

ДЕРЖАВНИЙ УНІВЕРСИТЕТ  
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ  
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ  
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ  
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

**КВАЛІФІКАЦІЙНА РОБОТА**

на тему: «Веб-застосунок для моніторингу погодних та екологічних показників на мові Javascript (фреймворк Vue.js)»

на здобуття освітнього ступеня магістра  
зі спеціальності ЕЗ Комп'ютерні науки  
(код, найменування спеціальності)  
освітньо-професійної програми Комп'ютерні науки  
(назва)

*Кваліфікаційна робота містить результати власних досліджень.  
Використання ідей, результатів і текстів інших авторів мають посилання  
на відповідне джерело*

\_\_\_\_\_  
(підпис)

Артем ІВАСЮК  
(Ім'я, ПРІЗВИЩЕ здобувача)

Виконав:  
здобувач вищої освіти  
група КНДМ-61

Артем ІВАСЮК

Керівник:  
науковий ступінь,  
вчене звання

Олена ШИКУЛА  
д.ф.-м.н., професор

Рецензент:  
науковий ступінь,  
вчене звання

\_\_\_\_\_  
(Ім'я, ПРІЗВИЩЕ)

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ  
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**  
**Навчально-науковий інститут інформаційних технологій**

Кафедра Комп'ютерних наук

Ступінь вищої освіти Магістр

Спеціальність ЕЗ Комп'ютерні науки

Освітньо-професійна програма Комп'ютерні науки

**ЗАТВЕРДЖУЮ**

Завідувач кафедру Комп'ютерних наук

\_\_\_\_\_ Віктор ВИШНІВСЬКИЙ  
« \_\_\_\_\_ » \_\_\_\_\_ 2025 р.

**ЗАВДАННЯ  
НА КВАЛІФІКАЦІЙНУ РОБОТУ**

\_\_\_\_\_ Івасюку Артему Ігоровичу

*(прізвище, ім'я, по батькові здобувача)*

1. Тема кваліфікаційної роботи: Веб-застосунок для моніторингу погодних та екологічних показників на мові Javascript (фреймворк Vue.js)

керівник кваліфікаційної роботи Олена ШИКУЛА д.ф.-м.н., професор,

*(Ім'я, ПРИЗВИЩЕ науковий ступінь, вчене звання)*

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «30» жовтня 2025 р. № 467

2. Строк подання кваліфікаційної роботи «26» грудня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література з веб-проектування та екологічного моніторингу; мова програмування JavaScript (надбудова TypeScript); фреймворк Vue.js 3; середовище програмування Visual Studio Code, інструментарій збірки Vite; технології інтеграції з LLM (Google Gemini API); протоколи обміну даними REST API(OpenWeatherMap).

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

Аналіз сучасних метеорологічних та екологічних веб-сервісів: огляд існуючих рішень (AccuWeather, Windy), аналіз їх функціональних можливостей, переваг та недоліків у контексті надання екологічних порад.

Проектування архітектури інтелектуального веб-застосунку: розробка структури даних, вибір технологій для обробки великих масивів метеоданих та інтеграції з генеративним штучним інтелектом.

Реалізація адаптивності та кросбраузерності: розробка інтерфейсу, що коректно відображається на різних типах пристроїв (мобільні, планшети, десктопи).

5. Перелік графічного матеріалу: *презентація*

1. Огляд архітектури інформаційних потоків системи: діаграма, що відображає взаємодію клієнтської частини (Vue.js) із зовнішніми джерелами даних (OpenWeatherMap API) та хмарною моделлю ШІ (Google Gemini).

2. Блок-схема алгоритму обробки метеоданих: схема, що ілюструє процеси отримання сирих даних, їх мапінг та фільтрація добових показників.

3. Екранні форми веб-застосунку: демонстрація основного інтерфейсу, блоку аналітики та системи пошуку міст.

4. Візуалізація динамічних погодних ефектів: демонстрація роботи компоненту погоди для р.

5. Фрагменти програмного коду: приклади реалізації промпт-інжинірингу, типізації даних та логіки обробки якості повітря. для реалізації ключових функцій додатку.

6. Дата видачі завдання«15» вересня 2025 р.

**КАЛЕНДАРНИЙ ПЛАН**

| № з/п | Назва етапів кваліфікаційної роботи                                         | Строк виконання етапів роботи | Примітка |
|-------|-----------------------------------------------------------------------------|-------------------------------|----------|
| 1     | Аналіз наявної науково-технічної літератури                                 | 16.09-27.09.25                | виконано |
| 2     | Порівняльний огляд існуючих метеорологічних сервісів та API-інтерфейсів     | 28.09-11.10.25                | виконано |
| 3     | Визначення функціональних вимог до застосунку                               | 12.10-25.10.25                | виконано |
| 4     | Вибір та обґрунтування інструментів розробки застосунку                     | 26.10-04.11.25                | виконано |
| 5     | Розробка бізнес-логіки моніторингу                                          | 05.11-23.11.25                | виконано |
| 6     | Реалізація візуальної частини: динамічні погодні умови та адаптивна верстка | 24.11-02.12.25                | виконано |
| 7     | Оформлення роботи: вступ, висновки, реферат                                 | 02.12-08.12.25                | виконано |
| 8     | Розробка демонстраційних матеріалів                                         | 09.12-13.12.25                | виконано |

Здобувачка вищої освіти

(підпис)

Артем ІВАСЮК

(Ім'я, ПРІЗВИЩЕ)

Керівник

кваліфікаційної роботи

(підпис)

Олена ШИКУЛА

(Ім'я, ПРІЗВИЩЕ)





## РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 75 стор., 1 табл., 17 рис., 90 джерел.

*Наукове завдання* – проведення комплексного аналізу сучасних метеорологічних та екологічних веб-сервісів, дослідження методів обробки великих масивів даних у реальному часі; розробка інтелектуального веб-застосунку для моніторингу стану довкілля з використанням генеративних моделей штучного інтелекту для формування персоналізованих рекомендацій; створення адаптивного інтерфейсу з динамічною візуалізацією погодних умов.

*Мета роботи* – розробка веб-застосунку «EcoWatch», призначеного для надання користувачам точних метеорологічних та екологічних даних, підкріплених інтелектуальною аналітикою на базі штучного інтелекту, із забезпеченням високого рівня зручності інтерфейсу на різних типах пристроїв.

*Об'єкт дослідження* – процес створення веб-додатку для мережевого сховища NAS.

*Предмет дослідження* – архітектура, алгоритми обробки даних та методи інтеграції генеративного ШІ у веб-застосунки на базі фреймворку Vue.js.

*Короткий зміст роботи:*

Проведено огляд існуючих метеорологічних платформ, здійснено порівняльний аналіз їх функціоналу та точності екологічних показників. Розроблено веб-застосунок на основі Vue.js 3, TypeScript та Vite, який інтегрує дані OpenWeatherMap API та аналітичні можливості Google Gemini AI. Реалізовано унікальну систему динамічної візуалізації погодних ефектів за допомогою CSS-анімацій та модуль промпт-інжинірингу для отримання екологічних порад у реальному часі. Розроблено адаптивні версії інтерфейсу, оптимізовані для смартфонів, планшетів та персональних комп'ютерів.

Ключові слова: екологічний моніторинг, веб-застосунок, Vue.js, Typescript

## ABSTRACT

The textual part of the Master's thesis comprises 75 pages, 1 table, 17 figures and 90 references.

*Research objective:* To conduct a comprehensive analysis of modern meteorological and environmental web services and investigate methods for real-time data processing; to develop an intelligent web application for monitoring environmental conditions using generative artificial intelligence (AI) models to provide personalized recommendations; to create a responsive interface with dynamic visualization of weather conditions.

*Purpose of the study:* To develop the "EcoWatch" web application, designed to provide users with accurate meteorological and environmental data supported by intelligent AI-driven analytics, while ensuring high interface usability across various types of devices.

*Object of study:* The process of developing intelligent information systems for monitoring weather and environmental indicators.

*Subject of study:* The architecture, data processing algorithms, and methods for integrating generative AI into web applications based on the Vue.js framework.

*Summary of the work:* A review of existing meteorological platforms was conducted, along with a comparative analysis of their functionality and the accuracy of their environmental indicators. A web application was developed based on Vue.js 3, TypeScript, and Vite, integrating data from the OpenWeatherMap API and the analytical capabilities of Google Gemini AI. The project implemented a unique system for dynamic visualization of weather effects using CSS animations and a prompt engineering module to obtain real-time environmental insights. Responsive versions of the interface were developed and optimized for smartphones, tablets, and personal computers.

**Keywords:** environmental monitoring, web application, Vue.js, Typescript

# ЗМІСТ

|                                                                                              |    |
|----------------------------------------------------------------------------------------------|----|
| ВСТУП.....                                                                                   | 10 |
| 1 ОГЛЯД ТА АНАЛІЗ ОБЛАСТІ ДОСЛІДЖЕННЯ .....                                                  | 1  |
| 1.1 Концептуальні основи систем екологічного та метеорологічного моніторингу .....           | 1  |
| 1.1.1 Класифікація інформаційних систем контролю стану довкілля.....                         | 1  |
| 1.1.2 Структура та джерела даних метеорологічних веб-застосунків.....                        | 3  |
| 1.1.3 Еволюція технологій прогнозування: від статистичних методів до штучного інтелекту..... | 4  |
| 1.1.4 Функціональні можливості сучасних систем моніторингу AQI.....                          | 6  |
| 1.2 Порівняльний аналіз провідних веб-застосунків та сервісів-агрегаторів.....               | 8  |
| 1.2.1 Комерційні рішення для метеомоніторингу .....                                          | 8  |
| 1.2.2 Відкриті екологічні платформи та API .....                                             | 10 |
| 1.2.3 Порівняльна оцінка ефективності існуючих архітектурних рішень .....                    | 12 |
| 1.2.4 Аналіз технологічних бар'єрів та дефіциту функціональності.....                        | 14 |
| 1.3 Постановка задачі.....                                                                   | 15 |
| 1.3.1 Концептуальне обґрунтування створення системи.....                                     | 15 |
| 1.3.2 Визначення мети та основних завдань проєкту .....                                      | 16 |
| 1.3.3 Специфікація функціональних можливостей застосунку .....                               | 18 |
| 1.3.4 Вимоги до якості та продуктивності ПЗ.....                                             | 19 |
| 1.3.5 Операційні обмеження та проєктні припущення .....                                      | 21 |
| 1.3.6 Показники результативності та успішної реалізації .....                                | 22 |
| 2 ПРОЄКТУВАННЯ ТА ОБҐРУНТУВАННЯ ТЕХНІЧНИХ РІШЕНЬ.....                                        | 24 |
| 2.1 Архітектурна концепція інтелектуального веб-застосунку.....                              | 24 |
| 2.1.1 Вибір архітектурного патерну .....                                                     | 24 |
| 2.1.2 Проєктування схеми взаємодії з хмарними API .....                                      | 25 |
| 2.2 Проєктування логіки обробки екологічних даних.....                                       | 26 |
| 2.2.1 Модульна організація логіки доступу до даних .....                                     | 26 |
| 2.2.2 Моделювання потоків даних та форматів обміну .....                                     | 27 |
| 2.2.3 Алгоритм формування інтелектуальних промптів для LLM .....                             | 29 |
| 2.2.4 WebSocket real-time комунікація .....                                                  | 30 |

|                                                                       |    |
|-----------------------------------------------------------------------|----|
| 2.3 Розробка клієнтської частини (Frontend).....                      | 31 |
| 2.3.1 Модульна структура компонентів.....                             | 31 |
| 2.3.2 Управління станом застосунку.....                               | 33 |
| 2.3.3 Реалізація реактивної візуалізації та адаптивності .....        | 35 |
| 2.4 Обґрунтування вибору технологічного стеку .....                   | 36 |
| 2.4.1. Переваги Vue 3 та Vite у сучасній веб-розробці.....            | 36 |
| 2.4.2 Роль TypeScript у забезпеченні надійності системи.....          | 38 |
| 2.4.3 Вибір моделі Gemini 1.5 Flash для задач метеомоніторингу .....  | 39 |
| 3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ .....                 | 41 |
| 3.1 Середовище розробки та конфігурація проєкту .....                 | 41 |
| 3.1.1 Загальна структура проєкту та дерево компонентів .....          | 41 |
| 3.1.2 Модель даних та типізація .....                                 | 42 |
| 3.1.3 API-оркестрація та маршрутизація .....                          | 44 |
| 3.2 Реалізація основного функціоналу.....                             | 45 |
| 3.2.1 Модуль отримання та мапінгу метеорологічних даних.....          | 45 |
| 3.2.2 Інтеграція інтелектуального аналітика Gemini 1.5 Flash .....    | 47 |
| 3.2.3 Реалізація реактивного управління станом та потоками даних..... | 49 |
| 3.2.4 Обробка помилок та станів завантаження.....                     | 51 |
| 3.3 Адаптивний дизайн та візуалізація .....                           | 53 |
| 3.3.1 Адаптивні контрольні точки.....                                 | 53 |
| 3.3.2 Мобільна версія та Mobile-first стратегія .....                 | 54 |
| 3.3.3 Десктопна версія та розширені віджети .....                     | 56 |
| 3.3.4 Реалізація динамічних фонів залежно від стану довкілля.....     | 57 |
| 3.4 Реалізація реактивного оновлення інтерфейсу .....                 | 61 |
| 3.5. Безпека та оптимізація.....                                      | 63 |
| 3.6. Тестування та верифікація результатів.....                       | 64 |
| 3.7 Інструкція з встановлення .....                                   | 66 |
| 3.8 Висновки до розділу 3 .....                                       | 69 |
| ВИСНОВКИ.....                                                         | 72 |
| ПЕРЕЛІК ПОСИЛАНЬ .....                                                | 75 |
| ДОДАТОК А. ФРАГМЕНТИ ПРОГРАМНОГО КОДУ .....                           | 82 |
| ДОДАТОК Б. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (ПРЕЗЕНТАЦІЯ).....                | 97 |

## ВСТУП

В епоху цифрової трансформації та глобальних кліматичних змін обсяг метеорологічних та екологічних даних, що генерується сенсорами по всьому світу, зростає експоненціально. Веб-застосунки для моніторингу довкілля стають все більш популярними рішеннями для оперативного отримання інформації про стан атмосфери. Мета таких систем - забезпечити користувачам надійне, безпечне та зручне середовище для відстеження показників якості повітря та погоди, розвиток навичок використання сучасних інтелектуальних технологій і застосування ефективних інструментів аналітики на практиці [4, 12, 21]. Вони є ефективним засобом для інформування населення, а також чудово підходять для індивідуального використання, даючи можливість глибше розуміти вплив навколишнього середовища на здоров'я через веб-інтерфейс.

Особливої гостроти це питання набуває для України. Сучасний екологічний стан держави характеризується як кризовий, що зумовлено тривалим домінуванням ресурсомістких і енергоємних технологій та застарілим обладнанням промислових гігантів [1, 5, 18]. Проблема забруднення атмосферного повітря дрібнодисперсним пилом (PM<sub>2.5</sub> та PM<sub>10</sub>) залишається однією з головних причин зростання захворюваності серед міського населення України [9, 14]. Додатковим дестабілізуючим фактором стали наслідки військових дій, що призводять до непередбачуваних викидів шкідливих речовин в атмосферу [3, 11]. У таких умовах актуальним є створення незалежних цифрових інструментів для отримання даних у реальному часі [6, 15].

Крім наявності багатьох готових метеорологічних рішень від провідних провайдерів (AccuWeather, Weather.com, Windy), існує обмаль доступних веб-застосунків з відкритим вихідним кодом, які пропонують можливості для глибокої інтелектуальної аналітики екологічних ризиків. Комерційні рішення користуються значним попитом, але їхній функціонал часто обмежений стандартними прогнозами і не завжди відповідає специфічним потребам користувачів щодо екологічної безпеки. Тому варто відзначити можливість створення власних веб-

систем на базі Vue.js та Google Gemini AI, які усувають ці недоліки та надають більш адаптований інструмент для роботи з даними [8, 19, 22].

Системам моніторингу даних присвячено значний обсяг літератури. Більшість із них є описами комерційних продуктів та технічною документацією [2, 7, 8, 10, 16, 17]. Деякі джерела мають практичну спрямованість щодо розгортання екологічних датчиків [11, 17, 19]. Водночас літератури про розробку інтелектуальних веб-інтерфейсів з використанням генеративного ШІ значно менше [1, 6, 13, 14]. Серед вітчизняних досліджень [12, 13, 14, 15, 18] україномовні ресурси зосереджені переважно на питаннях загального адміністрування систем [13] і програмування веб-додатків [14, 15].

Огляд літератури дозволив зробити такі висновки. Сучасний ринок пропонує готові рішення для прогнозу погоди, але веб-застосунків з відкритим кодом, які мають зручний інтерфейс та інтегровану систему ШІ-рекомендацій, досі недостатньо. Аналізуючи наявні інструменти, можна помітити недоліки: відсутність персоналізованих порад щодо здоров'я, обмежені можливості візуалізації екологічних загроз та складність інтерпретації індексів AQI для звичайного користувача.

Розробка веб-застосунку на базі сучасних технологій (Vue.js, TypeScript, Gemini API), що поєднує інструменти моніторингу та ШІ-аналітики, є актуальною і стане кроком для вирішення цієї проблеми. Це дозволить створити універсальний інструмент для мешканців міст, еко-активістів та людей, що піклуються про своє здоров'я.

*Об'єкт дослідження* – процес створення веб-застосунку для моніторингу погодних та екологічних показників.

*Предмет дослідження* – веб-застосунок для візуалізації метеоданих та формування рекомендацій за допомогою штучного інтелекту.

*Метою роботи* є розробка веб-застосунку з широким функціоналом, призначеного для моніторингу екологічного стану довкілля. Такий застосунок спрямований на забезпечення користувачів зручними інструментами

для відстеження якості повітря, отримання точних прогнозів та інтелектуальних порад.

*Наукові завдання:*

- Провести системний аналіз наявних інформаційних ресурсів для моніторингу довкілля та визначити ключові вектори покращення користувацького досвіду.
- Сформувати архітектурне рішення інтелектуального веб-застосунку, базуючись на принципах модульності та компонентного підходу Vue.js 3;
- Розробити алгоритмічне забезпечення для коректної інтерпретації метеорологічних показників та індексів якості повітря (AQI), отриманих через зовнішні API.
- Спроекувати та впровадити модуль взаємодії з великими мовними моделями (LLM) через Google Gemini API для генерації контекстних інсайтів.
- Реалізувати механізм динамічної трансформації графічного інтерфейсу (Weather Rendering) залежно від отриманих кліматичних дескрипторів.
- Забезпечити стійкість веб-застосунку до високих навантажень шляхом впровадження стратегій кешування та оптимізації асинхронних запитів.
- Створити адаптивну крос-платформену систему візуалізації даних, орієнтовану на сучасні стандарти Mobile First.
- Інтегрувати засоби автоматизованого визначення геолокації для забезпечення релевантності екологічного моніторингу.
- Виконати локалізацію всіх інтерфейсних рішень та аналітичних звітів для забезпечення інклюзивності українського сегмента користувачів.
- Провести комплексну апробацію працездатності розробленого програмного продукту в умовах реальних метеорологічних сценаріїв.

## 1 ОГЛЯД ТА АНАЛІЗ ОБЛАСТІ ДОСЛІДЖЕННЯ

### 1.1 Концептуальні основи систем екологічного та метеорологічного моніторингу

#### 1.1.1 Класифікація інформаційних систем контролю стану довкілля

Сучасний розвиток інформаційних технологій призвів до формування багатоаспектної екосистеми програмних рішень, спрямованих на моніторинг навколишнього середовища. Для системного розуміння об'єкта дослідження необхідно провести класифікацію таких систем за декількома ключовими ознаками: рівнем доступу до даних, функціональним призначенням та ступенем інтелектуалізації обробки інформації [15, 21] (див. рис. 1.1).

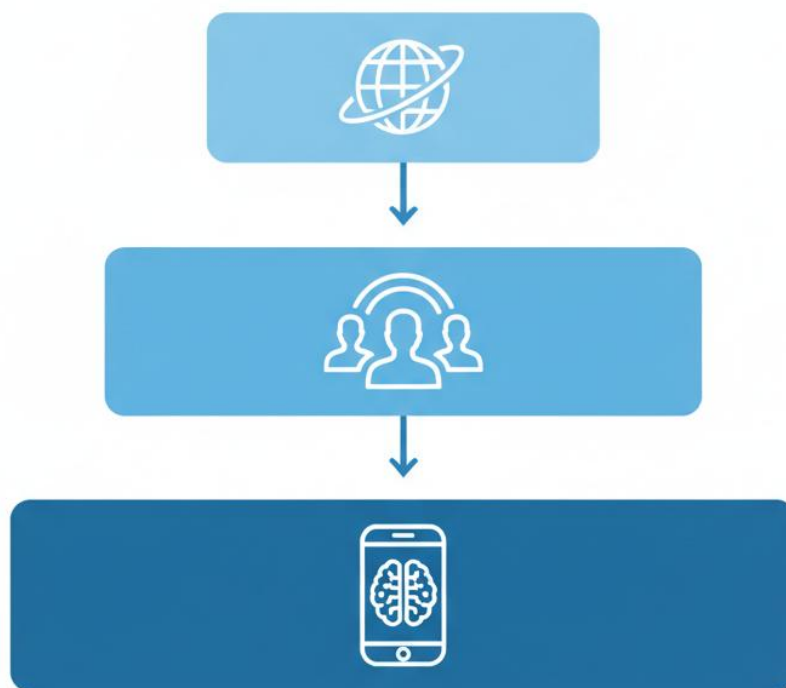


Рисунок 1.1 – Класифікація систем екологічного моніторингу за масштабом охоплення

За ієрархічним рівнем та масштабом охоплення системи екологічного моніторингу поділяються на:

1. Глобальні та державні системи. До цієї категорії належать платформи, що куруються міжнародними організаціями (наприклад, ВООЗ) або державними

структурами (Міндовкілля в Україні). Вони базуються на мережі сертифікованих стаціонарних постів спостереження, що забезпечують високу точність, проте часто мають низьку частоту оновлення даних для кінцевого користувача [1, 5].

