applyIndexFiltersBtn.addEventListener('click', () => {
            const data = {
                pollutant: indexFilters.pollutant.value,
                region: indexFilters.region.value,
```

```
            date: indexFilters.date.value
        };
        fetch('api.php?type=stations', {
            method: 'POST',
            headers: { 'Content-Type': 'application/x-www-form-urlencoded' },
            body: new URLSearchParams(data).toString()
        })
            .then(response => response.json())
            .then(stations => {
                // Оновлення карти (припускаємо, що Leaflet ініціалізований)
                if (window.map) {
                    map.eachLayer(layer => {
                        if (layer instanceof L.CircleMarker) map.removeLayer(layer);
                    });
                    stations.forEach(station => {
                        const color = station.pm25 > 50 ? 'red' : station.pm25 > 25 ? 'orange' : 'green';
                        L.circleMarker([station.latitude, station.longitude], {
                            radius: 8,
                            color: color,
                            fillOpacity: 0.8
                        })
                        .addTo(map)
                        .bindPopup(`
                            <b>${station.name}</b><br>
                            PM2.5: ${station.pm25} мкг/м³<br>
                            PM10: ${station.pm10} мкг/м³<br>
                            NO₂: ${station.no2} мкг/м³
                        `);
                    });
                }
            })
            .catch(error => console.error('Помилка фільтрації:', error));
    });
}

// Фільтри для data.php
const dataFilters = {
    period: document.getElementById('period'),
    pollutant: document.getElementById('pollutant')
};
const applyDataFiltersBtn = document.querySelector('.filters button');
if (dataFilters.period && dataFilters.pollutant && applyDataFiltersBtn) {
    applyDataFiltersBtn.addEventListener('click', () => {
        const period = dataFilters.period.value;
        const pollutant = dataFilters.pollutant.value;
        fetch(`api.php?type=measurements&period=${period}&pollutant=${pollutant}`)
            .then(response => response.json())
            .then(data => {
                // Оновлення таблиці
                const tbody = document.getElementById('table-body');
                if (tbody) {
```

```
                tbody.innerHTML = '';
                data.table.forEach(row => {
                    const tr = document.createElement('tr');
                    tr.innerHTML = `
                        <td>${row.timestamp}</td>
                        <td>${row.station}</td>
                        <td>${row.pollutant}</td>
                        <td>${row.value} мкг/м³</td>
                    `;
                    tbody.appendChild(tr);
                });
            }

            // Оновлення графіка
            if (window.trendChart) {
                trendChart.data.labels = data.chart.labels;
                trendChart.data.datasets[0].data = data.chart.values;
                trendChart.data.datasets[0].label = pollutant === 'all' ? 'Усі забруднювачі' : data.chart.pollutant;
                trendChart.update();
            }
        })
        .catch(error => console.error('Помилка завантаження даних:', error));
    });
}

// Сортування таблиці для data.php
const tableHeaders = document.querySelectorAll('#measurements-table th[data-sort]');
if (tableHeaders.length > 0) {
    tableHeaders.forEach(header => {
        header.addEventListener('click', () => {
            const sortKey = header.dataset.sort;
            const tbody = document.getElementById('table-body');
            const rows = Array.from(tbody.querySelectorAll('tr'));
            const isAsc = header.classList.toggle('asc');
            header.classList.toggle('desc', !isAsc);

            rows.sort((a, b) => {
                const aValue = a.children[['timestamp', 'station', 'pollutant', 'value'].indexOf(sortKey)].textContent;
                const bValue = b.children[['timestamp', 'station', 'pollutant', 'value'].indexOf(sortKey)].textContent;
                if (sortKey === 'value') {
                    return isAsc ? parseFloat(aValue) - parseFloat(bValue) : parseFloat(bValue) - parseFloat(aValue);
                }
                return isAsc ? aValue.localeCompare(bValue) : bValue.localeCompare(aValue);
            });

            tbody.innerHTML = '';
            rows.forEach(row => tbody.appendChild(row));
```