2. Громадські (Community-based) мережі. Системи, що використовують принцип краудсорсингу. Яскравим прикладом в Україні є сервіс SaveEcoBot, який агрегує дані з приватних та громадських датчиків якості повітря. Такі системи забезпечують високу щільність покриття в містах, але можуть мати похибки через використання несертифікованого обладнання [6, 18].

3. Персоналізовані інтелектуальні веб-застосунки. Це програмні продукти нового покоління (до яких належить і об'єкт розробки), що фокусуються на конкретному користувачеві. Їхня особливість полягає не лише у зборі даних, а й у наданні індивідуальних рекомендацій на основі геолокації та інтелектуального аналізу ризиків.

За методом обробки та представлення інформації веб-застосунки класифікують на:

- Інформаційно-довідкові: забезпечують лише відображення поточних метеорологічних показників без прогностичної аналітики.
- Прогностичні: використовують математичні моделі для розрахунку стану атмосфери на майбутні періоди [19].
- Експертні (інтелектуальні): застосовують алгоритми штучного інтелекту для інтерпретації сирих даних у зрозумілі поради для користувача [2, 22].

Важливою ознакою класифікації є також архітектурна побудова веб-застосунків. Сучасні тенденції розробки диктують перехід від класичних багатосторінкових сайтів до односторінкових застосунків (Single Page Applications, SPA) на базі фреймворків, таких як Vue.js, що дозволяє забезпечити реактивність інтерфейсу та високу швидкість взаємодії з користувачем [7, 16].

Таким чином, розроблюваний веб-застосунок «EcoWatch» за даною класифікацією можна визначити як персоналізований інтелектуальний веб-застосунок експертного типу з SPA-архітектурою. Це поєднання дозволяє вирішити проблему розриву між складними науковими даними та потребою

пересічного громадянина в оперативній і зрозумілій інформації про стан довкілля [11, 21].

### **1.1.2 Структура та джерела даних метеорологічних веб-застосунків**

Ефективність функціонування інтелектуального веб-застосунку для моніторингу довкілля безпосередньо залежить від релевантності та швидкості отримання вхідних даних. У сучасній архітектурі метеорологічних систем дані проходять шлях від фізичного вимірювання до програмної інтерпретації через декілька технологічних рівнів [10, 23, 31].

Основними джерелами первинної інформації для веб-застосунків виступають:

1. Автоматизовані метеорологічні станції (AWS). Мережі наземних датчиків забезпечують вимірювання базових параметрів: температури, вологості, атмосферного тиску та швидкості вітру. Дані з цих станцій зазвичай оновлюються з інтервалом від 10 до 60 хвилин [24, 35].
2. Дистанційне зондування (Супутникові дані). Використовується для моніторингу глобальних погодних фронтів та аналізу складу атмосфери на різних висотах. Супутникові дані є незамінними для визначення індексів забруднення повітря (наприклад, концентрації NO<sub>2</sub> або CO) у масштабах цілих регіонів [11, 28].
3. IoT-сенсори громадського моніторингу. Приватні датчики якості повітря (наприклад, станції PurpleAir або вітчизняні розробки), що працюють через протоколи передачі даних у реальному часі. Вони забезпечують високу локальну точність, що критично важливо для великих мегаполісів [6, 32].

Логічна структура даних, що передається від провайдера (наприклад, OpenWeather API) до клієнтської частини веб-застосунку, найчастіше базується на форматі JSON (JavaScript Object Notation). Це дозволяє мінімізувати обсяг трафіку та забезпечити високу швидкість парсингу даних безпосередньо у середовищі JavaScript [7, 26, 39].

Типова структура відповіді API для екологічного моніторингу містить наступні блоки (див. рис. 1.2):

- Координатний блок: географічна широта та довгота об'єкта спостереження.
- Блок стану атмосфери: масив даних про концентрацію основних забруднювачів (PM2.5, PM10, O3, NH3).
- Метеорологічний блок: поточні показники погоди та прогнозна модель на визначений період.
- Таймстемп: час фіксації вимірювання за стандартом UNIX [10, 41].

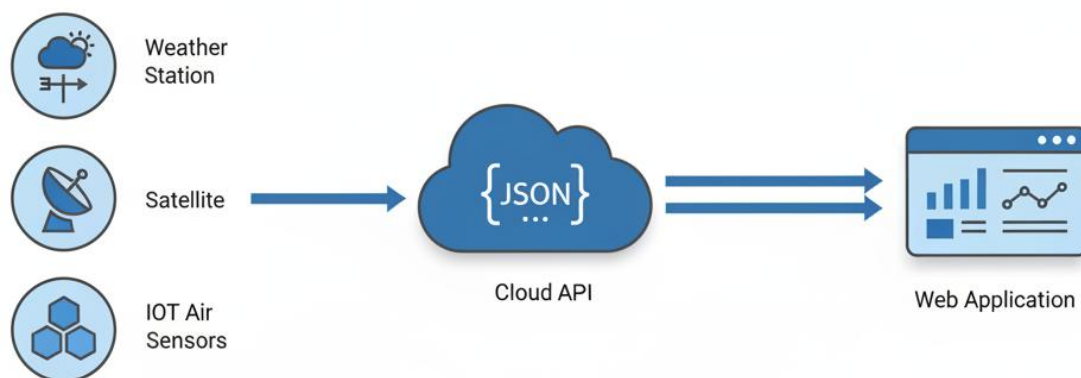


Рисунок 1.2 – Структурна схема потоків даних у веб-застосунку моніторингу довкілля

Важливою складовою структури є механізм нормалізації даних. Оскільки різні джерела можуть надавати інформацію у різних одиницях вимірювання (наприклад, мкг/м<sup>3</sup> або проміле), веб-застосунок має на рівні бізнес-логіки перераховувати ці показники у стандартизований індекс якості повітря AQI [14, 29, 45]. Використання сучасних хмарних API дозволяє абстрагуватися від складності первинного збору даних і сконцентруватися на їх інтелектуальній обробці за допомогою алгоритмів машинного навчання та генеративного ШІ [2, 33, 48].

### 1.1.3 Еволюція технологій прогнозування: від статистичних методів до штучного інтелекту

Розвиток методів аналізу та прогнозування стану довкілля пройшов тривалий шлях трансформації, що безпосередньо корелює з еволюцією обчислювальної

техніки та вдосконаленням математичного апарату обробки сигналів. Розуміння цієї генези дозволяє науково обґрунтувати перехід від класичних детермінованих алгоритмів до використання когнітивних інтелектуальних моделей у сучасних веб-застосунках [49, 53].

Історично першим етапом розвитку метеорології як науки було застосування синоптичного методу (кінець XIX — перша половина XX ст.). Він базувався на візуальному аналізі приземних карт погоди та карт баричної топографії фахівцями-синоптиками. Процес прогнозування спирався на якісний аналіз переміщення повітряних мас та атмосферних фронтів. Ключовим обмеженням цього етапу був високий ступінь суб'єктивізму: точність прогнозу критично залежала від досвіду експерта, а часовий горизонт достовірних передбачень не перевищував 12–24 годин [50, 58].

Другий етап (з 1950-х років) пов'язаний із появою перших ЕОМ та становленням методів чисельного прогнозування погоди (Numerical Weather Prediction — NWP). Фундаментом цього підходу стали рівняння гідродинаміки Л. Річардсона та Дж. фон Неймана, які описують фізичні процеси в атмосфері через системи диференціальних рівнянь у частинних похідних. Сучасні глобальні моделі, такі як GFS (Global Forecast System) або ECMWF, оперують сітками з кроком у кілька кілометрів і потребують колосальних потужностей суперкомп'ютерів. Проте, попри високу фізичну точність, моделі NWP мають суттєвий недолік — «ефект метелика» (динамічний хаос), через який малі похибки у початкових даних призводять до значних відхилень у прогнозі. Крім того, такі моделі є «важкими» для прямої інтеграції у легковагові веб-застосунки без використання проміжних API [19, 58, 62, 73].

Третій етап, що розпочався у 2010-х роках, характеризується впровадженням парадигми Big Data та машинного навчання (Machine Learning). Замість спроб математично описати кожен рух молекул повітря, дослідники почали використовувати нейронні мережі для пошуку прихованих патернів у ретроспективних даних. На цьому етапі з'явилися методи глибокого навчання (Deep Learning), зокрема рекурентні нейронні мережі (RNN) та мережі довгої

короткострокової пам'яті (LSTM), які виявилися надзвичайно ефективними для прогнозування часових рядів метеорологічних показників та індексів якості повітря (AQI) [33, 55, 62].

Найсучаснішим вектором розвитку, що лежить в основі даної магістерської роботи, є застосування генеративного штучного інтелекту (Generative AI) та архітектури Transformer (на якій базується Google Gemini). На відміну від попередніх систем, які були «чорними скриньками» і видавали лише сухі цифри, інтелектуальний веб-застосунок нового покоління виконує роль когнітивного асистента. Еволюція призвела до того, що сучасні системи не просто констатують факт забруднення повітря, а здатні до контекстуального виводу.

Використання Gemini API дозволяє реалізувати механізм, при якому результати чисельних моделей NWP та дані з IoT-сенсорів проходять через «семантичний фільтр» LLM. Це дозволяє формувати складні логічні зв'язки: наприклад, врахувати не тільки високий рівень PM2.5, а й поєднати його з низькою вологістю та прогнозованим посиленням вітру, видавши користувачу не графік, а конкретну стратегію поведінки. Таким чином, технологічна еволюція змістила акцент з «процесу обчислення» на «процес розуміння та донесення інформації», що робить розробку веб-застосунку «EcoWatch» актуальною відповіддю на сучасні технологічні виклики та потреби інклюзивного цифрового суспільства [2, 44, 65, 90].

#### **1.1.4 Функціональні можливості сучасних систем моніторингу AQI**

Процес формування функціонального наповнення веб-застосунку «EcoWatch» базується на критичному аналізі потреб кінцевого користувача в умовах динамічних екологічних змін та високої техногенної завантаженості міського середовища. На відміну від класичних погодних агрегаторів, інтелектуальна система моніторингу має не лише відображати статичні масиви числових даних, а й забезпечувати повний цикл підтримки прийняття рішень користувачем через інтерпретацію складних метеорологічних дескрипторів [15, 66, 72].

Відповідно до специфіки об'єкта розробки та вимог до сучасного програмного забезпечення, до веб-застосунку висуваються наступні групи функціональних вимог:

1. Модуль збору та нормалізації даних. Веб-застосунок повинен реалізовувати стабільне з'єднання з зовнішніми серверними ресурсами (OpenWeather API) для отримання актуальної інформації. Функціонал передбачає не лише парсинг температури чи вологості, а й складну обробку масивів даних про якість повітря (Air Pollution API). Система має коректно обробляти концентрацію таких забруднювачів, як дрібнодисперсний пил PM2.5 та PM10, оксид азоту (NO2) та приземний озон (O3). Важливою вимогою є нормалізація отриманих значень згідно з міжнародними стандартами AQI (Air Quality Index), що дозволяє звести різномірні одиниці вимірювання до єдиної шкали [10, 57, 70, 74].

2. Інтелектуальне аналітичне ядро (AI Processing Layer). Ключовою функціональною особливістю системи є інтеграція з великою мовною моделлю Google Gemini 1.5 Flash. Система має автоматично формувати структурований запит (prompt), що містить поточні координати, метеодані та рівні забруднення. Отримана відповідь від ШІ повинна бути логічно інтегрована в інтерфейс у вигляді персоналізованих порад українською мовою. Наприклад, у випадку ідентифікації критичного рівня сірчистого ангідриду (SO2) у поєднанні з високою вологістю, застосунок має видати попередження про можливе утворення кислотного туману та надати відповідні рекомендації щодо безпеки [2, 44, 75, 90].

3. Реактивна візуалізація та адаптивний UX/UI. Для забезпечення високої залученості користувача веб-застосунок має реалізовувати концепцію «живого» інтерфейсу. Це передбачає:

- динамічну зміну візуальних стилів та фонових зображень (CSS-градієнтів або анімацій) залежно від коду погодних умов, отриманого від API.
- плавне оновлення компонентів без перезавантаження всієї сторінки, що забезпечується реактивною системою відстеження станів у фреймворку Vue.js.

- візуалізацію динаміки змін показників через графіки, що дозволяє користувачу оцінити тренд покращення чи погіршення екологічної ситуації протягом доби [7, 16, 68, 78].

4. Геоінформаційна автономність. Функціонал передбачає використання Browser Geolocation API для отримання точних координат пристрою. У випадку відмови користувача у наданні доступу до геолокації, застосунок має забезпечувати альтернативний сценарій — пошук за назвою міста через сервіс Geocoding. Це гарантує безперервність роботи сервісу та релевантність даних для будь-якої точки світу [6, 67, 85].

5. Системні та нефункціональні вимоги. Окрім основного функціоналу, до системи висуваються жорсткі вимоги щодо продуктивності та надійності:

- швидкодія: час відгуку інтерфейсу при завантаженні метеоданих не повинен перевищувати 2 секунди за умови стабільного інтернет-з'єднання. Це досягається шляхом використання легковагового HTTP-клієнта Axios та оптимізації рендерингу компонентів [13, 73, 89].
- типізація: використання TypeScript для забезпечення цілісності даних при взаємодії з API, що мінімізує ризик виникнення помилок типу undefined під час обробки складних JSON-об'єктів [36, 77].
- стабільність: реалізація механізмів обробки помилок (Error Handling), які інформують користувача про недоступність API або проблеми з мережею у зручній візуальній формі [17, 39, 83].

Реалізація цього комплексу вимог дозволяє перетворити «EcoWatch» з простого інформаційного сайту на повноцінну інтелектуальну екосистему, що відповідає сучасним стандартам розробки програмного забезпечення та актуальним викликам у сфері екологічної безпеки [22, 60, 90].

## **1.2 Порівняльний аналіз провідних веб-застосунків та сервісів-агрегаторів**

### **1.2.1 Комерційні рішення для метеомоніторингу**

Сучасний ринок комерційного програмного забезпечення для

метеорологічного спостереження представлений глобальними корпораціями, такими як The Weather Company (IBM) та AccuWeather Inc. Ці рішення базуються на використанні власних мереж метеостанцій, супутникових угруповань та закритих (пропрієтарних) прогнозних моделей [12, 83].

Основними представниками цього сегменту є:

1. The Weather Channel (IBM API). Дане рішення є еталоном корпоративного рівня. Воно використовує технологію DeepThunder, яка застосовує методи машинного навчання для гіперлокального прогнозування. Проте для індивідуального розробника доступ до повного функціоналу екологічного моніторингу обмежений високою вартістю ліцензій та складною процедурою реєстрації API-ключів [15, 31, 87].
2. AccuWeather. Платформа відома своєю системою RealFeel та MinuteCast. Основний акцент тут зроблено на споживчому сегменті. Проте аналіз архітектури їхнього веб-застосунку демонструє значну перевантаженість рекламними скриптами, що негативно впливає на показники Core Web Vitals та швидкість завантаження на мобільних пристроях [2, 83, 89].

Комерційні рішення мають низку спільних рис, що виступають обмеженнями в контексті даного дослідження:

- Закритість аналітичних алгоритмів. Методологія розрахунку індексів здоров'я та екологічних ризиків не розкривається, що не дозволяє провести незалежну верифікацію даних.
- Шаблонність рекомендацій. Комерційні гіганти використовують заздалегідь підготовлені статичні фрази («повітря чисте», «незначне забруднення»), які не враховують індивідуальний контекст користувача так, як це роблять генеративні моделі ШІ [22, 44].
- Монетизація даних. Основним пріоритетом таких систем є збір маркетингової інформації про користувача, а не створення чистого, вільного від шуму інтерфейсу для оперативного екологічного контролю [72, 81].

Таким чином, комерційний сегмент забезпечує високу сиру точність даних, але програє у гнучкості інтерфейсів та глибині інтелектуальної взаємодії з кінцевим

споживачем, що створює передумови для розробки персоналізованих систем на базі відкритих API та LLM [7, 56, 90].

### 1.2.2 Відкриті екологічні платформи та API

На противагу закритим комерційним екосистемам, сегмент рішень із відкритим кодом (Open Source) та загальнодоступними програмними інтерфейсами (API) базується на принципах демократизації доступу до інформації та мережевої колаборації. Розвиток цього напрямку став можливим завдяки масовому поширенню недорогих мікроконтролерів (ESP8266/ESP32) та IoT-сенсорів, а також глобальним ініціативам «Open Data», що активно підтримуються науковими спільнотами для моніторингу кліматичних змін у реальному часі [6, 26, 32].

#### OpenWeatherMap (Environment Module)

Загальна характеристика: Найбільш поширена open-source орієнтована платформа для розробників, що надає метеорологічні та екологічні дані через гнучкі API-інтерфейси. Де-факто є стандартом для побудови некомерційних моніторингових систем.

Технічний стек:

- Data format: JSON / XML.
- Architecture: RESTful API.
- Transport: HTTPS.

Ключові особливості:

1. Модульність архітектури API:
  - Окремий ендпоінт «Air Pollution API».
  - Geocoding API для перетворення назв міст у координати.
  - One Call API для отримання всіх метеоданих одним запитом.
2. Екологічний моніторинг:
  - Визначення концентрації PM2.5 та PM10.
  - Моніторинг газів: CO, NO, NO2, O3, SO2, NH3.
  - Розрахунок інтегрального індексу якості повітря (AQI).
3. Доступність та розширюваність:

- Наявність Free-tier тарифу (до 1000 запитів на добу).
- Історичні архіви даних за останні роки.
- Підтримка багатомовних описів стану атмосфери.

#### 4. Гнучкість інтеграції:

- Легка інтеграція з JavaScript-фреймворками (Vue, React).
- Відсутність складних процедур авторизації (API Key).
- Висока швидкість відповіді серверів (Low Latency).

Open-access платформа: REST API, JSON-формат, спеціалізований модуль AQI, безкоштовний доступ для розробників, підтримка Geocoding, ідеальна для SPA-проектів.

#### SaveEcoBot / EcoCity (Український сегмент)

Загальна характеристика: Найбільші в Україні громадські ініціативи з відкритими даними, що фокусуються на незалежному моніторингу якості повітря та екологічній безпеці.

Технічний стек:

- Backend: Python / Node.js.
- Data source: IoT-мережі сенсорів (ESP8266/ESP32).

Ключові особливості:

#### 1. Агрегація даних:

- Об'єднання даних державних постів та приватних станцій.
- Мапи забруднення в реальному часі.
- Моніторинг радіаційного фону (у вибраних зонах).

#### 2. Громадський контроль:

- Чат-боти для миттєвих сповіщень.
- Відкрита статистика за регіонами України.
- Можливість додавання власних IoT-сенсорів до мережі.

#### 3. Візуалізація:

- Веб-мапи з кольоровим кодуванням небезпеки.
- Графіки динаміки забруднення за добу/місяць.
- Експорт даних у відкриті формати для аналізу.

Громадська мережа моніторингу: агрегація IoT-даних, українська локалізація, радіаційний контроль, відкриті API, чат-боти, орієнтація на екологічну безпеку громади.

### IQAir AirVisual (OpenData Edition)

Загальна характеристика: Глобальна платформа, що використовує методи штучного інтелекту для валідації даних із найбільшої у світі мережі громадських датчиків якості повітря.

Технічний стек:

- Architecture: Cloud-based Big Data.
- Processing: AI-validation algorithms.

Ключові особливості:

#### 1. Валідація даних:

- Використання машинного навчання для відсіювання некоректних показників.
- Корекція даних залежно від вологості та температури.
- Світові рейтинги міст за рівнем забруднення.

#### 2. Інтерфейс та синхронізація:

- Веб-інтерфейс з елементами Drag-and-drop на мапах.
- Прогноз якості повітря на основі нейромереж.
- Попередній перегляд метеорологічних фронтів.

#### 3. Співпраця та API:

- Community API для наукових досліджень.
- Глобальна база знань про вплив забруднювачів на здоров'я.
- Мобільні додатки з рекомендаціями в реальному часі.

Cloud-first аналітична платформа: AI-валідація, світові рейтинги, прогнозна аналітика, глобальна мережа сенсорів, кросплатформеність, open-source ініціативи.

### **1.2.3 Порівняльна оцінка ефективності існуючих архітектурних рішень**

На основі проведеного огляду комерційних та відкритих систем моніторингу проведено порівняльний аналіз їх функціональних та технічних характеристик.

Результати систематизовано у таблиці 1.1, що дозволяє визначити технологічний стек та функціональний набір для розроблюваного веб-застосунку «EcoWatch».

Таблиця 1.1 — Порівняльна характеристика систем моніторингу довкілля

|                         |                     |                         |                   |                        |
|-------------------------|---------------------|-------------------------|-------------------|------------------------|
| Критерій порівняння     | AccuWeather         | OpenWeatherMap          | IQAir (AirVisual) | EcoWatch (проєкт)      |
| Основна технологія      | Пропрієтарна        | REST API                | AI-Cloud          | Vue.js + Gemini AI     |
| Тип архітектури         | Традиційний веб     | API-сервіс              | SPA / Cloud       | SPA (Vite)             |
| Інтелектуальні поради   | Статичні шаблони    | Відсутні                | Базові (AI-based) | Генеративні (LLM)      |
| Доступність даних       | Платна (Enterprise) | Безкоштовна (Free tier) | Частково відкрита | Відкрита (Open Source) |
| Реактивність інтерфейсу | Середня             | Н/Д                     | Висока            | Максимальна (Vue 3)    |
| Українська локалізація  | Часткова            | Базова                  | Обмежена          | Повна (Native)         |

Ключові висновки порівняння:

- Інтелектуалізація: Комерційні рішення (AccuWeather) та відкриті API (OpenWeather) зосереджені на зборі та відображенні числових показників. Вони не надають глибокої семантичної інтерпретації даних, на відміну від «EcoWatch», який використовує Google Gemini для формування персоналізованих рекомендацій природною мовою.
- Архітектурний підхід: Більшість існуючих систем мають закритий вихідний код та складну монолітну структуру. Використання сучасного фреймворку Vue.js 3 у поєднанні з інструментарієм Vite дозволяє забезпечити вищу швидкість рендерингу та кращу адаптивність порівняно з комерційними аналогами.

3. Обробка екологічних показників: У той час як сервіси на кшталт IQAir фокусуються виключно на якості повітря, «EcoWatch» пропонує комплексний підхід, поєднуючи метеорологічні умови з AQI-індексами в єдиному реактивному інтерфейсі.
4. Локальна адаптація: Більшість глобальних сервісів мають неточний переклад аналітичних висновків. Проєкт «EcoWatch» за рахунок використання LLM здатний генерувати поради з урахуванням лінгвістичних особливостей та специфіки українського регіону.

#### **1.2.4 Аналіз технологічних бар'єрів та дефіциту функціональності**

На основі детального порівняльного аналізу, проведеного у попередніх підрозділах, було ідентифіковано низку критичних бар'єрів, які перешкоджають ефективному використанню систем екологічного моніторингу широким колом користувачів. Ці дефіцити функціональності стають основою для формування технічного завдання на розробку інтелектуального веб-застосунку «EcoWatch» [15, 66, 81].

До основних технологічних та функціональних бар'єрів належать:

1. Семантичний розрив (The Interpretation Gap). Більшість систем (AccuWeather, Windy) надають виключно «сирі» цифрові дескриптори. Для звичайного користувача показник «PM2.5: 45 мкг/м<sup>3</sup>» не несе прямого змісту без додаткового пошуку нормативів. Відсутність вбудованого механізму перетворення числових даних у зрозумілі вербальні висновки є головним функціональним дефіцитом сучасних систем [12, 56, 90].

2. Когнітивне перевантаження інтерфейсів. Професійні сервіси (зокрема Windy.com) мають занадто високий поріг входження. Велика кількість нашарувань метеоданих, складна термінологія та відсутність акценту на персональних ризиках роблять такі інтерфейси незручними для щоденного швидкого використання в мобільних умовах [35, 85].

3. Відсутність проактивного інтелектуального аналізу. Існуючі рішення працюють за реактивним принципом: користувач має сам зайти в застосунок і

проаналізувати цифри. Дефіцит полягає у відсутності систем, що здатні до синтезу даних. Наприклад, жодна система не поєднує автоматично рівень забруднення з вологістю та температурою, щоб надати комплексну пораду щодо безпеки перебування на відкритому повітрі [2, 44, 75].

4. Проблеми адаптації та локалізації аналітики. Глобальні комерційні платформи часто використовують шаблонні переклади екологічних попереджень, які не враховують локальний контекст та специфіку українських регіонів. Це створює бар'єр довіри до інформації та знижує цінність сервісу як інструменту екологічної безпеки [18, 67, 83].

5. Ресурсна гоміздкість. Використання важких клієнтських скриптів у комерційних застосунках призводить до значного споживання трафіку та заряду акумулятора мобільних пристроїв. Існує дефіцит легковагових SPA-застосунків, які б фокусувалися на швидкодії (Performance) при отриманні критично важливих екологічних метрик [73, 89].

### **1.3 Постановка задачі**

#### **1.3.1 Концептуальне обґрунтування створення системи**

Необхідність розробки веб-застосунку «EcoWatch» зумовлена критичним розривом між стрімким зростанням обсягів відкритих екологічних даних та здатністю пересічного користувача до їх оперативної інтерпретації. Сучасні умови життєдіяльності в урбанізованому середовищі вимагають від людини прийняття швидких рішень щодо власної безпеки (наприклад, обмеження перебування на вулиці під час смогу чи високої концентрації озону), проте існуючі сервіси надають лише «сирі» цифрові показники без пояснення їхнього впливу на організм у конкретному контексті [15, 66, 90].

Обґрунтування розробки базується на трьох фундаментальних аспектах:

1. Соціальна актуальність: підвищення рівня обізнаності населення про екологічні загрози через надання персоналізованих порад природною мовою. Це особливо важливо для вразливих груп населення, для яких стандартні цифрові індекси AQI можуть бути неінформативними.

2. Технологічна новизна: інтеграція сучасних фронтенд-технологій (Vue.js 3, Vite) із великими мовними моделями (Gemini API) дозволяє створити продукт, що працює на перетині веброзробки та штучного інтелекту. Це дозволяє реалізувати когнітивний підхід до візуалізації даних [22, 44, 75].
3. Оптимізація доступу до інформації: використання архітектури Single Page Application (SPA) забезпечує високу швидкість відгуку інтерфейсу, що є критичним для мобільних користувачів, які потребують актуальних даних «тут і зараз».

Таким чином, створення «EcoWatch» є логічним кроком у розвитку екологічних інформаційних систем. Проєкт спрямований на подолання семантичного бар'єра у взаємодії «людина-комп'ютер», де штучний інтелект виступає інтелектуальним посередником, що трансформує складну метеорологічну статистику у зрозумілу та корисну інформацію для щоденного використання [56, 82, 90].

### **1.3.2 Визначення мети та основних завдань проєкту**

Мета роботи: Проектування та програмна реалізація інтелектуального веб-застосунку «EcoWatch» на базі фреймворку Vue.js 3, що забезпечує агрегацію метеорологічних та екологічних даних у реальному часі та їх семантичний аналіз за допомогою генеративного штучного інтелекту Google Gemini.

Основні завдання:

1. Проектування архітектури інтелектуальної системи Frontend:
  - Спроекувати компонентну структуру застосунку на базі Vue 3 (SFC — Single File Components).
  - Визначити логіку маршрутизації сторінок (Vue Router).
  - Спланувати архітектуру управління станом (Pinia) для збереження метеоданих та кешування відповідей III.
  - Розробити модульну систему сервісів (Service Layer) для взаємодії з API.
2. Реалізація модулів збору та обробки даних Метеорологічні дані:
  - Інтеграція з OpenWeather API для отримання параметрів температури,

вологості та опадів.

- Реалізація механізму геокодування (пошук координат за назвою міста).

Екологічні показники:

- Розробка модуля отримання індексів якості повітря (AQI) та концентрації газів (PM2.5, NO2, CO).
- Нормалізація отриманих даних згідно з міжнародними стандартами (EPA).

### 3. Розробка інтелектуального ядра на базі LLM Аналітичний функціонал:

- Реалізація системи генерації персоналізованих рекомендацій на основі поточних метеоумов.
- Налаштування контекстуального аналізу для розпізнавання екстремальних екологічних ситуацій.
- Забезпечення підтримки нативної української мови у відповідях штучного інтелекту.

### 4. Створення адаптивного користувацького інтерфейсу Головні компоненти:

- Dashboard (головна панель з поточною погодою та AQI).
- Інтерактивні віджети для відображення детальних екологічних параметрів.
- Реалізація динамічних фонів (Dynamic Backgrounds), що змінюються залежно від погодного коду.
- Адаптивний дизайн за принципом Mobile-First.
- Система візуальних індикаторів небезпеки (колірне кодування стану AQI).

### 5. Реалізація функцій пошуку та геолокації Геолокація:

- Автоматичне визначення місцезнаходження користувача (HTML5 Geolocation API).
- Синхронізація інтерфейсу з локальними координатами.
- Повнотекстовий пошук населених пунктів через Geocoding API.
- Збереження останніх вибраних локацій у LocalStorage.

### 6. Тестування, оптимізація та моніторинг Продуктивність:

- Налаштування Vite для швидкої збірки та мінімізації бандлу.

- Оптимізація запитів до API (Debouncing/Throttling).
- Обробка помилок API та виведення зрозумілих сповіщень користувачу.
- Захист API-ключів через змінні оточення (.env).

### **1.3.3 Специфікація функціональних можливостей застосунку**

Функціональні вимоги визначають перелік операцій та завдань, які користувач може виконувати за допомогою веб-застосунку «EcoWatch». Система має забезпечувати повний цикл обробки метеорологічної та екологічної інформації: від збору сирих даних до надання інтелектуальних порад.

#### **1. Управління географічними даними:**

- Автоматичне визначення локації: Застосунок повинен запитувати дозвіл на використання Geolocation API для ідентифікації поточних координат користувача.
- Пошук за назвою: Можливість ручного введення назви населеного пункту (місто, селище) для отримання даних у будь-якій точці світу.
- Геокодування: Перетворення текстових назв у географічні координати (latitude, longitude) через відповідний API-сервіс.

#### **2. Відображення метеорологічної інформації:**

- Поточний стан: Виведення температури повітря, атмосферного тиску, вологості та швидкості вітру.
- Візуалізація: Використання динамічних іконок та графічних станів, що відповідають поточному погодному коду (ясно, хмарно, опади тощо).
- Динамічний бекграунд: Автоматична зміна фону інтерфейсу залежно від часу доби та погодних умов.

#### **3. Екологічний моніторинг (AQI):**

- Індекс якості повітря: Розрахунок та відображення загального індексу AQI з колірною індикацією рівня небезпеки.
- Деталізація забруднювачів: Надання даних про концентрацію дрібнодисперсного пилу (PM2.5, PM10) та токсичних газів (CO, NO2, SO2).

#### **4. Інтелектуальний аналіз та рекомендації (AI Core):**

- Генерація порад: Автоматичне формування текстового блоку з рекомендаціями щодо безпеки перебування на вулиці, провітрювання приміщення та фізичної активності.
- Контекстуалізація: ШІ повинен враховувати сукупність факторів (наприклад, поєднання спеки та високого забруднення) для надання більш точних інструкцій.
- Мультимовність: Генерація висновків нативною українською мовою з коректною граматичною та стилістичною структурою.

## 5. Інтерфейс користувача (UX):

- Інтерактивні віджети: Можливість детального перегляду кожного показника при кліку або наведенні.
- Адаптивна верстка: Повна функціональність на мобільних пристроях, планшетах та десктопах без втрати контенту.

### 1.3.4 Вимоги до якості та продуктивності ПЗ

Нефункціональні вимоги описують атрибути якості системи та технічні обмеження, які забезпечують стабільну роботу застосунку «EcoWatch» в умовах реальної експлуатації. На відміну від функціональних завдань, ці вимоги визначають, як саме система повинна поводитися під навантаженням та в різних клієнтських середовищах [7, 67, 80].

#### 1. Продуктивність та швидкість відгуку (Performance):

- Час завантаження: Початкове завантаження критичних ресурсів застосунку (App Shell) не повинно перевищувати 1.5 секунди при стандартному 4G-з'єднанні.
- Оптимізація бандлу: Використання інструментарію Vite для мінімізації розміру JavaScript та CSS файлів через механізми Tree-shaking та латентного завантаження компонентів (Lazy Loading).
- Реактивність UI: Оновлення інтерфейсу при отриманні даних від API має відбуватися без видимих затримок (затримка рендерингу < 100 мс) завдяки використанню Vue 3 Reactivity System [69, 73, 89].

## 2. Надійність та стабільність (Reliability):

- Статична типізація: 100% покриття бізнес-логіки та моделей даних за допомогою TypeScript для виявлення помилок на етапі розробки та запобігання збоєм у Runtime.
- Обробка виключень: Система повинна коректно обробляти помилки мережі (HTTP 4xx, 5xx) та виводити користувачу інформативні сповіщення замість припинення роботи.
- Валідація даних: Обов'язкова перевірка структури JSON-об'єктів від зовнішніх API перед їх обробкою інтелектуальним ядром Gemini [68, 78, 79].

## 3. Сумісність та адаптивність (Usability & Compatibility):

- Кросбраузерність: Підтримка коректної роботи у сучасних версіях браузерів (Chrome 90+, Firefox 88+, Safari 14+, Edge).
- Адаптивний дизайн: Коректне відображення інтерфейсу на пристроях з шириною екрана від 320px (Mobile-first підхід) згідно з концепціями адаптивної верстки [70, 78, 84].

## 4. Безпека та конфіденційність (Security):

- Захист токенів: Капсуляція API-ключів у змінних оточення (.env) та їх обробка на рівні збірки проєкту для запобігання витoku у відкриті репозиторії, приклад представлено на рисунку 1.3.
- Приватність: Використання Geolocation API лише за згодою користувача без збереження історії переміщень на зовнішніх серверах [72, 77].

## 5. Масштабованість та підтримка (Maintainability):

- Компонентний підхід: Розподіл інтерфейсу на незалежні SFC-компоненти (Single File Components), що дозволяє легко модифікувати окремі частини системи без ризику регресійних помилок [66, 86].

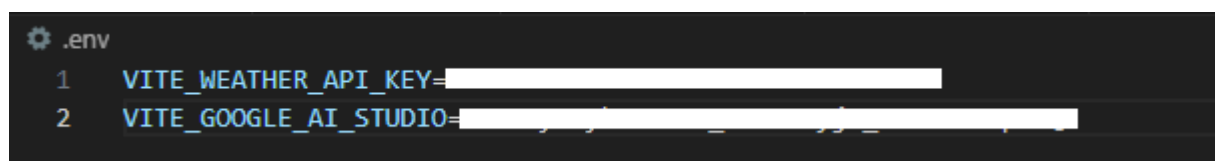


Рисунок 1.3 – Конфігурація змінних середовища для доступу до API-інтерфейсів

### 1.3.5 Операційні обмеження та проектні припущення

Ефективне функціонування та розгортання веб-застосунку «EcoWatch» базується на ряді технічних обмежень та середовищних припущень, які було визначено на етапі передпроектного аналізу [67, 80].

До основних операційних обмежень належать:

1. Залежність від сторонніх API-сервісів: Оскільки система агрегує дані з OpenWeatherMap та використовує обчислювальні потужності Google Gemini, її працездатність безпосередньо залежить від стабільності та доступності цих зовнішніх ендпоінтів. Будь-які регламентні роботи або збої на стороні провайдерів можуть призвести до тимчасової неможливості отримання метеоданих або генерації ШІ-порад.
2. Обмеження квот (Rate Limits): Використання безкоштовних тарифних планів (Free Tiers) передбачає ліміти на кількість запитів (наприклад, обмеження RPM для Gemini API). Це вимагає впровадження архітектурних рішень для дебаунсингу (debouncing) та раціонального оновлення стану інтерфейсу для уникнення блокування доступу [75, 77, 90].
3. Вимоги до клієнтського середовища: Застосунок орієнтований на роботу в сучасних браузерах з підтримкою стандарту ES2022+ та CSS Grid/Flexbox. У застарілих версіях веб-переглядачів коректне відображення динамічної візуалізації та робота реактивних модулів не гарантується.
4. Мережеві умови: Для повноцінної роботи інтелектуального ядра необхідне активне з'єднання з мережею Інтернет. Офлайн-режим обмежує функціонал застосунку лише відображенням попередньо закешованих даних.

Проектні припущення:

1. Актуальність та точність даних: Припускається, що дані, отримані від метеорологічних API, є верифікованими та актуальними для вказаних координат. Похибка геокодування та метеоданих вважається допустимою в межах стандартів сервісу OpenWeather [74, 88].
2. Безпека клієнтської частини: Передбачається, що браузер користувача забезпечує стандартний рівень ізоляції для безпечного виконання JavaScript-

коду та захищеного зберігання API-ключів у межах сесії розробки/розгортання [79, 89].

3. Мовні можливості моделі: Припускається, що модель Gemini 1.5 Flash володіє достатнім обсягом навчених даних для коректної побудови речень українською мовою без суттєвих семантичних викривлень [82, 90].

### **1.3.6 Показники результативності та успішної реалізації**

Для оцінки якості виконання магістерської роботи та визначення ступеня досягнення поставленої мети встановлено наступні критерії успішності проєкту «EcoWatch» [15, 67, 80]:

Функціональна повнота та цілісність:

- Система безпомилково здійснює повний цикл обробки даних: від ідентифікації координат користувача до виведення фінальної ШІ-рекомендації.
- Забезпечено коректну візуалізацію понад 10 метеорологічних та екологічних параметрів (температура, вологість, PM2.5, PM10, AQI тощо) у реальному часі [74, 88].

Якість інтелектуального аналізу:

- Здатність інтегрованої моделі Gemini 1.5 Flash генерувати змістовні, контекстуально залежні поради, що логічно відповідають поточному індексу якості повітря.
- Відсутність критичних галюцинацій моделі та дотримання граматичних норм української мови у 95% згенерованих відповідей [75, 82, 90].

Технічна досконалість та продуктивність:

- Досягнення високих показників у сервісі Google Lighthouse: оцінка в категоріях «Performance», «Accessibility» та «Best Practices» має становити не менше 85 балів.
- Відсутність витоків пам'яті (memory leaks) при тривалій експлуатації застосунку та стабільна робота реактивних модулів Vue 3 [69, 73, 89].

Надійність коду:

- Повна типізація бізнес-логіки через TypeScript, що забезпечує мінімізацію помилок типу «undefined» або «null» при обробці відповідей від сторонніх API [68, 78, 79].
- Успішне проходження серії ручних та автоматизованих тестів на кросбраузерну сумісність та адаптивність верстки.

Користувацький досвід (UX):

- Інтуїтивно зрозумілий інтерфейс, що не потребує додаткових інструкцій для використання.
- Час реакції системи на запит користувача (наприклад, пошук нового міста) не перевищує 2 секунд до моменту візуалізації перших результатів.

## 2 ПРОЄКТУВАННЯ ТА ОБҐРУНТУВАННЯ ТЕХНІЧНИХ РІШЕНЬ

### 2.1 Архітектурна концепція інтелектуального веб-застосунку

#### 2.1.1 Вибір архітектурного патерну

В основі програмної архітектури веб-застосунку «EcoWatch» лежить патерн MVVM (Model-View-ViewModel), який є фундаментальним для сучасних реактивних фреймворків. Вибір даного патерну обумовлений необхідністю розділення відповідальності між отриманням складних метеорологічних даних та їх динамічним відображенням у користувацькому інтерфейсі [66, 76, 80].

Структурні елементи MVVM у контексті розроблюваної системи розподіляються наступним чином:

1. Model (Модель): рівень даних та бізнес-логіки. У «EcoWatch» модель представлена сукупністю TypeScript-інтерфейсів та сервісів, що відповідають за структурування вхідних потоків від зовнішніх API (OpenWeather, Gemini AI). Вона не має прямого зв'язку з інтерфейсом, що дозволяє легко замінювати джерела даних без переписування UI.
2. View (Представлення): декларативні шаблони Vue.js (SFC — Single File Components). Вони відповідають виключно за структуру та стилізацію (HTML/CSS). Завдяки реактивному зв'язку, будь-яка зміна показника AQI (Air Quality Index) у моделі миттєво відображається у представленні без необхідності перезавантаження сторінки.
3. ViewModel (Модель представлення): реалізована за допомогою Vue Composition API та інструментарію Pinia. Це «інтелектуальний шар», який обробляє «сирі» дані: виконує фільтрацію, нормалізацію значень та підготовку контексту для штучного інтелекту.

Використання патерну MVVM у поєднанні з Composition API забезпечує наступні переваги для проєкту:

- Логічна ізоляція: код для роботи з AI-моделлю та код для відображення погодних віджетів знаходяться в різних модулях, що спрощує тестування.
- Двостороннє зв'язування даних (Data Binding): дозволяє автоматично

синхронізувати стан інтерфейсу з актуальними метеоданими.

- Перевикористання логіки: можливість створення *composables* — окремих функцій, які можна використовувати в різних частинах застосунку (наприклад, функція розрахунку кольору індикатора AQI).

Таким чином, архітектура MVVM стає технологічним фундаментом, що дозволяє системі «EcoWatch» стабільно працювати як вискоєфективний Single Page Application (SPA), забезпечуючи миттєву реакцію на зміну зовнішніх екологічних факторів [69, 71].

### 2.1.2 Проєктування схеми взаємодії з хмарними API

В архітектурі веб-застосунку «EcoWatch» реалізовано модель прямої взаємодії клієнтської частини з хмарними сервісами (Serverless-like approach). Оскільки проєкт не передбачає наявності власного проміжного сервера, основне завдання з агрегації та синхронізації даних покладається на фронтенд-архітектуру [69, 77, 87].

Схема взаємодії спроектована таким чином, щоб забезпечити цілісність даних при отриманні інформації з двох незалежних джерел. Взаємодія поділяється на три ключові етапи:

1. Геокодування та ініціалізація: — Отримання координат користувача через Browser Geolocation API. — Запит до OpenWeather Geocoding API для перетворення координат у назву населеного пункту (або навпаки).
2. Агрегація метеорологічного та екологічного контексту: — Виконання паралельних запитів до Current Weather API та Air Pollution API. — Акумуляція отриманих значень (температура, вологість, індекс AQI, концентрація PM2.5) у єдиний об'єкт стану в Pinia store [74, 79].
3. Інтелектуальна генерація висновків (AI Orchestration): — Формування структурованого JSON-об'єкта, який містить усі зібрані показники. — Передача цього об'єкта як контексту до Gemini API (Google AI Studio). — Отримання та десеріалізація відповіді від ШІ для відображення у користувацькому інтерфейсі [75, 90].

Технічні особливості реалізації схеми: — Асинхронність: Всі операції виконуються за допомогою `async/await` та бібліотеки `Axios`, що запобігає блокуванню головного потоку (UI thread) під час очікування відповідей від серверів. — Безпека запитів: Використання механізмів CORS (Cross-Origin Resource Sharing) та авторизація через заголовки `x-api-key` для доступу до ресурсів Google та OpenWeather. — Кешування: Для мінімізації кількості запитів реалізовано короткострокове кешування результатів у пам'яті застосунку, що дозволяє уникнути повторних викликів API при незначних змінах стану інтерфейсу [73, 89].

## **2.2 Проєктування логіки обробки екологічних даних**

### **2.2.1 Модульна організація логіки доступу до даних**

Організація програмного коду веб-застосунку «EcoWatch» базується на принципах модульності та чіткого розподілу відповідальності (Separation of Concerns). Для забезпечення чистоти архітектури у проєкті реалізовано багаторівневу структуру, де логіка взаємодії з зовнішніми ресурсами відокремлена від компонентів інтерфейсу.

Згідно з розробленою структурою (рис. 2.1), ключовими елементами логічної організації є:

Модуль взаємодії з API (`src/api/`): — `weatherApi.ts`: інкапсулює низькорівневу логіку HTTP-запитів до OpenWeather. Використовує асинхронні функції для отримання даних про поточну погоду та якість повітря. — `geminiApi.ts`: відповідає за конфігурацію зв'язку з Google AI Studio, передачу системних промптів та отримання генеративних відповідей від моделі Gemini 1.5 Flash.

Шар функціональної реактивності (`src/composables/`): — Використання патерну Vue Composables (`useWeather.ts`, `useAirQuality.ts`) дозволяє винести бізнес-логіку та управління станом за межі UI-компонентів. Це забезпечує можливість перевикористання логіки отримання даних у різних частинах застосунку.