```
        });
    });
}

// Фільтри для alerts.php
const statusFilter = document.getElementById('status-filter');
const dangerFilter = document.getElementById('danger-filter');
if (statusFilter && dangerFilter) {
    const applyFilters = () => {
        const status = statusFilter.value;
        const danger = dangerFilter.value;
        window.location.href = `alerts.php?status=${status}&danger=${danger}`;
    };
    statusFilter.addEventListener('change', applyFilters);
    dangerFilter.addEventListener('change', applyFilters);
}

// Позначення сповіщень як прочитаних (alerts.php)
window.markAsRead = function(alertId, button) {
    fetch('api.php', {
        method: 'POST',
        headers: { 'Content-Type': 'application/x-www-form-urlencoded' },
        body: `action=mark_read&id=${alertId}`
    })
        .then(response => response.json())
        .then(data => {
            if (data.success) {
                const row = button.closest('tr');
                const statusCell = row.querySelector('.status-unread');
                if (statusCell) {
                    statusCell.textContent = 'Прочитане';
                    statusCell.className = 'status-read';
                    button.disabled = true;
                }
            } else {
                alert('Помилка при позначенні сповіщення.');
            }
        })
        .catch(error => {
            console.error('Помилка:', error);
            alert('Не вдалося позначити сповіщення.');
        });
};
});
```

Код файлу service-worker.js

```
// Версія кешу для оновлення
const CACHE_NAME = 'airguard-cache-v1';

// Список ресурсів для кешування
const urlsToCache = [
```

```
    '/',
    '/index.php',
    '/data.php',
    '/reports.php',
    '/alerts.php',
    '/login.php',
    '/styles.css',
    '/scripts.js',
    '/assets/leaflet.js',
    '/assets/leaflet.css',
    '/assets/chart.js',
    '/assets/tcpdf/tcpdf.php', // Для reports.php (якщо доступний клієнту)
    '/favicon.ico' // Якщо є favicon
];

// Подія встановлення Service Worker
self.addEventListener('install', event => {
    event.waitUntil(
        caches.open(CACHE_NAME)
            .then(cache => {
                console.log('Відкрито кеш:', CACHE_NAME);
                return cache.addAll(urlsToCache);
            })
            .then(() => self.skipWaiting()) // Активувати SW одразу
    );
});

// Подія активації Service Worker
self.addEventListener('activate', event => {
    event.waitUntil(
        caches.keys().then(cacheNames => {
            return Promise.all(
                cacheNames.map(cacheName => {
                    if (cacheName !== CACHE_NAME) {
                        console.log('Видаляємо старий кеш:', cacheName);
                        return caches.delete(cacheName);
                    }
                })
            );
        }).then(() => self.clients.claim()) // Взяти контроль над клієнтами
    );
});

// Подія отримання запиту
self.addEventListener('fetch', event => {
    const requestUrl = new URL(event.request.url);

    // Стратегія для API-запитів (Network First)
    if (requestUrl.pathname === '/api.php') {
        event.respondWith(
            fetch(event.request)
```

```
            .then(networkResponse => {
                // Кешуємо успішну відповідь
                return caches.open(CACHE_NAME).then(cache => {
                    cache.put(event.request, networkResponse.clone());
                    return networkResponse;
                });
            })
            .catch(() => {
                // Повертаємо кешовану відповідь у разі офлайну
                return caches.match(event.request)
                    .then(response => {
                        return response || new Response(JSON.stringify({
                            error: 'Офлайн-режим: дані недоступні'
                        }), {
                            status: 503,
                            headers: { 'Content-Type': 'application/json' }
                        });
                    });
            })
        );
        return;
    }
    // Стратегія для статичних ресурсів (Cache First)
    event.respondWith(
        caches.match(event.request)
            .then(response => {
                if (response) {
                    return response; // Повертаємо з кешу
                }
                return fetch(event.request)
                    .then(networkResponse => {
                        // Кешуємо нові ресурси
                        if (networkResponse.status === 200) {
                            caches.open(CACHE_NAME).then(cache => {
                                cache.put(event.request, networkResponse.clone());
                            });
                        }
                        return networkResponse;
                    });
            })
            .catch(() => {
                // Офлайн-заглушка для HTML-сторінок
                if (event.request.mode === 'navigate') {
                    return caches.match('/index.php');
                }
                return new Response('Офлайн: ресурс недоступний', {
                    status: 503
                });
            })
    );
});
```