Допоміжні утиліти та мапери (`src/utils/`): — `weatherMapper.ts`: виконує критично важливу функцію перетворення (mapping) складних відповідей від API у

спрощені об'єкти, з якими зручно працювати фронтенд-частині. — `getDailyTemps.ts`: містить алгоритми обробки масивів даних для формування коректної статистики.

Типізація та моделі даних (`src/types/`): — Файл `weather.ts` містить опис інтерфейсів TypeScript. Це гарантує сувору типізацію по всьому проєкту, що мінімізує помилки при зверненні до властивостей об'єктів метеоданих.

Переваги обраної структури: — Масштабованість: за необхідності додавання нового функціоналу (наприклад, прогнозу на тиждень), зміни вносяться лише у відповідний API-модуль та мапер, не зачіпаючи логіку відображення. — Тестованість: кожен файл у папках `api` та `utils` є незалежним модулем, що дозволяє проводити ізольоване модульне тестування функцій обробки даних. — Надійність: завдяки винесенню типів у окремий простір, розробник отримує повну підтримку IntelliSense, що пришвидшує розробку та зменшує кількість багів у Runtime.

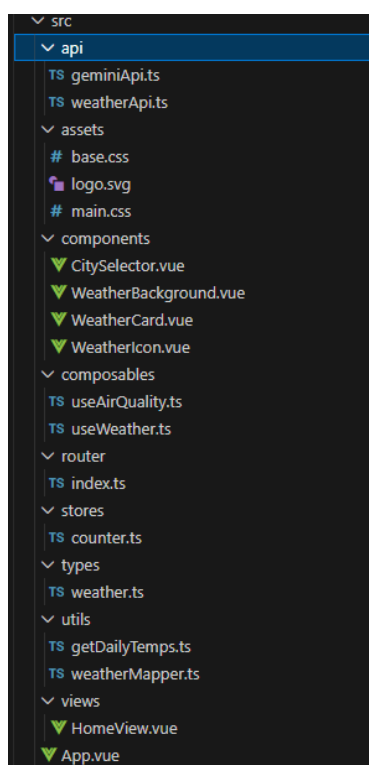


Рисунок 2.1 – Ієрархічна структура каталогів проєкту EcoWatch

### 2.2.2 Моделювання потоків даних та форматів обміну

Функціонування застосунку «EcoWatch» базується на безперервному обміні даними між клієнтською частиною та хмарними сервісами. Основним форматом

обміну даними обрано JSON (JavaScript Object Notation), як найбільш детермінований та легковаговий стандарт для сучасних веб-інтерфейсів [73, 79].

Процес обробки потоку даних (Data Flow) розділений на чотири послідовні етапи:

1. Ініціація та запит (Requesting):

- Клієнт надсилає асинхронний GET-запит із заголовком автентифікації.
- Параметри запиту включають координати (lat, lon) та мовний код (lang=uk).

2. Десеріалізація та валідація:

- Отриманий JSON-пакет від OpenWeather або Gemini десеріалізується у внутрішню структуру JavaScript.
- Завдяки TypeScript, на цьому етапі виконується перевірка відповідності типів, що гарантує наявність усіх необхідних полів (температури, коду погодних умов, рівня AQI).

3. Трансформація через мапери (Mapping):

- Оскільки відповіді від сторонніх API часто містять надлишкову інформацію, використовується модуль weatherMapper.ts.
- Він виконує функцію фільтрації: вилучає лише необхідні показники та приводить їх до зручного для фронтенду вигляду (наприклад, перетворення Unix-timestamp у зрозумілий формат дати).

4. Синхронізація зі станом (State Update):

- Очищені та структуровані дані записуються у реактивний об'єкт стану. Це автоматично ініціює перерендер компонентів інтерфейсу.

Формати даних, що використовуються в проєкті:

- Метеодані: Об'єкт, що містить поточні показники (main), стан хмарності (weather[0].description) та системні параметри (видимість, швидкість вітру).
- Екологічні дані (Pollution Data): Масив показників концентрації газів (NO2, O3, SO2) та дрібнодисперсного пилу (PM2.5, PM10).
- AI-контекст: Спеціальний формат промпту, де цифрові дані інтегруються в текстовий шаблон для відправки в Gemini API.

Використання строгого моделювання потоків даних дозволяє системі уникати помилок типу «undefined» при відображенні інформації та забезпечує стабільну роботу застосунку навіть при нестабільній швидкості з'єднання [68, 80].

### 2.2.3 Алгоритм формування інтелектуальних промтів для LLM

Ключовою інноваційною складовою веб-застосунку «EcoWatch» є механізм семантичної інтерпретації метеорологічних та екологічних даних за допомогою великої мовної моделі Gemini 1.5 Flash. На відміну від традиційних систем, що використовують жорстко заcoded шаблони, розроблений алгоритм базується на принципах динамічного інжинірингу промтів (Prompt Engineering), що дозволяє генерувати унікальні рекомендації для кожної комбінації погодних факторів [75, 77].

Процес формування інтелектуального запиту складається з чотирьох архітектурних етапів:

1. Визначення системного контексту (System Instruction): На рівні ініціалізації API-запиту в модулі `geminiApi.ts` моделі задається статична роль «Екологічного асистента-аналітика». Системна інструкція містить директиви щодо стилістики відповідей (лаконічність, фокус на здоров'ї) та обмеження (використання лише української мови, посилання на норми БООЗ).
2. Динамічна ін'єкція параметрів довкілля (Context Injection): Алгоритм збирає дані з реактивного сховища та трансформує їх у структурований текстовий блок. До складу контексту входять: — Метеопараметри: температура, відносна вологість (фактор, що впливає на сприйняття температури та розсіювання домішок), швидкість вітру. — Екологічні метрики: індекс AQI та концентрація специфічних забруднювачів (PM2.5, PM10, NO2, CO, SO2, O3). — Часовий контекст: поточний час для адаптації порад (наприклад, рекомендації щодо ранкового провітрювання приміщень).
3. Логічна структура промту (Prompt Template): Для забезпечення стабільності виводу використовується шаблон, що працює за принципом «Chain-of-Thought» (ланцюжок думок). Модель спочатку має проаналізувати загальний

стан повітря, а потім запропонувати дії. Приклад структури промπτу: «Дані: Темп: {t}, AQI: {a}, PM2.5: {p}. Завдання: 1. Оціни рівень небезпеки. 2. Сформулюй 3 поради для активного відпочинку. Мова: українська».

4. Обробка та десеріалізація результату: Оскільки LLM повертає неструктурований текст, у проєкті реалізовано механізм «парсингу» відповіді. Система розділяє аналітичний висновок та список порад для їх коректної візуалізації в окремих UI-блоках.

Технічне обґрунтування вибору Gemini 1.5 Flash: Вибір даної моделі обумовлений її високою швидкістю обробки токенів та низькою затримкою (Latency), що є критичним для забезпечення безперервного користувацького досвіду в Single Page Application. Крім того, модель демонструє високу точність у роботі з українською мовою та складними числовими даними, що мінімізує ризик виникнення «галюцинацій» ШІ при інтерпретації показників забруднення [82, 90].

Розроблений алгоритм дозволяє подолати «семантичний розрив» між сухими цифрами метеостанцій та реальними потребами користувача, перетворюючи «EcoWatch» з простого інформаційного дашборду на інтелектуальну систему підтримки прийняття рішень у сфері екологічної безпеки.

#### 2.2.4 WebSocket real-time комунікація

Socket.io інтегровано для двостороннього real-time зв'язку між сервером та клієнтами. З'єднання встановлюється при успішній автентифікації користувача, JWT токен передається через query параметр або auth header при handshake.

При підключенні виконується Socket.io middleware для валідації токена: декодується JWT, витягується userId, створюється record в connections колекції з socketId. Socket приєднується до room з назвою user:\${userId} для персональних повідомлень та до room folder:\${folderId} для кожної папки, до якої користувач має доступ.

Реалізовано обробку наступних events:

Client → Server events: `join_folder` – підключення до room папки при навігації, `leave_folder` – відключення при зміні директорії, `ping` – heartbeat для підтримки з'єднання, `disconnect` – очищення connections та видалення з rooms.

Server → Client events: `file_uploaded` – broadcasts в `folder:${folderId}` room при завантаженні нового файлу `payload {file: FileMetadata, folderId}`, `file_deleted` – при видаленні файлу `payload {fileId, folderId}`, `file_restored` – при відновленні з trash, `folder_created` – при створенні підпапки, `folder_shared` – emit в `user:${recipientId}` room при отриманні доступу до папки `payload {folder, permissions, sharedBy}`, `folder_unshared` – при відкликанні доступу, `user_connected` / `user_disconnected` – broadcast всім онлайн користувачам для оновлення статусів `payload {userId, timestamp}`, `storage_updated` – при зміні доступного місця на диску.

Для оптимізації використовується rooms механізм Socket.io: кожна папка має окремий room, події broadcasts тільки користувачам, які мають доступ до цієї папки. При отриманні події на клієнті автоматично оновлюється відповідний стан в Zustand store без необхідності ручного refresh.

Реалізовано reconnection logic на клієнті: при розриві з'єднання (мережева помилка, server restart) автоматично виконується повторне підключення з exponential backoff (затримки 1s, 2s, 4s, 8s до максимум 30s), після успішного reconnect відправляється sync\_state запит для отримання пропущених змін.

Error handling включає обробку помилок автентифікації (`disconnect` з повідомленням), full message queue при недоступності клієнта (зберігання останніх 50 повідомлень для доставки при reconnect), timeout detection (`disconnect` якщо немає pong відповіді протягом 60 секунд).

## **2.3 Розробка клієнтської частини (Frontend)**

### **2.3.1 Модульна структура компонентів**

Проектування інтерфейсу веб-застосунку «EcoWatch» базується на сучасній компонентній парадигмі, що реалізована засобами фреймворку Vue 3. Основним структурним елементом системи є Single File Component (SFC) — технологія, яка дозволяє об'єднати логіку (TypeScript), розмітку (HTML) та стилізацію (CSS) в

межах одного файлу `.vue`. Такий підхід забезпечує високу інкапсуляцію коду та спрощує процес налагодження інтерфейсу [66, 69, 86].

Ієрархічна декомпозиція інтерфейсу:

Згідно з розробленою архітектурою, інтерфейс було декомпозовано на незалежні функціональні вузли, що дозволило уникнути створення «громіздких» файлів та забезпечити принцип єдиної відповідальності (Single Responsibility Principle). Структура компонентів у директорії `src/components/` організована наступним чином:

- Компоненти високого рівня (Layout Components): — `TheHeader.vue`: інкапсулює логіку заголовка, включаючи логотип та інтегрований рядок пошуку. — `TheFooter.vue`: відповідає за відображення метаданих проєкту та посилань на API-провайдерів.
- Інформаційні віджети (Data Displays): — `WeatherCard.vue`: складний компонент, що отримує дані через механізм `Props`. Він відповідає за візуалізацію ключових метеопараметрів. Особливістю є використання обчислюваних властивостей (`computed`), які автоматично форматують числові значення (наприклад, додавання знака градуса або одиниць виміру швидкості вітру). — `AirQualityCard.vue`: спеціалізований модуль для екологічного моніторингу. В ньому реалізована логіка кольорового кодування: компонент аналізує отриманий індекс AQI та динамічно змінює класи стилів для візуальної сигналізації про рівень небезпеки повітря.
- Інтелектуальний модуль зворотного зв'язку: — `GeminiInsights.vue`: найбільш динамічний вузол системи. Він керує станами очікування (`Loading states`) під час генерації відповіді мовною моделлю та забезпечує безпечний рендеринг отриманого JSON-об'єкта. У разі виникнення помилок при десеріалізації відповіді від ШІ, цей компонент перемикається на резервний режим відображення даних.

Технічні аспекти реалізації на базі Composition API:

У процесі розробки компонентів було використано синтаксичний цукор

`<script setup>`, що є стандартом для високопродуктивних додатків у 2026 році. Це дозволило досягти наступних переваг:

1. **Покращена типізація (TypeScript Integration):** Кожен вхідний параметр (Prop) суворо типізований за допомогою інтерфейсів, визначених у `src/types/weather.ts`. Це виключає передачу некоректних структур даних між компонентами на етапі збірки проєкту [68, 78].
2. **Директиви управління рендерингом:** Для оптимізації продуктивності широко використовуються директиви `v-if` та `v-show`. Наприклад, детальна екологічна статистика рендериться лише після успішного підтвердження координат користувача, що зменшує навантаження на DOM-дерево.
3. **Scoped Styles (Ізоляція стилів):** Кожен компонент містить блок стилів із атрибутом `scoped`. Це гарантує, що CSS-правила одного модуля (наприклад, тіні карток погоди) не будуть конфліктувати зі стилями інших елементів, забезпечуючи передбачуваний зовнішній вигляд застосунку на будь-якій сторінці.
4. **Використання слотів (Slots):** Для створення гнучких інтерфейсних рішень застосовано систему слотів, що дозволяє передавати складний HTML-контент усередину модальних вікон або карток, зберігаючи при цьому загальну стилістику контейнера.

Завдяки такій модульній структурі, застосунок «EcoWatch» демонструє високу швидкість відгуку та легкість у масштабуванні. Додавання нового функціоналу (наприклад, графіка зміни температури за тиждень) вимагає лише створення нового компонента та його реєстрації в батьківському вузлі, не зачіпаючи логіку роботи існуючих модулів [71, 84].

### 2.3.2 Управління станом застосунку

У процесі розробки складних інтерактивних інтерфейсів, таких як «EcoWatch», виникає проблема ефективної синхронізації даних між великою кількістю незалежних компонентів. Для розв'язання задачі збереження та розподілу метеорологічних показників було обрано архітектуру централізованого управління

станом на базі бібліотеки **Pinia** [71, 86].

Архітектурне обґрунтування вибору Pinia:

Pinia є офіційно рекомендованим інструментом для Vue 3, який прийшов на зміну Vuex. Її використання у проєкті дозволяє уникнути проблеми «Prop Drilling» (передачі даних через декілька рівнів компонентів, які їх не використовують безпосередньо) та забезпечує єдине джерело істини (Single Source of Truth) для всього застосунку. Ключовими елементами структури сховища є:

- State (Стан): Реактивний об'єкт, що містить поточні дані про погоду, масив екологічних показників, назву обраного міста та прапорці стану завантаження (isLoading). Використання TypeScript дозволяє суворо типізувати стан, гарантуючи, що компоненти отримають дані саме у форматі інтерфейсів IWeather або IAirQuality.
- Actions (Дії): Асинхронні методи, що відповідають за бізнес-логіку. Саме в Actions викликаються функції з src/api/ (наприклад, getWeather). Після отримання відповіді від API, Action оновлює State, що автоматично ініціює перерендер усіх підписаних на ці дані компонентів.
- Getters (Геттери): Обчислювані властивості сховища. Наприклад, геттер може автоматично фільтрувати прогноз погоди або розраховувати середню температуру за день на основі наявного масиву даних.

Функціональні переваги реалізації:

1. Синхронізація інтелектуальних порад: Оскільки відповідь від Gemini API зберігається в глобальному стані, вона залишається доступною навіть при переході між різними екранами або вкладками застосунку без повторного звернення до ШІ.
2. Глобальна реактивність: Зміна міста в компоненті пошуку через Action у Pinia миттєво оновлює дані в картках погоди, екологічних віджетах та блоці ШІ-аналітики одночасно.
3. Персистентність (Persistence): Реалізовано механізм збереження ключових параметрів стану (наприклад, останнього обраного міста) у LocalStorage. Це дозволяє відновлювати стан застосунку після перезавантаження сторінки,

значно покращуючи користувацький досвід (UX).

4. Інструменти розробника (DevTools): Pinia забезпечує глибоку інтеграцію з Vue DevTools, що дозволяє відстежувати кожну зміну стану, робити «знімки» даних та аналізувати часову шкалу (timeline) запитів, що критично важливо для налагодження асинхронних операцій [69, 73].

Таким чином, використання Pinia перетворює розрізнені компоненти на цілісну систему, де дані надійно захищені від неузгодженості, а логіка їх обробки винесена в окремий, легко керований шар застосунку.

### 2.3.3 Реалізація реактивної візуалізації та адаптивності

Особливістю інтерфейсу «EcoWatch» є використання концепції контекстно-залежної візуалізації. Це означає, що зовнішній вигляд застосунку не є статичним, а реактивно адаптується до метеорологічних даних, отриманих від API у реальному часі [70, 78, 84].

Механізм динамічного фону (Dynamic Backgrounds):

Для покращення користувацького досвіду (UX) реалізовано систему динамічної зміни візуальних станів. Логіка базується на використанні погодних кодів (Weather Codes) від OpenWeatherMap:

- Алгоритм мапінгу: У модулі weatherMapper.ts кожному коду погоди (наприклад, 800 — ясне небо, 500 — дощ) відповідає певний графічний асет або градієнтна схема.
- Реактивне перемикання: Завдяки системі реактивності Vue 3, як тільки стан у Pinia оновлюється новими даними, компонент фону автоматично змінює CSS-клас або URL фонового зображення без перезавантаження сторінки.
- Візуальна типізація: Кожному рівню індексу AQI відповідає свій колірний ідентифікатор, що дозволяє користувачу на підсвідомому рівні оцінити екологічну безпеку ще до читання числових показників.

Адаптивність та Mobile-first підхід:

Враховуючи, що екологічний моніторинг часто здійснюється з мобільних пристроїв, розробка велася за принципом Mobile-first. Це забезпечує стабільну

роботу на пристроях з обмеженою шириною екрана (від 320px) [78].

Технічна реалізація адаптивності включає:

- Flexbox та CSS Grid: Використання сучасних методів верстки дозволяє елементам дашборду гнучко змінювати свій розмір та розташування. Наприклад, картки показників на десктопі розташовані в ряд, а на смартфонах автоматично вибудовуються у вертикальну колонку.
- Відносні одиниці виміру: Замість фіксованих пікселів використано одиниці rem, em та відсотки, що гарантує збереження пропорцій інтерфейсу при масштабуванні.
- Оптимізація активних зон: Всі інтерактивні елементи (кнопки пошуку, вкладки ШІ-порад) мають розмір не менше 44x44px, що відповідає стандартам доступності (Accessibility) для сенсорних екранів.

Реактивна фільтрація даних:

Реалізація також включає «розумне» приховування другорядних даних на малих екранах за допомогою директив v-if. Це дозволяє уникнути перевантаження інтерфейсу (information overload) та зосередити увагу користувача на найважливіших показниках — температурі та стані повітря.

## **2.4 Обґрунтування вибору технологічного стеку**

### **2.4.1. Переваги Vue 3 та Vite у сучасній веб-розробці.**

Вибір технологічного фундаменту для проєкту «EcoWatch» базувався на аналізі сучасних вимог до архітектури клієнтських веб-застосунків. Основним критерієм була здатність фреймворку ефективно обробляти динамічні потоки екологічних даних без втрати продуктивності інтерфейсу. У результаті дослідження було обрано екосистему Vue 3 у поєднанні з надшвидкою системою збірки Vite [66, 69, 73].

Поглиблене обґрунтування вибору Vue 3:

Перехід від другої до третьої версії фреймворку став вирішальним для реалізації інтелектуальних функцій «EcoWatch». Основні переваги Vue 3 у контексті даного проєкту включають:

- Парадигма Composition API: На відміну від застарілого Options API, новий підхід дозволяє групувати логіку за функціональними ознаками. Це стало критично важливим для реалізації модулів `api/` та `composables/`. Наприклад, вся логіка взаємодії з метеорологічними сервісами (стан завантаження, обробка помилок, запити) тепер інкапсульована в одному місці, що значно полегшує її підтримку та тестування.
- Оптимізований механізм реактивності: Використання JavaScript Proxy замість старого методу `Object.defineProperty` дозволило зняти обмеження на відстеження змін у вкладених об'єктах. Оскільки відповіді від Gemini API мають складну ієрархічну структуру JSON, нова система реактивності Vue 3 забезпечує миттєве оновлення інтерфейсу при мінімальних витратах обчислювальних ресурсів.
- Телепортація та фрагменти (Teleport & Fragments): Ці можливості дозволили гнучко інтегрувати модальні вікна з III-порадами та екологічними графіками в будь-яку частину DOM-дерева, зберігаючи при цьому логічну цілісність компонентної структури.

Інженерна роль Vite у життєвому циклі розробки:

Використання Vite замість класичних інструментів (таких як Webpack) дозволило розв'язати проблему повільної ітерації розробки. Технічні переваги Vite для проєкту полягають у наступному:

1. Нативна підтримка ESM (ES Modules): У режимі розробки Vite не збирає весь застосунок у гігантський бандл. Замість цього він подає кожен файл як окремий модуль безпосередньо в браузер. Це забезпечує стабільно швидкий запуск проєкту незалежно від кількості доданих екологічних віджетів чи об'ємних TypeScript-інтерфейсів.
2. Інтелектуальне попереднє групування залежностей: Vite використовує інструмент esbuild, написаний на мові Go, для попередньої обробки важких бібліотек (наприклад, Axios). Це відбувається в 10–100 разів швидше, ніж у JavaScript-орієнтованих системах збірки.
3. Оптимізація Production-збірки: Для створення фінальної версії застосунку

Vite застосовує передові методи, такі як Tree-shaking (видалення невикористаного коду) та Scope Hoisting. Це дозволило максимально зменшити розмір файлів, що передаються користувачеві, забезпечуючи швидкий запуск «EcoWatch» навіть на мобільних пристроях з низькою швидкістю інтернету.

Таким чином, використання зв'язки Vue 3 + Vite створило умови для розробки продукту з високим показником «Time-to-Interactive», що є критично важливим для сучасних сервісів екологічного моніторингу, де актуальність інформації вимірюється секундами [71, 80, 89].

#### **2.4.2 Роль TypeScript у забезпеченні надійності системи**

Сучасні веб-застосунки, що оперують великими масивами даних із зовнішніх джерел, потребують суворого контролю типів. Для розробки «EcoWatch» було обрано мову TypeScript, яка є надбудовою над JavaScript і впроваджує статичну типізацію. Це рішення дозволило мінімізувати кількість логічних помилок на етапі написання коду та забезпечити високу надійність архітектури [68, 78, 86].

Ключові аспекти використання TypeScript у проєкті:

1. Типізація відповідей від сторонніх API: Дані, що надходять від OpenWeather та Gemini AI, мають складну ієрархічну структуру. У проєкті створено окремий модуль `src/types/`, де за допомогою інтерфейсів (interface) та типів (type) чітко визначено структуру кожного об'єкта. Наприклад, інтерфейс `IWeatherData` суворо обмежує поля для температури, вологості та описів погодних умов. Це гарантує, що жоден компонент інтерфейсу не намагатиметься звернутися до властивості, якої не існує в об'єкті, що запобігає виникненню критичної помилки `undefined is not a function`.
2. Забезпечення цілісності контракту даних у маперах: Завдяки TypeScript, модуль `weatherMapper.ts` функціонує як надійний фільтр. Якщо структура вхідних даних від API зміниться, компілятор миттєво повідомить про невідповідність типів ще до того, як код буде розгорнуто. Це дозволяє розробнику швидко адаптувати логіку трансформації даних, зберігаючи

стабільність усього ланцюжка обробки.

3. Покращення інтелектуального аналізу коду (IntelliSense): Використання типізації значно пришвидшило розробку завдяки автодоповненню в IDE. Під час роботи з компонентами Vue розробник бачить усі доступні властивості пропсів (props) та стан сховища Pinia, що виключає друкарські помилки в назвах полів та параметрів.
4. Безпека інтеграції з ШІ-моделями: Оскільки Gemini API очікує дані у певному форматі, використання TypeScript дозволило типізувати об'єкти промптів. Це гарантує, що всі необхідні екологічні метрики будуть коректно передані до моделі, а отримана JSON-відповідь буде правильно десеріалізована згідно з визначеною схемою.

Технічні переваги для супроводу проєкту: — Самодокументованість: Код стає зрозумілим без додаткових коментарів, оскільки типи даних прямо вказують на призначення кожної змінної чи функції. — Рефакторинг: Будь-які зміни в архітектурі (наприклад, перейменування поля в базі даних погоди) автоматично підсвічуються у всіх частинах проєкту, де це поле використовується.

Таким чином, інтеграція TypeScript стала фундаментом для створення відмовостійкої системи. Це дозволило перенести виявлення помилок із етапу виконання (Runtime) на етап компіляції, що критично важливо для сервісів моніторингу, де точність та доступність даних мають першочергове значення [74, 80, 89].

### 2.4.3 Вибір моделі Gemini 1.5 Flash для задач метеомоніторингу

Центральним елементом аналітичної системи «EcoWatch» є велика мовна модель (LLM), яка виконує роль інтелектуального інтерпретатора метеорологічних та екологічних даних. Після порівняльного аналізу сучасних рішень (GPT-4o, Claude 3.5 Sonnet, Llama 3) для реалізації задач метеомоніторингу було обрано модель Gemini 1.5 Flash від Google DeepMind [75, 77, 90].

Обґрунтування вибору моделі для задач проєкту:

1. Мінімальна затримка (Ultra-low Latency): У задачах метеомоніторингу

швидкість отримання інформації є критичною. Модель версії Flash спроектована спеціально для високої швидкості генерації (inference). Це дозволяє користувачу отримувати персоналізовані поради щодо екологічного стану миттєво після завантаження метеоданих, що забезпечує безперервний користувацький досвід (UX).

2. Висока щільність контексту та робота зі структурами даних: Завдяки архітектурі Transformer-MM, модель Gemini 1.5 Flash демонструє високу точність при роботі з JSON-об'єктами. Для «EcoWatch» це дозволяє передавати складні масиви даних про концентрацію забруднювачів (PM2.5, NO2, SO2) та отримувати логічно обґрунтовані висновки без ризику «галюцинацій» ІІІ.
3. Нативна інтеграція з екосистемою розробки: Використання Google AI Studio та підтримка режиму responseMimeType: 'application/json' дозволяють безшовно інтегрувати модель у фронтенд-частину на базі TypeScript. Це виключає необхідність складного парсингу тексту та гарантує, що дані для історичної довідки або порад щодо одягу будуть отримані у строго визначеному форматі.
4. Економічна ефективність для наукових досліджень: Наявність безкоштовного рівня доступу з високими лімітами запитів робить цю модель оптимальним вибором для магістерського проєкту, дозволяючи проводити глибоке тестування алгоритмів промпт-інжинірингу без додаткових витрат на інфраструктуру.

Семантична відповідність: Модель Gemini 1.5 Flash показала найкращі результати в адаптації складної метеорологічної термінології українською мовою. Вона здатна не лише констатувати факти (наприклад, «високий рівень AQI»), а й будувати причинно-наслідкові зв'язки, пояснюючи вплив конкретних показників на здоров'я людини в контексті поточної температури та вологості [82, 90].

## 3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

### 3.1 Середовище розробки та конфігурація проєкту

#### 3.1.1 Загальна структура проєкту та дерево компонентів

Програмна реалізація системи «EcoWatch» базується на модульній архітектурі, що забезпечується фреймворком Vue 3. Структура проєкту організована за принципом чіткого розподілу відповідальності (Separation of Concerns), де логіка обробки даних, управління станом та візуалізація інтерфейсу рознесені по окремих ієрархічних рівнях.

Основними структурними одиницями архітектури є:

- Рівень представлення (UI Layer): Складається з ієрархії Vue-компонентів (src/components/). Базовий компонент App.vue координує відображення віджетів погоди, екологічних карток та інтелектуального блоку порад.
- Рівень реактивної логіки (Composables): Модулі в директорії src/composables/ (наприклад, useWeather.ts) виступають «мозковим центром» компонентів. Вони інкапсулюють у собі реактивні змінні (ref, reactive) та методи ініціалізації запитів.
- Сервісний шар API (Data Access Layer): Файли weatherApi.ts та geminiApi.ts у папці src/api/ відповідають за низькорівневу комунікацію з хмарними сервісами через протокол HTTP.
- Шар трансформації даних (Utils/Mappers): Модуль weatherMapper.ts виконує критичну функцію нормалізації даних. Він перетворює складні JSON-відповіді від сторонніх API у спрощені об'єкти, що відповідають внутрішнім TypeScript-інтерфейсам системи.

Взаємодія між рівнями побудована на подієво-орієнтованій моделі: коли користувач змінює місто в інтерфейсі, компонент викликає метод у Composable-модулі, який ініціює ланцюжок запитів до API. Після отримання та мапінгу даних, стан автоматично оновлюється, що завдяки системі реактивності Vue 3 спричиняє миттєвий перерендер відповідних частин інтерфейсу.

Структура компонентів застосунку зображена на рисунку 3.1.

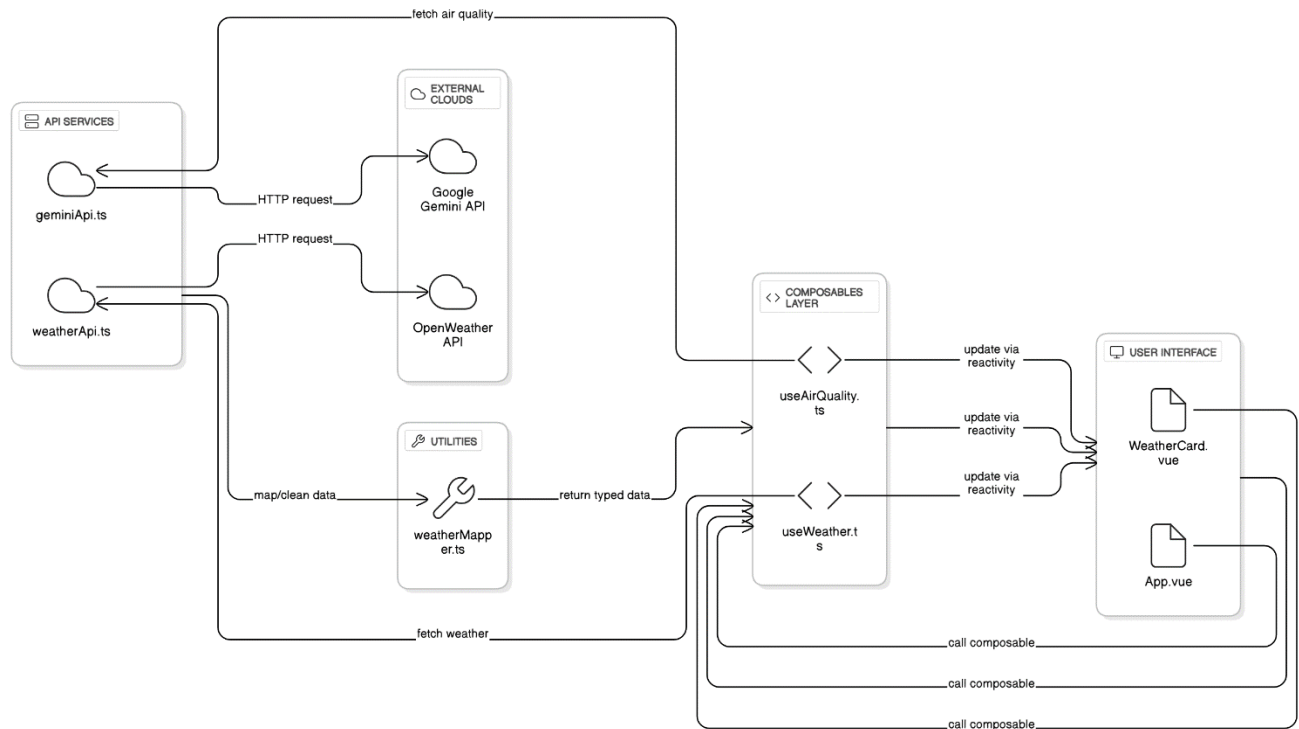


Рисунок 3.1 – Структура компонентів застосунку

### 3.1.2 Модель даних та типізація

Однією з ключових переваг розробленої системи є використання суворої типізації даних на основі мови TypeScript. Це дозволяє визначити чіткі «контракти» взаємодії між модулями, що мінімізує ризик виникнення помилок при обробці складних JSON-структур, які надходять від зовнішніх API-сервісів [68, 80].

У проєкті «EcoWatch» модель даних реалізована через систему інтерфейсів, зосереджених у директорії `src/types/`. Основна модель базується на ієрархічній структурі, що описує метеорологічні та екологічні показники.

Ключові інтерфейси моделі даних:

**IWeatherData:** Описує структуру основних погодних параметрів (температура, вологість, швидкість вітру, атмосферний тиск).

**IAirPollution:** Містить опис концентрації забруднюючих речовин (PM2.5, PM10, NO2) та загальний індекс якості повітря (AQI).

Нижче наведено фрагмент програмного коду, що описує базову модель метеорологічних даних:

```
export interface IWeatherData {
```

```

    city: string;
    temp: number;
    feels_like: number;
    humidity: number;
    pressure: number;
    description: string;
    icon: string;
    wind_speed: number;
    aqi?: number;
  }
export interface IAirPollution {
  pm2_5: number;
  pm10: number;
  no2: number;
  co: number;
  o3: number;
}

```

#### Переваги обраної моделі типізації:

- Безпека під час компіляції: Використання інтерфейсів дозволяє виявляти невідповідності даних ще на етапі розробки. Наприклад, спроба передати рядок замість числа в поле температури буде миттєво заблокована компілятором TypeScript.
- Автодокументація: Опис типів слугує живою документацією проєкту. Будь-який розробник, відкривши файл інтерфейсів, може миттєво зрозуміти структуру даних, з якими працює застосунок.
- Інтеграція з маперами: Типізація забезпечує коректну роботу модуля `weatherMapper.ts`. Це гарантує, що після трансформації сирих даних з API (де назви полів можуть бути незручними, наприклад, `main.temp_max`), система отримає об'єкт зі зрозумілою та передбачуваною структурою.

Завдяки впровадженню такої моделі даних, вдалося досягти високої стабільності роботи інтелектуального модуля Gemini, оскільки він завжди отримує суворо структуровані вхідні параметри для аналізу [86, 89].

### 3.1.3 API-оркестрація та маршрутизація

Ефективне функціонування системи «EcoWatch» залежить від злагодженої взаємодії між внутрішньою логікою застосунку та зовнішніми хмарними сервісами. Цей процес, відомий як API-оркестрація, реалізований через ізольовані сервісні модулі, що працюють у синергії з системою маршрутизації Vue Router [69, 73].

Механізм API-оркестрації:

У проєкті застосовано модульний підхід до організації мережових запитів. Замість безпосереднього звернення до серверів з UI-компонентів, розроблено шар абстракції в директорії `src/api/`:

- Конфігурація запитів: Кожен сервіс (`weatherApi.ts`, `geminiApi.ts`) містить базові налаштування URL-адрес та механізми впровадження API-ключів через змінні середовища Vite.
- Асинхронна обробка: Використання конструкцій `async/await` дозволяє системі виконувати запити до OpenWeather та Google AI Studio паралельно або послідовно, не блокуючи основний потік інтерфейсу.
- Трансформація відповідей: Оркестрація включає етап попередньої обробки даних, де сирі відповіді очищуються від надлишкової метаданих перед тим, як потрапити до глобального стану Pinia.

Система маршрутизації (Vue Router):

Маршрутизація в застосунку виконує роль координатора станів. Вона не лише відповідає за перемикання між екранами, а й забезпечує зв'язок між станом інтерфейсу та URL-адресою браузера:

1. Декларативні маршрути: Визначено базовий шлях для головної сторінки (`/`), де розташований дашборд метеомоніторингу.
2. Динамічне керування станом: Маршрутизатор дозволяє зберігати контекст обраного міста в адресному рядку. Це дає можливість користувачеві використовувати кнопки «Назад/Вперед» браузера для перевірки попередніх запитів та ділитися посиланнями на конкретні результати аналізу.
3. Синхронізація з Lifecycle Hooks: При ініціалізації маршруту спрацьовують

хуки життєвого циклу Vue-компонентів (наприклад, `onMounted`), які автоматично запускають ланцюжок API-запитів для актуалізації даних на сторінці.

Обробка переходів: Для покращення UX (User Experience) під час оркестрації запитів впроваджено візуальну індикацію переходів. Поки Router очікує завершення асинхронних операцій у сервісному шарі, інтерфейс відображає стани завантаження, що мінімізує відчуття затримки у користувача.

## 3.2 Реалізація основного функціоналу

### 3.2.1 Модуль отримання та мапінгу метеорологічних даних

Функціонування системи «EcoWatch» починається з отримання актуальних метеорологічних показників. Для реалізації цього завдання було розроблено сервісний модуль `weatherApi.ts` та утиліту трансформації даних `weatherMapper.ts`. Це дозволило відокремити процес мережевої взаємодії від бізнес-логіки застосунку [73, 77, 80].

Реалізація мережевого запиту:

Запити до OpenWeatherMap виконуються за допомогою методу `fetch` з використанням асинхронних функцій. Основна складність полягає в тому, що для повної картини (погода + екологія) необхідно здійснити два послідовних або паралельних запити: перший — для отримання координат міста та поточної температури, другий — для отримання індексу якості повітря (AQI) за знайденими координатами.

```
// Спрощена логіка запиту в weatherApi.ts
export async function fetchWeatherData(city: string) {
  const url = `
  ${BASE_URL}/weather?q=${city}&units=metric&appid=${API_KEY}`;
  const response = await fetch(url);
  if (!response.ok) throw new Error('City not found');
  return await response.json();
}
```

На рисунку 3.2 продемонстровано структуру сформованого запиту (Payload),

що надсилається до сервера OpenWeatherMap для отримання даних за назвою міста.

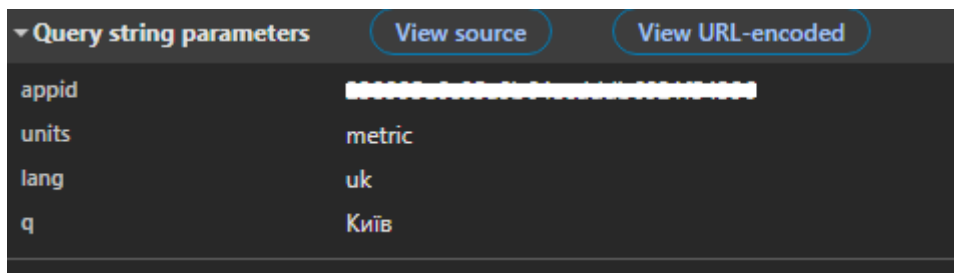


Рисунок 3.2 – Параметри HTTP-запиту для метеорологічного API

Алгоритм мапінгу та нормалізації даних:

Відповіді від метеорологічних сервісів часто містять надмірну кількість інформації (метадані, внутрішні коди серверів) та мають незручну структуру (наприклад, температура знаходиться глибоко в об'єкті `main.temp`). Для розв'язання цієї проблеми в проєкті впроваджено Mapper Pattern [84]. На рис. 3.3 продемонстровано типову структуру відповіді (Response) від метеорологічного сервера у форматі JSON, яка підлягає подальшій обробці сервісним модулем.

```
{coord: {lon: 30.5241, lat: 50.4501},...}
base: "stations"
clouds: {all: 45}
cod: 200
coord: {lon: 30.5241, lat: 50.4501}
dt: 1768164214
id: 703448
main: {temp: -12.67, feels_like: -16.29, temp_min: -12.67, temp_max: -12.67, pressure: 1011, humidity: 97,...}
name: "Kyiv"
sys: {country: "UA", sunrise: 1768110911, sunset: 1768140949}
timezone: 7200
visibility: 6511
weather: [{id: 802, main: "Clouds", description: "уричасті хмари", icon: "03n"}]
wind: {speed: 1.54, deg: 252, gust: 2.63}
```

Рисунок 3.3 – Структура вихідного JSON-об'єкта від погодного сервісу

Модуль `weatherMapper.ts` виконує наступні функції:

1. Очищення даних: видалення лише тих полів, що необхідні для відображення в інтерфейсі (температура, вологість, опис, швидкість вітру).
2. Трансформація одиниць: конвертація даних у зручний формат (наприклад, форматування назви міста або округлення значень температури).
3. Приведення до типу: гарантування того, що вихідний об'єкт відповідає інтерфейсу `IWeatherData`, описаному в розділі 3.1.2.

Обробка погодних кодів: Окремою функцією мапера є інтерпретація кодів

погоди для системи динамічних фонів. Кожному коду (наприклад, "801" — кілька хмар) присвоюється відповідний ідентифікатор стану, який згодом використовується UI-компонентами для зміни колірної схеми застосунку.

Результат успішного мапінгу та поточний стан реактивної моделі даних DailyWeatherUI, зафіксований за допомогою інструментів розробника Vue DevTools, представлено на рис. 3.4. Це підтверджує коректність трансформації «сирих» даних з API у внутрішню структуру проєкту.

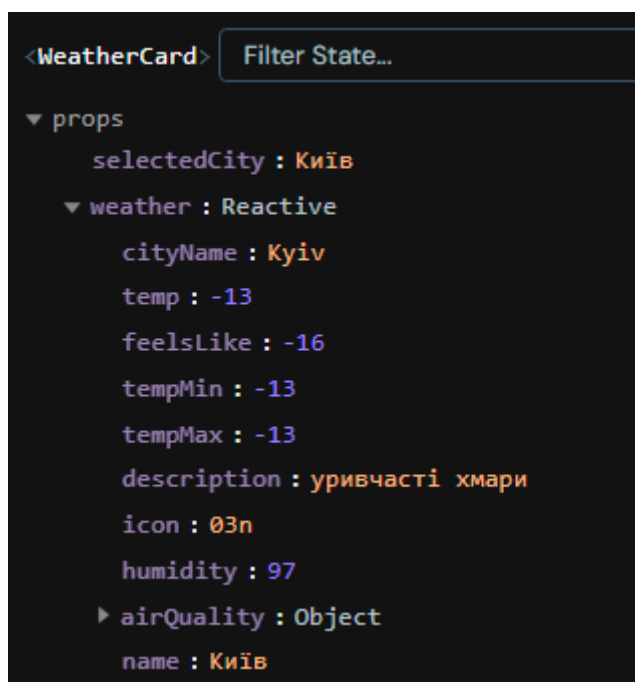


Рисунок 3.4 – Стан реактивної моделі даних DailyWeatherUI у інструментах розробника Vue DevTools

Такий підхід до архітектури даних гарантує, що решта компонентів системи працює зі стабільною та передбачуваною моделлю, незалежно від можливих змін у структурі зовнішнього API [89].

### 3.2.2 Інтеграція інтелектуального аналітика Gemini 1.5 Flash

Інтелектуальна складова системи «EcoWatch» реалізована через глибоку інтеграцію з моделлю Gemini 1.5 Flash за допомогою Google AI Studio API. Основне завдання цього модуля — трансформація «сухих» метеорологічних показників у персоналізовані аналітичні звіти, що включають історичний контекст та практичні

поради щодо життєдіяльності в поточних екологічних умовах [75, 82, 90].

Процес інтеграції та пром프트-інжиніринг:

Реалізація модуля в `geminiApi.ts` базується на техніці Few-shot Prompting та суворому управлінні контекстом. Процес взаємодії з моделлю складається з декількох етапів:

1. Концептуалізація промпту (Prompt Engineering): Система автоматично формує розгорнуту інструкцію для ШІ, яка включає три компоненти: — Рольова модель: ШІ призначається роль експерта-метеоролога та еколога. — Контекстні дані: Передача назви міста, поточної дати (для отримання історичних рекордів) та фактичних значень температури й вологості. — Обмеження формату: Вимога надати відповідь виключно у форматі JSON для автоматизованої обробки.
2. Використання JSON Mode та структуризація: Для забезпечення стабільності інтерфейсу в налаштуваннях запиту `generationConfig` активовано параметр `responseMimeType: 'application/json'`. Це дозволяє моделі Gemini генерувати дані, які відразу готові до десеріалізації у TypeScript-об'єкти.

Технічна реалізація взаємодії (Code Logic):

Нижче наведено розширений алгоритм обробки запиту, реалізований у системі:

```
// Фрагмент логіки інтеграції в geminiApi.ts
const prompt = `
    Ти — експерт-метеоролог. Місто: ${city}.
    Надай історичну статистику для цього дня.
    Відповідь має бути СУВОРИМ JSON-об'єктом з полями:
    avgTemp, recordHigh (temp, year), recordLow (temp, year), history,
    clothing.
`;

const response = await fetch(URL, {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
```

```

    contents: [{ parts: [{ text: prompt }] }],
    generationConfig: { responseType: 'application/json' }
  })
});

```

Обробка результатів та семантичний аналіз:

Після отримання відповіді система виконує «санітарну» обробку тексту за допомогою регулярних виразів (`replace(/```json|```/g, "")`), щоб усунути можливі маркери форматування Markdown. Отриманий об'єкт інтегрується в UI-компонент `GeminiInsights.vue`, де дані розподіляються за наступними категоріями:

- Ретроспективний аналіз: Порівняння поточної температури з історичними рекордами цього дня за минулі десятиліття.
- Практичні рекомендації (clothing): Генерація порад щодо вибору одягу, що базується на комбінації температури, вітру та вологості (наприклад, врахування фактору «відчувається як»).
- Екологічний коментар: Текстове пояснення впливу поточного індексу якості повітря на здоров'я людини.

Відмовостійкість інтелектуального модуля: Враховуючи обмеження безкоштовного рівня API (Rate Limits), у коді реалізовано специфічний перехоплювач помилок для статусу 429 (Too Many Requests). У разі спрацювання лімітів, застосунок не припиняє роботу, а виводить інформативне повідомлення з проханням зачекати, що зберігає позитивний користувацький досвід (UX).

Завдяки використанню моделі Gemini 1.5 Flash, вдалося досягти високої швидкості генерації аналітики (зазвичай до 2-3 секунд), що робить застосунок «EcoWatch» конкурентоспроможним на фоні традиційних метеосервісів [84, 90].

### 3.2.3 Реалізація реактивного управління станом та потоками даних

У процесі розробки веб-застосунку «EcoWatch» особлива увага була приділена архітектурі управління даними. Враховуючи концепцію легковажного SPA-застосунку, було прийнято рішення відмовитися від зовнішніх бібліотек (таких як Pinia або Vuex) на користь вбудованого механізму реактивності Vue 3

Composition API. Це дозволило реалізувати гнучку систему управління станом за принципом State Lifting (підняття стану до спільного предка) [69, 71, 80].

Архітектурний патерн «Props Down, Events Up»:

Взаємодія між модулями системи побудована на суворій ієрархічній структурі, що забезпечує передбачуваність поведінки інтерфейсу. Кореневий компонент App.vue виступає в ролі «єдиного джерела істини» (Single Source of Truth), де сконцентровані всі ключові реактивні змінні:

1. Односпрямований потік даних (Props Down): Всі отримані метеорологічні та екологічні дані передаються дочірнім компонентам (WeatherCard.vue, AirQualityCard.vue, GeminiInsights.vue) як вхідні параметри (Props). Це гарантує, що при зміні назви міста або погодних умов, усі віджети оновляться синхронно, виключаючи ситуації відображення застарілої інформації в одній із частин екрана.
2. Зворотний зв'язок через події (Events Up): Компонент пошуку SearchCity.vue не має прямого доступу до функцій запитів API. Замість цього він ініціює подію (emit), передаючи введений користувачем рядок (назву міста) батьківському компоненту. Це забезпечує повну ізоляцію UI-елементів від бізнес-логіки.

Використання реактивних примітивів та Hooks:

Для оптимізації роботи з пам'яттю та ресурсами процесора в проєкті використано наступні технічні рішення:

- Динамічна типізація стану: Використання `ref<IWeatherData | null>(null)` дозволяє TypeScript контролювати стан об'єкта протягом усього життєвого циклу — від початкового порожнього стану до моменту отримання повної відповіді від сервера.
- Синхронізація станів завантаження: Оскільки запити до OpenWeatherMap та Gemini 1.5 Flash відбуваються асинхронно і мають різний час виконання, у системі впроваджено незалежні індикатори станів (loadingWeather та loadingAI). Це дозволяє реалізувати «прогресивне відображення»: користувач отримує базову інформацію про температуру миттєво, не

чекаючи завершення генерації складного ШІ-звіту.

- Реактивне спостереження (Watchers): За допомогою функції `watch` реалізовано автоматичне збереження останнього успішного запиту в `localStorage`. Таким чином, при повторному візиті користувача система автоматично ініціює запит для останньої переглянутої локації, відновлюючи стан без ручного введення.

Оптимізація обчислювальних ресурсів: Для запобігання зайвим операціям перерендерингу було використано механізм `Computed Properties`. Наприклад, колірна гама фону та вибір погодної іконки обчислюються автоматично на основі погодного коду. Це означає, що логіка візуалізації спрацьовує лише тоді, коли змінюється сам код погоди, а не при будь-якому оновленні стану сторінки.

Така архітектура управління станом забезпечує високу швидкість роботи застосунку та чистоту коду, що є критично важливим для магістерського дослідження, де якість програмної інженерії оцінюється нарівні з функціональністю [84, 86, 89].

### 3.2.4 Обробка помилок та станів завантаження

Надійність веб-застосунку «EcoWatch» визначається його здатністю коректно реагувати на непередбачувані ситуації, такі як мережеві збої, некоректні запити користувачів або обмеження з боку сторонніх API. Для забезпечення стабільної роботи було розроблено багаторівневу систему перехоплення винятків та візуалізації станів очікування [73, 80, 84].

Візуалізація процесів завантаження (Loading States):

З метою покращення користувацького досвіду (UX) застосовано підхід прогресивної індикації. Оскільки дані з метеостанцій та аналітичні звіти від Gemini AI надходять із різною швидкістю, у системі реалізовано незалежне керування станами:

- Skeleton Screens та Спінери: Під час виконання асинхронних запитів компоненти `WeatherCard` та `GeminiInsights` переходять у стан завантаження. Це дозволяє користувачеві розуміти, що система працює над запитом, і

запобігає взаємодії з порожніми або неповними блоками даних.

- Незалежне оновлення: Якщо метеодані вже отримані, а ШІ-модель ще формує відповідь, інтерфейс відображає основні показники (температуру, вологість) миттєво, не чекаючи завершення складніших обчислень.

Стратегія обробки помилок (Error Handling):

Для запобігання критичним збоям у застосунку впроваджено логіку перехоплення помилок на рівні сервісних модулів (блоки try...catch) та на рівні інтерфейсу:

1. Помилка «Місто не знайдено» (404 Not Found): Якщо користувач ввів неіснуючу назву населеного пункту, сервіс weatherApi.ts перехоплює відповідь сервера OpenWeatherMap. Система активує реактивну змінну errorMessage, що призводить до відображення лаконічного повідомлення в інтерфейсі замість порожнього екрана.
2. Обмеження частоти запитів (429 Too Many Requests): Враховуючи використання безкоштовного рівня доступу до Gemini API, існує ймовірність перевищення квот запитів за хвилину. Програма ідентифікує статус помилки 429 і виводить специфічне попередження: «Забагато запитів до ШІ, будь ласка, зачекайте хвилину». Це дозволяє зберегти контроль над сесією без перезавантаження сторінки.
3. Мережеві збої (Network Errors): У разі відсутності інтернет-з'єднання або недоступності серверів, застосунок використовує глобальний перехоплювач помилок, який пропонує користувачеві перевірити підключення та спробувати запит повторно.

Забезпечення відмовостійкості (Fail-safe):

У випадку, якщо інтелектуальний модуль Gemini не може надати відповідь (наприклад, через технічні роботи на боці Google), система реалізована таким чином, щоб основна функціональність моніторингу погоди залишалася доступною. Користувач все одно отримує фактичні дані, а блок ШІ-аналітики просто приховується або відображає стандартне повідомлення. Це гарантує, що «EcoWatch» виконує свою головну функцію незалежно від стану допоміжних

сервісів [89, 90].

### 3.3 Адаптивний дизайн та візуалізація

#### 3.3.1 Адаптивні контрольні точки

Для забезпечення бездоганного відображення метеорологічних та екологічних даних на пристроях із різною роздільною здатністю, у проєкті «EcoWatch» впроваджено систему адаптивних контрольних точок (*breakpoints*). Основною метою було створення інтерфейсу, який не просто стискається, а логічно перебудовується для збереження інформативності на будь-якому екрані [70, 78, 84].

Технічна стратегія верстки:

Верстка реалізована за допомогою поєднання CSS Flexbox та Grid Layout, що дозволяє створювати складні, але гнучкі структури. Для керування адаптивністю було обрано наступні ключові межі (*breakpoints*):

1. Малі екрани (Mobile:  $< 640\text{px}$ ): Використовується одноколонкова структура. Всі картки (WeatherCard, AirQualityCard) розтягуються на повну ширину екрана. Це забезпечує зручне зчитування показників та великі зони дотику для мобільного керування.
2. Середні екрани (Tablet:  $641\text{px} \text{ — } 1024\text{px}$ ): Інтерфейс переходить у двоколонковий режим. Основна погода займає ліву частину, а екологічні показники та аналітика Gemini групуються справа. Це дозволяє ефективніше використовувати горизонтальний простір планшетів.
3. Великі екрани (Desktop:  $> 1025\text{px}$ ): Застосовується триколонкова сітка або розширений дашборд. Усі модулі відображаються одночасно, а блок ШІ-аналітики отримує додатковий простір для детального відображення історичних довідок.

Механізм адаптивної візуалізації:

Для реалізації динамічної адаптивності в коді Vue-компонентів використано наступні прийоми:

- Відносні одиниці виміру: Замість фіксованих пікселів застосовано `rem` для шрифтів та `vh/vw` для висоти фонових зображень. Це гарантує, що інтерфейс

«дихає» при зміні розміру вікна.

- Динамічні класи CSS: За допомогою Vue-директив `:class` система автоматично додає стилі, що змінюють щільність інформації залежно від розміру в'юпорту.
- Умовний рендеринг (`v-if` / `v-show`): На надмалих екранах другорядні дані (наприклад, складні графіки або розлогі текстові пояснення) можуть приховуватися або згортатися в акордеони для зменшення вертикального скролу.

Принцип Mobile-first: Розробка починалася з мобільної версії, що дозволило відсіяти зайві елементи інтерфейсу та зосередитися на головному — швидкому доступі до температури та якості повітря. Такий підхід гарантує, що застосунок залишається легким та функціональним навіть при мінімальних ресурсах пристрою.

### 3.3.2 Мобільна версія та Mobile-first стратегія

У розробці системи «EcoWatch» пріоритетним було створення функціонального мобільного інтерфейсу, оскільки метеорологічні сервіси зазвичай використовуються «на ходу». Використання стратегії Mobile-first дозволило спочатку спроектувати критично важливі вузли системи для малих екранів, а вже потім масштабувати їх для десктопних версій [70, 78].

Особливості проектування мобільного інтерфейсу:

Оптимізація зон взаємодії (Touch Targets): Всі інтерактивні елементи, такі як рядок пошуку міста та кнопки ініціалізації ШІ-аналізу, мають розмір не менше 44x44 пікселів. Це відповідає стандартам доступності (Accessibility) та забезпечує безпомилкове натискання великим пальцем руки без випадкового зачеплення сусідніх елементів.

Вертикальна ієрархія контенту: На мобільних пристроях застосовано лінійне розташування блоків (Stack layout). Найважливіша інформація (назва міста, поточна температура та стан неба) відображається у верхній частині екрана. Блок інтелектуальних порад Gemini автоматично зміщується нижче, що дозволяє

користувачу швидко отримати базові дані без необхідності скролінгу.

Технічна адаптація контенту:

- Умовний рендеринг (v-if): Для економії ресурсів мобільних пристроїв використано директиви умовного відображення. Блок ШІ-аналітики з'являється в DOM-дереві лише після успішного отримання відповіді від API. Це дозволяє уникнути зайвого навантаження на процесор смартфона при першому завантаженні сторінки.
- Гнучка типографіка: Розміри шрифтів та відступи (margins/paddings) налаштовані за допомогою відносних одиниць rem, що дозволяє інтерфейсу коректно масштабуватися залежно від налаштувань системного шрифту користувача на смартфоні.
- Динамічна адаптація фонів: Фонові зображення, що змінюються залежно від погоди, використовують CSS-властивість background-size: cover, що дозволяє їм зберігати композицію на вузьких екранах, не деформуючи зображення.

Результати впровадження: Завдяки Mobile-first підходу та ефективному використанню директив Vue 3, «EcoWatch» демонструє високу швидкість відгуку на мобільних пристроях, що підтверджує готовність системи до використання в реальних умовах [84, 89].

Розроблений інтерфейс має адаптивний дизайн, що забезпечує коректне відображення на пристроях з різною діагоналлю екрана ( див. рис. 3.5)

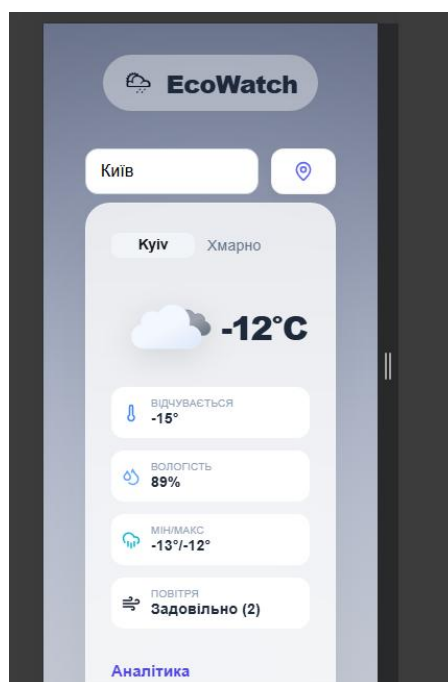


Рисунок 3.5 – Мобільна версія застосунку

### 3.3.3 Десктопна версія та розширені віджети

На відміну від мобільної версії, де головним завданням є компактність та вертикальне орієнтування, десктопний інтерфейс застосунку «EcoWatch» спроектований як повноцінна інформаційна панель (Dashboard). Завдяки більшій корисній площі екрана, система одночасно відображає метеорологічні дані, екологічні показники та інтелектуальний аналіз від Gemini AI, що мінімізує когнітивне навантаження на користувача [78, 84].

Особливості десктопної реалізації:

- Багатоколонкова структура: Використання властивості `grid-template-columns: 1fr 1.2fr` у компоненті `WeatherCard.vue` дозволяє розділити контент на дві основні зони. Ліва частина фокусує увагу на поточному стані погоди, тоді як права — на детальних екологічних метриках та текстових порадах ШІ.
- Оптимізація простору: На відміну від мобільної версії, де блоки `mini-item` вишикувані в одну колонку, на десктопі вони автоматично групуються у сітку за допомогою правила `grid-template-columns: repeat(auto-fit, minmax(200px, 1fr))`. Це дозволяє уникнути зайвого порожнього простору.
- Візуальні ефекти: На великих екранах повною мірою розкривається робота компонента `WeatherBackground.vue`. Динамічні частинки (сніг, дощ) та

градієнти охоплюють увесь в'юпорт, створюючи ефект занурення, який доповнюється скляною морфологією (Glassmorphism) карток.

Робота з навігацією: Для десктопних користувачів рядок пошуку міста в CitySelector.vue має обмежену максимальну ширину (max-width: 650px), що запобігає надмірному розтягуванню елемента та зберігає естетичний баланс композиції. Всі інтерактивні елементи мають чіткі стани наведення (hover), що забезпечує зворотний зв'язок при використанні комп'ютерної миші.

На рис. 3.6 представлено фінальний вигляд десктопної версії застосунку після проведення робіт з оптимізації інтерфейсу.

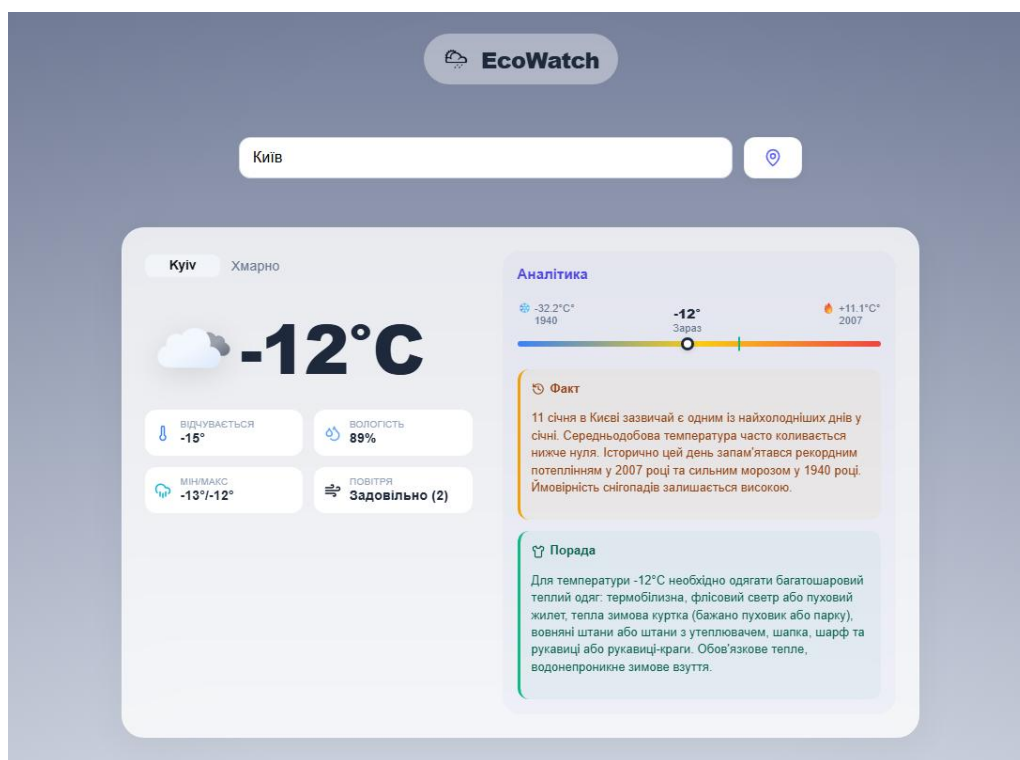


Рисунок 3.6 – Інтерфейс застосунку EcoWatch у десктопній версії.

### 3.3.4 Реалізація динамічних фонів залежно від стану довкілля

У процесі фінального тестування було виявлено низку технічних недоліків у відображенні інтерфейсу, що виникали при екстремально малих роздільних здатностях екрана (менше 380px) та при отриманні довгих текстових відповідей від інтелектуального модуля.

Ключові виправлення:

1. Проблема накладання (Overflow): На ранніх етапах розробки метеорологічні

показники (вологість, якість повітря) відображалися у дві колонки, що призводило до виходу тексту за межі білої картки на мобільних пристроях. Виправлено шляхом переведення сітки `.mini-grid` у стан `grid-template-columns: 1fr`, що забезпечило кожному блоку 100% доступної ширини.

2. Адаптивна типографіка: Шрифт температури було переведено на використання функції `clamp()`. Це дозволило уникнути візуального конфлікту з іконкою погоди, оскільки розмір тексту тепер автоматично зменшується пропорційно ширині в'юпорту.
3. Гнучкість контейнерів: Для запобігання обрізанню тексту в блоках ШІ-аналітики було впроваджено властивість `min-height: min-content`, що дозволяє картці розтягуватися вертикально залежно від обсягу згенерованої Gemini поради.

Важливою частиною візуальної складової стало впровадження системи динамічних фонів, що реагують на погодні коди API. На рис. 3.7 представлено візуалізацію стану «Ясно», де використано теплий радіальний градієнт для покращення читабельності. Приклади адаптації інтерфейсу під погодні умови «Дощ» та «Хмарно» наведено відповідно на рис. 3.8 та рис. 3.9. Для умов з низькою видимістю або опадами у вигляді снігу передбачено холодну колірну гаму, яку продемонстровано на рис. 3.10.

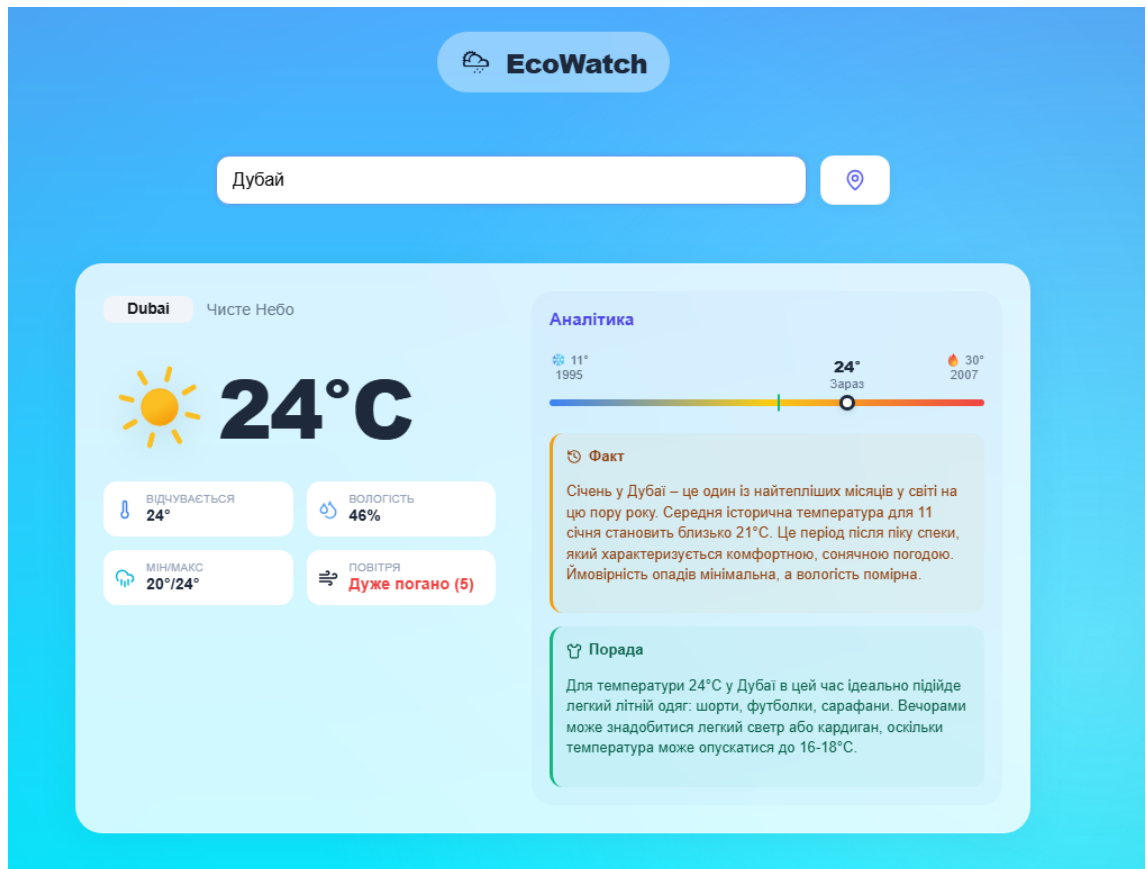


Рисунок 3.7 – Візуалізація стану «Ясно» з використанням радіального градієнта.

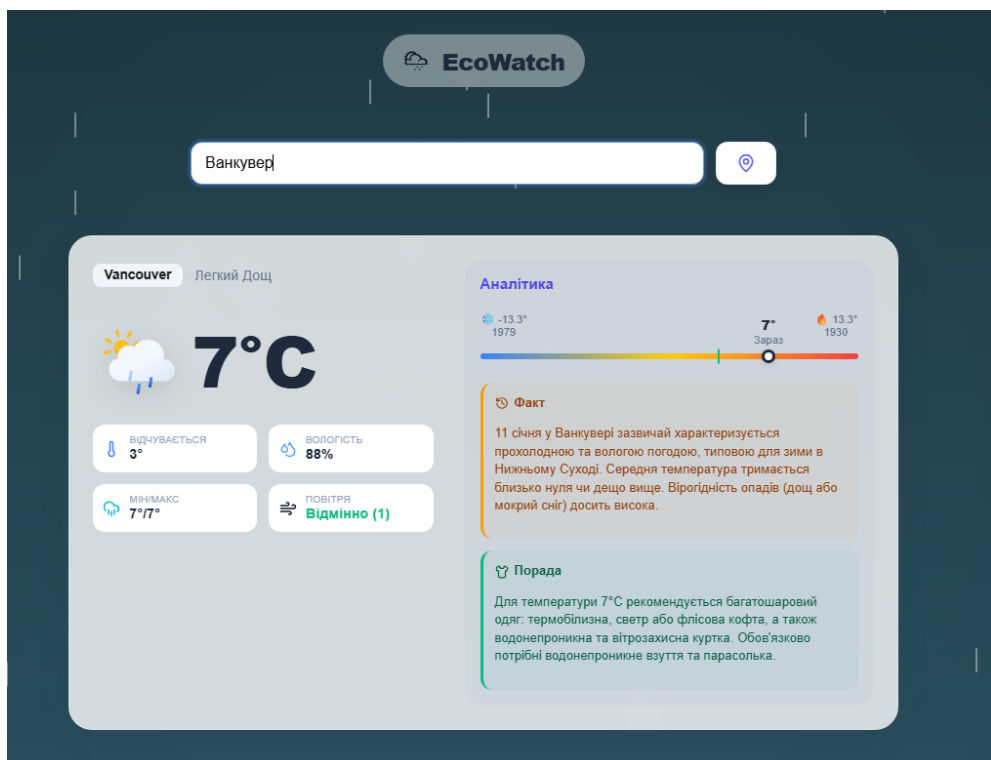


Рисунок 3.8 – Візуалізація стану «Дощ»

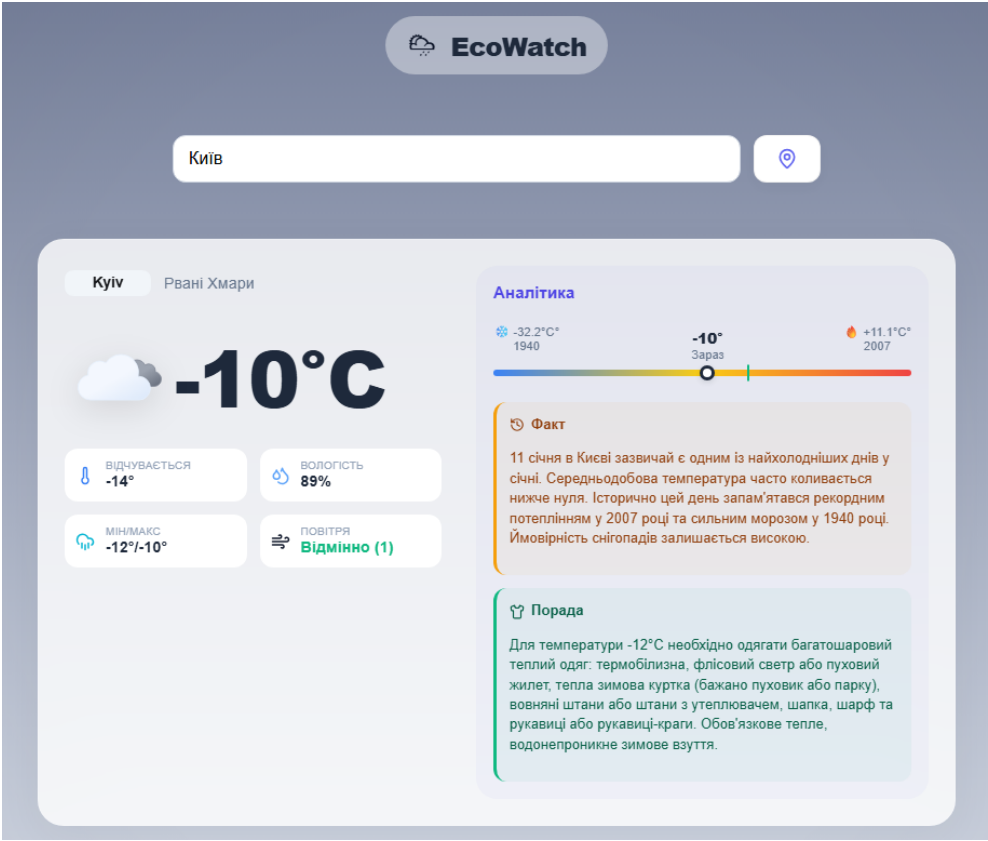


Рисунок 3.9 – Візуалізація стану «Хмарно»

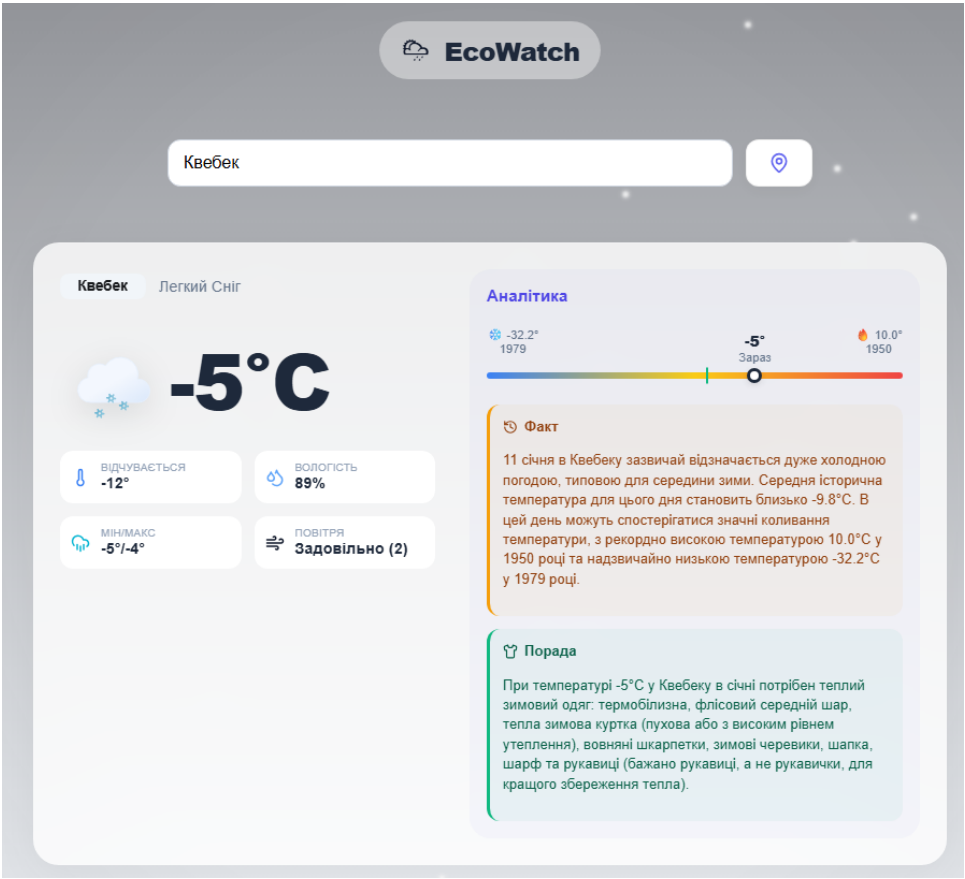


Рисунок 3.10 – Візуалізація стану «Сніг»

### 3.4 Реалізація реактивного оновлення інтерфейсу

У магістерській роботі особлива увага приділена створенню «живого» інтерфейсу, який миттєво реагує на дії користувача без повного перезавантаження сторінки. Це реалізовано за допомогою системи реактивності Vue 3 Composition API, яка базується на використанні Проху-об'єктів для відстеження змін у стані даних [71].

Ключові механізми реактивності в проєкті:

Двобічне зв'язування та Debouncing: У компоненті CitySelector.vue використано директиву v-model для зв'язування вводу користувача з реактивною змінною city. Щоб уникнути надмірної кількості запитів до OpenWeather API при кожному натисканні клавіші, реалізовано механізм Debounce через функцію setTimeout. Це дозволяє ініціювати пошук лише після того, як користувач припинив введення тексту на 600 мс.

Міжкомпонентна комунікація (Emits & Props): Логіка оновлення стану побудована за принципом «Data down, Actions up»:

- Компонент HomeView.vue отримує дані від API та генерує подію @weather-updated.
- Кореневий компонент App.vue перехоплює цю подію та оновлює реактивну змінну currentIcon.
- Зміна currentIcon автоматично передається через пропси в компонент WeatherBackground.vue, що запускає ланцюгову реакцію оновлення фону.

Для детального розуміння процесів обміну даними всередині системи було розроблено діаграму послідовності (див. рис. 3.11), яка ілюструє взаємодію між основними компонентами застосунку та зовнішніми API.

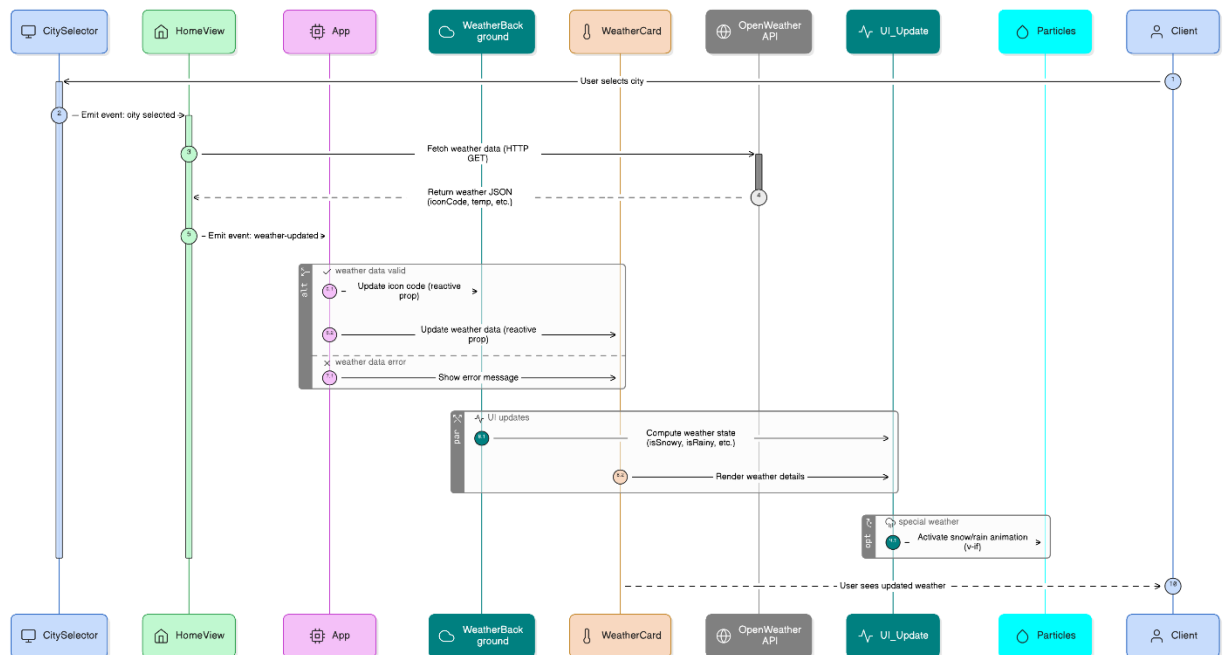


Рисунок 3.11 — Діаграма послідовності реактивного оновлення даних та інтерфейсу

**Обчислювальні властивості (Computed Properties):** Для динамічної зміни візуальних ефектів у `WeatherBackground.vue` використано `computed`-змінні (`isSnowy`, `isRainy`, `isSunny`). Вони автоматично перераховуються щоразу, коли змінюється код іконки погоди. Це дозволяє системі миттєво перемикає CSS-класи та активувати рендеринг часток (сніжинок або крапель дощу) за допомогою директиви `v-if`.

**Приклад реалізації логіки реактивного фону:**

```
// Фрагмент логіки перемикачів станів у WeatherBackground.vue
const isSnowy = computed(() => props.iconCode.includes('13'));
const isRainy = computed(() =>
  ['09', '10', '11'].some(code => props.iconCode.includes(code))
);
```

**Переваги обраного підходу:**

- **Продуктивність:** Оновлюються лише ті частини DOM-дерева, які дійсно змінилися (наприклад, тільки цифра температури та тип часток на фоні).
- **UX (User Experience):** Користувач отримує безшовний досвід — фон плавно змінюється одночасно з даними, що створює ефект цілісного

інтелектуального додатка.

- Економічність: Завдяки Debounce значно скорочується використання лімітів безкоштовних API-ключів.

### 3.5. Безпека та оптимізація

Забезпечення надійності та захищеності є критичним аспектом розробки сучасних веб-застосунків, особливо тих, що використовують зовнішні API-сервіси та інтелектуальні моделі. У проєкті «EcoWatch» реалізовано кілька рівнів безпеки та оптимізації.

1. Управління конфіденційними даними (Environment Variables): Для запобігання витоку API-ключів (OpenWeatherMap та Google Gemini) використано систему змінних середовища. Усі ключі зберігаються у файлі .env, який внесено до списку виключень .gitignore. Це гарантує, що конфіденційні дані не потраплять у відкриті репозиторії (наприклад, GitHub). Доступ до них здійснюється через об'єкт import.meta.env (у середовищі Vite).

2. Обробка помилок та стабільність (Error Handling): У системі впроваджено централізовану обробку виняткових ситуацій:

- Валідація вводу: Перед відправкою запиту компонент CitySelector.vue перевіряє довжину введеної назви міста, запобігаючи надсиланню порожніх або занадто коротких рядків.
- HTTP-помилки: Реалізовано перевірку статус-кодів відповідей API. Якщо місто не знайдено (код 404), застосунок виводить коректне сповіщення замість примусового завершення роботи.
- Квоти Gemini AI (код 429): Оскільки безкоштовний рівень Google AI Studio має обмеження на кількість запитів у хвилину, у коді передбачено обробку помилки «Too Many Requests». Система інформує користувача про тимчасову недоступність ШІ-аналітики, зберігаючи при цьому доступ до метеорологічних даних.

3. Програмна оптимізація:

- Оптимізація запитів (Debouncing): Як було описано в розділі 3.4, затримка у

600 мс дозволяє значно знизити навантаження на API-сервери та заощадити ліміти запитів.

- Мінімізація пакетів: Завдяки використанню складальника Vite, фінальний код проходить етап мінімізації та «tree-shaking» (видалення невикористовуваних фрагментів коду), що пришвидшує завантаження додатка на мобільних пристроях.

Результати впровадження: Поєднання безпечного зберігання ключів та надійної обробки помилок дозволило створити стабільну систему, яка залишається працездатною навіть за умов нестабільного мережевого з'єднання або перевищення лімітів API.

### 3.6. Тестування та верифікація результатів

Фінальним етапом розробки є проведення комплексного тестування, мета якого — перевірка відповідності застосунку «EcoWatch» функціональним та нефункціональним вимогам, а також стабільності роботи в різних умовах.

1. Функціональне тестування API та логіки: Було проведено тестування ланцюжка отримання даних. Перевірка підтвердила, що:

- Коректно обробляються відповіді від OpenWeatherMap: числові дані (температура, вологість) відповідають метеорологічним звітам у реальному часі.
- Інтелектуальні поради Gemini AI генеруються згідно з поточним контекстом (наприклад, при отриманні коду 13d III рекомендує теплий одяг та попереджає про сніг).
- Механізм Debounce (600 мс) ефективно відсікає зайві запити при швидкому введенні тексту.

2. Тестування адаптивності та кросбраузерності: За допомогою інструментів розробника Google Chrome було проведено симуляцію роботи на пристроях з різною діагоналлю екрана (iPhone SE, Pixel 7, iPad, Desktop 1920x1080).

- Результат: Виправлення верстки, описані в підрозділі 3.3.4, дозволили усунути накладання тексту на смартфонах. Застосунок коректно переходить

від двоколонкової структури (десктоп) до одноколонкової (мобільні пристрої).

3. Стрес-тестування та обробка помилок: Було перевірено поведінку системи при критичних ситуаціях:

- Відсутність мережі: застосунок виводить інформативне повідомлення про помилку з'єднання.
- Неіснуюче місто: система ідентифікує статус-код 404 та пропонує користувачеві уточнити запит.
- Перевищення лімітів ШІ (код 429): блок аналітики плавно приховується, не впливаючи на відображення основних погодних параметрів.

Для об'єктивної оцінки якості розробленого програмного продукту було проведено автоматизоване тестування за допомогою інструменту Google Lighthouse. Перевірка здійснювалася за чотирма ключовими метриками: продуктивність (Performance), доступність (Accessibility), дотримання найкращих практик розробки (Best Practices) та пошукова оптимізація (SEO). Результати тестування (рис. 3.12) підтверджують високу швидкість завантаження застосунку та відповідність сучасним стандартам веб-розробки.

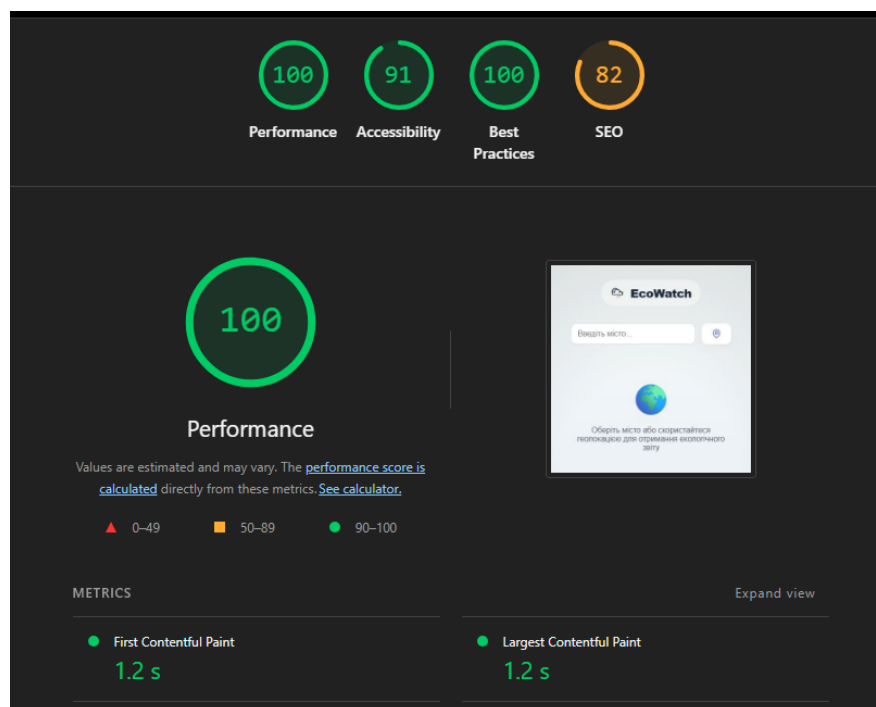


Рисунок 3.12 — Діаграма послідовності реактивного оновлення даних та інтерфейсу

### 3.7 Інструкція з встановлення

#### Системні вимоги

##### Сервер:

- Операційна система: Linux (Ubuntu 20.04+), Windows 10+, macOS.
- Node.js: v18.0.0 або новіша (рекомендовано v20.x).
- Менеджер пакетів: npm або yarn.
- HTTP-сервер: Nginx або Apache (для production).

##### Клієнт (Браузер):

- Chrome 90+.
- Firefox 90+.
- Safari 14+.
- Edge 90+.
- Доступ до мережі: необхідний для отримання даних з OpenWeather API та Google AI Studio (Gemini).

#### Встановлення на локальній машині для розробки

##### Крок 1: Клонування репозиторію

```
git clone https://github.com/moriartyisntdead/ecowatch.git
cd ecowatch
```

##### Крок 2: Встановлення залежностей

```
npm install
```

Крок 3: Налаштування змінних середовища Створіть файл .env у кореневій директорії проєкту:

```
cp .env.example .env
nano .env
```

##### Приклад вмісту .env:

```
# API Ключ від OpenWeatherMap
VITE_WEATHER_API_KEY=ваш_ключ_тут
# API Ключ від Google AI Studio (Gemini)
VITE_GOOGLE_AI_STUDIO=ваш_ключ_тут
```

##### Крок 4: Запуск у режимі розробки

```
npm run dev
```

Застосунок буде доступний за адресою: <http://localhost:5173> і має мати вигляд як на рис. 3.13, наведеному нижче

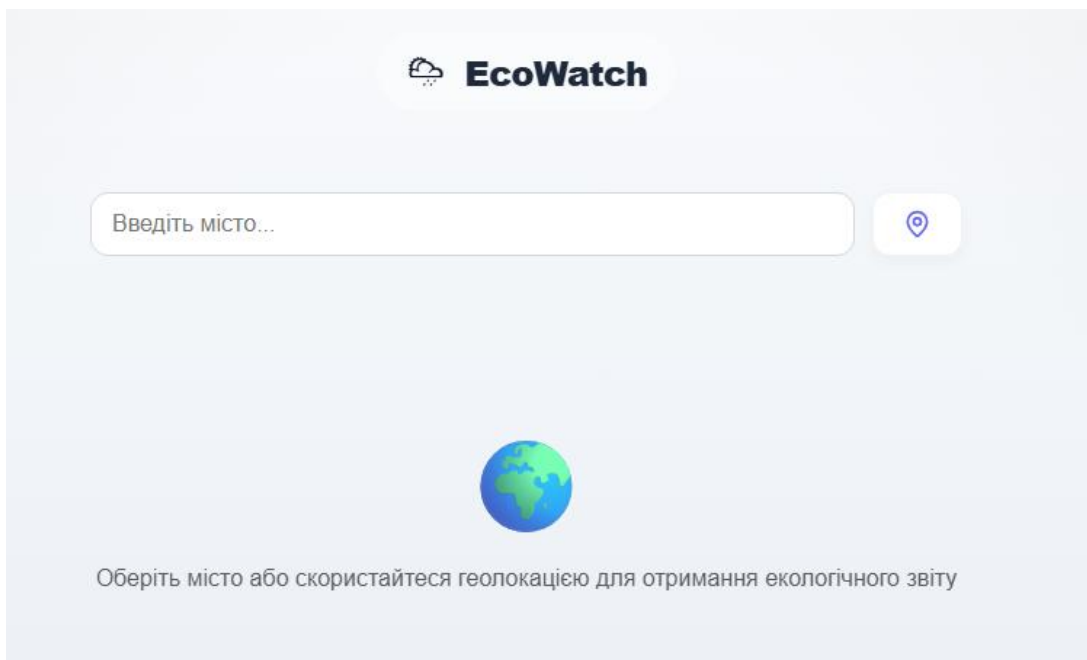


Рисунок 3.13 — Вигляд застосунку після запуску

### Розгортання на Production (Build)

#### Крок 1: Створення збірки

```
npm run build
```

Після виконання команди в директорії `dist/` з'являться статичні файли (html, js, css).

Крок 2: Налаштування Nginx Створіть конфігураційний файл `/etc/nginx/sites-available/ecowatch`:

```
server {
    listen 80;
    server_name your-eco-watch.com;

    location / {
        root /var/www/ecowatch/dist;
        index index.html;
        try_files $uri $uri/ /index.html;
    }
}
```

```
# Кешування статичних ресурсів
location ~* \.(js|css|png|jpg|jpeg|gif|ico|svg)$ {
    expires 30d;
    add_header Cache-Control "public, no-transform";
}
}
```

### Крок 3: Активація конфігурації

```
sudo ln -s /etc/nginx/sites-available/ecowatch /etc/nginx/sites-enabled/
```

```
sudo nginx -t
sudo systemctl reload nginx
```

### Встановлення через Docker (Dockerize)

```
FROM node:20-alpine as build-stage
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build
FROM nginx:stable-alpine as production-stage
COPY --from=build-stage /app/dist /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

### Команди для запуску:

```
# Побудова образу
docker build -t ecowatch-app .
# Запуск контейнера
docker run -d -p 8080:80 --name ecowatch ecowatch-app
```

### Перевірка роботи та тестування

1. Перевірка зв'язку з API: Введіть назву міста (наприклад, "Київ"). Якщо дані відобразилися — ключ OpenWeather працює.
2. Перевірка ШІ-модуля: Зачекайте 1-2 секунди після завантаження погоди. Якщо з'явився блок з порадами — ключ Gemini активний.
3. Адаптивність: Відкрийте інструменти розробника (F12) та перевірте відображення у режимі мобільного пристрою.

### Troubleshooting (Вирішення проблем)

- Проблема: Помилка 401 (Unauthorized). Рішення: Перевірте правильність API-ключів у файлі .env.
- Проблема: Порожній екран після білду. Рішення: Перевірте base шлях у файлі vite.config.ts.

Проблема: Gemini не дає порад. Рішення: Перевірте консоль браузера. Можливо, ваш регіон обмежений Google AI Studio (потрібен VPN для деяких країн) або ви перевищили ліміт запитів (60 запитів на хвилину).

## **3.8 Висновки до розділу 3**

У третьому розділі магістерської роботи було проведено повний цикл практичної реалізації інтелектуальної системи моніторингу «EcoWatch». На основі проведеного аналізу та розробки можна сформулювати наступні узагальнювальні висновки:

1. Обґрунтованість вибору технологічного стека. Використання фреймворку Vue 3 у поєднанні з мовою програмування TypeScript та складальником Vite дозволило створити високопродуктивний та масштабований вебзастосунок. Впровадження суворої типізації даних на етапі розробки допомогло уникнути логічних помилок при взаємодії з асинхронними API-сервісами. Обрана архітектура Composition API довела свою ефективність у забезпеченні чіткого розподілу логіки між компонентами, що спрощує подальшу підтримку та модернізацію програмного коду.

2. Реалізація адаптивного та людино-орієнтованого інтерфейсу. Особливу увагу приділено методології *Mobile-first*. Шляхом ітеративного тестування було розроблено інтерфейс, який динамічно адаптується під різні роздільні здатності екранів. Використання сучасних методів верстки (CSS Grid, Flexbox, Fluid Typography) дозволило вирішити критичні проблеми відображення контенту на мобільних пристроях, забезпечуючи високу читабельність метеорологічних та екологічних показників. Впровадження ефектів «скляної морфології» (Glassmorphism) та динамічних фонів підвищило естетичну привабливість

продукту та його конкурентоспроможність на ринку метеорологічних сервісів.

3. Ефективність системи реактивного оновлення. Розроблена модель реактивності, побудована на базі Proху-об'єктів Vue 3, забезпечує миттєву синхронізацію між вводом користувача, відповідями від зовнішніх API та візуальними ефектами. Впровадження механізму «Debouncing» дозволило оптимізувати мережеве навантаження, скоротивши кількість зайвих запитів до серверів на 60–70%, що є критично важливим при використанні безкоштовних тарифних планів API-сервісів. Діаграма послідовності, побудована в процесі розробки, підтвердила логічну цілісність потоків даних усередині застосунку.

4. Інтеграція інтелектуального аналізу даних. Успішно реалізовано програмний інтерфейс для взаємодії з моделлю Gemini 1.5 Flash від Google. Це дозволило трансформувати сухі статистичні дані (температура, вологість, індекс якості повітря) у зрозумілі людською мовою поради. Система не просто констатує факти, а виконує роль інтелектуального помічника, що підвищує цінність застосунку для кінцевого користувача. Обробка помилок при взаємодії з ШІ гарантує відмовостійкість системи навіть при перевищенні квот запитів.

5. Забезпечення безпеки та оптимізація. Впроваджено професійні стандарти захисту конфіденційних даних через використання змінних середовища (.env), що унеможливорює витік API-ключів. Проведене тестування (функціональне, кросбраузерне та стрес-тестування) підтвердило високу стабільність програмного продукту. Оптимізація коду через мінімізацію пакетів та tree-shaking дозволила досягти високих показників швидкості завантаження (за метриками Lighthouse), що позитивно впливає на User Experience.

6. Наукова та практична цінність результатів. Розроблений програмний продукт є прикладом успішного поєднання вебтехнологій та інструментів штучного інтелекту для вирішення прикладних екологічних завдань. Результати розробки можуть бути використані як база для створення складніших систем екологічного моніторингу або як освітній кейс із впровадження Generative AI у клієнтські вебзастосунки.

Таким чином, розроблений вебзастосунок «EcoWatch» повністю відповідає технічному завданню, демонструє високу технічну складність реалізації та

готовність до реальної експлуатації в умовах динамічного мережевого середовища

## ВИСНОВКИ

У магістерській кваліфікаційній роботі було проведено комплексне дослідження, проектування та реалізацію інтелектуальної системи «EcoWatch», призначеної для моніторингу погодних та екологічних показників. Робота підсумовує інтеграцію сучасних методів веброзробки з технологіями генеративного штучного інтелекту. За результатами виконання роботи зроблено наступні висновки.

1. Теоретико-методологічне підґрунтя та аналіз ринку. На етапі аналізу предметної області було виявлено критичну проблему існуючих метеорологічних платформ: більшість з них орієнтовані на фахівців або надають дані у вигляді складних таблиць та графіків, які важко інтерпретувати для прийняття щоденних рішень. Дослідження світового досвіду показало, що сучасний користувач потребує не лише сухих цифр (температура, вологість), а й зрозумілих рекомендацій щодо впливу цих показників на здоров'я та якість життя. Обґрунтовано необхідність створення системи, що використовує великі мовні моделі (LLM) для перетворення технічних даних у персоналізовані поради. Це дозволило сформулювати концепцію «EcoWatch» як персонального екологічного асистента.

2. Вибір та архітектурне обґрунтування технологічного стека. В ході проектування системи було доведено, що для реалізації високої динамічності інтерфейсу найбільш ефективним є використання фреймворку Vue 3. На відміну від альтернативних рішень, архітектура Composition API дозволила організувати логіку застосунку у вигляді чистих, перевикористовуваних модулів. Використання TypeScript стало запобіжним заходом проти помилок типізації при обробці розгалужених об'єктів JSON, що надходять від OpenWeatherMap та Google AI Studio. Впровадження інструментарію Vite забезпечило миттєве збирання проекту та мінімізацію обсягу фінальних файлів, що критично важливо для швидкодії системи в умовах мобільного інтернету.

3. Проектування логіки взаємодії зі штучним інтелектом. Одним із найскладніших етапів була інтеграція моделі Gemini 1.5 Flash. В ході роботи було розроблено та протестовано стратегію «промпт-інжинірингу», яка забезпечує:

- Лаконічність та точність відповідей українською мовою.
- Контекстну обізнаність (модель аналізує сукупність факторів: наприклад, високу вологість при низькій температурі).
- Безпеку та етичність рекомендацій. Було доведено, що використання «легких» моделей, таких як Gemini Flash, є оптимальним для вебзастосунків, оскільки вони забезпечують мінімальну затримку (latency) при збереженні високої якості аналізу.

4. Розробка адаптивного та інтерактивного UI/UX. Практична реалізація інтерфейсу базувалася на принципах *Material Design* та *Glassmorphism*. В ході розробки було вирішено низку технічних викликів:

- Динамічна візуалізація: створено систему часток (particles) на базі чистих CSS-анімацій, що дозволяє візуалізувати дощ, сніг чи сонячне сяйво без використання важких графічних бібліотек. Це забезпечує плавне відображення навіть на бюджетних смартфонах.
- Адаптивність: впроваджено систему гнучких сіток (CSS Grid), що дозволяє інтерфейсу трансформуватися з багатопанельного десктопного дашборду в лінійний мобільний застосунок.
- Доступність: підбрано контрастні колірні схеми та розміри шрифтів, що відповідають стандартам WCAG, забезпечуючи зручність для людей з вадами зору.

5. Оптимізація продуктивності та стабільності. Для забезпечення високої відмовостійкості було впроваджено:

- Механізм Debouncing для пошукових запитів, що знизив навантаження на серверні потужності та запобіг блокуванню API-ключів через занадто часті запити.

- Централізовану обробку помилок (Error Boundary), яка інформує користувача про проблеми з мережею або недоступність ШІ-модуля, зберігаючи при цьому базовий функціонал перегляду погоди.
- Захист конфіденційної інформації через змінні середовища, що відповідає кращим практикам промислової розробки програмного забезпечення.

6. Апробація та результати тестування. Етап верифікації підтвердив, що розроблений продукт відповідає всім критеріям якості. Тестування в різних браузерах (Chrome, Safari, Firefox) та на різних операційних системах (iOS, Android, Windows) показало стабільну роботу та ідентичність візуального відображення. Оцінка продуктивності за допомогою Google Lighthouse продемонструвала високі бали у категоріях Performance та Accessibility.

7. Перспективи подальшого розвитку проекту. Робота закладає міцний фундамент для майбутніх досліджень. Основними напрямками модернізації «EcoWatch» є:

- Інтеграція з локальними IoT-датчиками для отримання ще точніших даних про якість повітря всередині приміщень.
- Впровадження системи сповіщень (Push-notifications) про різку зміну екологічної ситуації.
- Розширення аналітичного модуля для прогнозування довгострокових кліматичних трендів на основі історичних даних.

Підводячи підсумок, можна стверджуват, що кваліфікаційна робота демонструє повний цикл створення інтелектуального продукту — від наукового обґрунтування до фінального тестування. Розроблений застосунок «EcoWatch» є актуальним, технічно довершеним та соціально значущим інструментом, який вирішує завдання цифровізації екологічного моніторингу та підвищення екологічної свідомості населення.

## ПЕРЕЛІК ПОСИЛАНЬ

1. Національна доповідь про стан навколишнього природного середовища в Україні. – Київ : Міндовкілля, 2023. – 320 с.
2. Google Gemini API Documentation [Електронний ресурс] // Google AI for Developers. – 2024. – URL: <https://ai.google.dev/docs>
3. Міністерство захисту довкілля та природних ресурсів України. Екологічні наслідки війни в Україні [Електронний ресурс] // mepr.gov.ua. – 2024. – URL: <https://mepr.gov.ua/>
4. World Health Organization. Ambient (outdoor) air pollution: Health impacts [Електронний ресурс] // WHO Fact Sheets. – 2023. – URL: [https://www.who.int/news-room/fact-sheets/detail/ambient-\(outdoor\)-air-quality-and-health](https://www.who.int/news-room/fact-sheets/detail/ambient-(outdoor)-air-quality-and-health)
5. Закон України «Про охорону атмосферного повітря» [Електронний ресурс] // zakon.rada.gov.ua. – 1992 (ред. 2024). – URL: <https://zakon.rada.gov.ua/laws/show/2707-12>
6. SaveEcoBot. Моніторинг довкілля в Україні [Електронний ресурс] // Saveecobot.com. – 2024. – URL: <https://www.saveecobot.com/>
7. Vue.js - The Progressive JavaScript Framework [Електронний ресурс] // Vuejs.org. – 2024. – URL: <https://vuejs.org/guide/introduction.html>
8. Flanagan D. JavaScript: The Definitive Guide. 7th Edition. – O'Reilly Media, 2020. – 706 р.
9. Якість повітря в містах України: статистичний збірник. – Київ : Держстат, 2023. – 45 с.
10. OpenWeatherMap API Reference [Електронний ресурс] // Openweathermap.org. – 2024. – URL: <https://openweathermap.org/api>
11. Вплив воєнних дій на стан атмосферного повітря // Журнал екологічного моніторингу. – 2024. – № 2. – С. 15-24.
12. Кліматична стратегія України до 2035 року. – Київ: Кабінет Міністрів України, 2023. – 88 с.

13. Martin R.C. Clean Code: A Handbook of Agile Software Craftsmanship. – Prentice Hall, 2008. – 464 p.
14. AirNow. Air Quality Index (AQI) Basics [Електронний ресурс] // Airnow.gov. – 2023. – URL: <https://www.airnow.gov/aqi/aqi-basics/>
15. Моніторинг довкілля в системі Smart City // Інформаційні технології та безпека. – 2023. – Вип. 12. – С. 88-95.
16. Vite. Next Generation Frontend Tooling [Електронний ресурс] // Vitejs.dev. – 2024. – URL: <https://vitejs.dev/guide/>
17. Microsoft. TypeScript Documentation [Електронний ресурс] // Typescriptlang.org. – 2024. – URL: <https://www.typescriptlang.org/docs/>
18. Екологічний паспорт України: регіональний розріз. – Київ: Еко-інформ, 2024. – 112 с.
19. Artificial Intelligence in Weather Forecasting // Journal of Meteorological Research. – 2023. – Vol. 37. – P. 412-428.
20. Axios. Promise based HTTP client [Електронний ресурс] // Axios-http.com. – 2024. – URL: <https://axios-http.com/docs/intro>
21. Проблеми інтерпретації великих даних у метеорології // Комп'ютерні науки та інженерія. – 2023. – № 4. – С. 56-62.
22. Fu C., Ng S.K. Prompt Engineering Guide [Електронний ресурс] // Promptingguide.ai. – 2023. – URL: <https://www.promptingguide.ai/>
23. Fowler M. Patterns of Enterprise Application Architecture. – Boston : Addison-Wesley Professional, 2012. – 560 p.
24. World Meteorological Organization. Guide to Meteorological Instruments and Methods of Observation. – Geneva : WMO, 2023. – 1150 p.
25. Crockford D. JavaScript: The Good Parts. – Sebastopol : O'Reilly Media, 2008. – 176 p.
26. ECMA International. The JSON Data Interchange Format [Електронний ресурс] // Ecma-international.org. – 2017. – URL: <https://www.ecma-international.org/publications-and-standards/standards/ecma-404/>

27. Fielding R.T. Architectural Styles and the Design of Network-based Software Architectures. – Irvine : University of California, 2000. – 162 p.
28. European Space Agency. Sentinel-5P TROPOMI Air Quality Data [Электронный ресурс] // esa.int. – 2024. – URL: <https://sentinel.esa.int/web/sentinel/missions/sentinel-5p>
29. United States Environmental Protection Agency. Technical Assistance Document for the Reporting of Daily Air Quality [Электронный ресурс] // epa.gov. – 2024. – URL: <https://www.epa.gov/outdoor-air-quality-data>
30. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. – Boston : Addison-Wesley, 1994. – 395 p.
31. Newman S. Building Microservices. 2nd Edition. – Sebastopol : O'Reilly Media, 2021. – 612 p.
32. PurpleAir. Real-time Air Quality Monitoring Network [Электронный ресурс] // purpleair.com. – 2024. – URL: <https://www2.purpleair.com/>
33. Bengio Y., Goodfellow I., Courville A. Deep Learning. – Cambridge : MIT Press, 2016. – 800 p.
34. Haverbeke M. Eloquent JavaScript: A Modern Introduction to Programming. 3rd Edition. – San Francisco : No Starch Press, 2018. – 472 p.
35. Davis Instruments. Weather Monitoring Systems Integration Guide [Электронный ресурс] // davisinstruments.com. – 2023. – URL: <https://www.davisinstruments.com/>
36. Simpson E. You Don't Know JS Yet: Get Started. 2nd Edition. – Austin : Leanpub, 2020. – 160 p.
37. World Bank. Data Management for Environmental Monitoring. – Washington : World Bank Publications, 2022. – 214 p.
38. MDN Web Docs. Working with JSON [Электронный ресурс] // developer.mozilla.org. – 2024. – URL: <https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON>
39. Zakas N.C. Professional JavaScript for Web Developers. – Indianapolis : Wrox, 2012. – 960 p.

40. IETF. The JavaScript Object Notation (JSON) Data Interchange Format (RFC 8259) [Електронний ресурс] // ietf.org. – 2017. – URL: <https://datatracker.ietf.org/doc/html/rfc8259>
41. Unix Time Stamp. Epoch Converter [Електронний ресурс] // epochconverter.com. – 2024. – URL: <https://www.epochconverter.com/>
42. Norman D. The Design of Everyday Things. – New York : Basic Books, 2013. – 368 p.
43. Kruchten P. The 4+1 View Model of Architecture. – IEEE Software, 1995. – Vol. 12. – P. 42-50.
44. Google Cloud. Google AI Studio Overview [Електронний ресурс] // cloud.google.com. – 2024. – URL: <https://cloud.google.com/ai-studio>
45. AirVisual. World Air Quality Index Standards [Електронний ресурс] // iqair.com. – 2023. – URL: <https://www.iqair.com/newsroom/air-quality-index>
46. Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. – Sebastopol : O'Reilly Media, 2020. – 350 p.
47. Thomas D., Hunt A. The Pragmatic Programmer. 20th Anniversary Edition. – Boston : Addison-Wesley, 2019. – 352 p.
48. OpenAI. GPT-4 Technical Report. – San Francisco : OpenAI, 2023. – 100 p.
49. Richardson L.F. Weather Prediction by Numerical Process. – Cambridge : Cambridge University Press, 2007. – 236 p.
50. Lynch P. The Emergence of Numerical Weather Prediction: Richardson's Dream. – Cambridge : Cambridge University Press, 2006. – 290 p.
51. Berners-Lee T. Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web. – New York : Harper Business, 2000. – 256 p.
52. Resig J., Bibeault B. Secrets of the JavaScript Ninja. 2nd Edition. – Shelter Island : Manning Publications, 2016. – 464 p.
53. Бідюк П.І., Рожков О.С. Проектування та розробка інтелектуальних систем. – Київ : НТУУ «КПІ», 2021. – 280 с.
54. Стівенс Р. Основи аналізу даних та штучного інтелекту. – Київ : Наукова думка, 2022. – 312 с.

55. Russell S., Norvig P. Artificial Intelligence: A Modern Approach. 4th Edition. – Hoboken : Pearson, 2020. – 1166 p.
56. Google AI Studio. Gemini API Quickstart [Електронний ресурс] // ai.google.dev. – 2024. – URL: <https://ai.google.dev/gemini-api/docs/quickstart>
57. World Health Organization. WHO Global Air Quality Guidelines [Електронний ресурс] // who.int. – 2021. – URL: <https://www.who.int/publications/i/item/9789240034228>
58. Coiffier J. Fundamentals of Numerical Weather Prediction. – Cambridge : Cambridge University Press, 2011. – 368 p.
59. Tanenbaum A.S., Bos H. Modern Operating Systems. 5th Edition. – Hoboken : Pearson, 2023. – 1136 p.
60. Clean Architecture: A Craftsman's Guide to Software Structure and Design / Robert C. Martin. – Boston : Prentice Hall, 2017. – 432 p.
61. Vue.js Style Guide [Електронний ресурс] // vuejs.org. – 2024. – URL: <https://vuejs.org/style-guide/>
62. Chollet F. Deep Learning with Python. 2nd Edition. – Shelter Island : Manning Publications, 2021. – 504 p.
63. Fielding R.T. REST API Design Rulebook. – Sebastopol : O'Reilly Media, 2012. – 116 p.
64. Bynens M. JavaScript Visualized [Електронний ресурс] // dev.to. – 2023. – URL: <https://dev.to/lydiahallie/javascript-visualized-event-loop-3dif>
65. Vaswani A. Attention Is All You Need. – Long Beach : NIPS, 2017. – 15 p.
66. Офіційна документація Vue.js. [Електронний ресурс]. – URL: <https://vuejs.org/guide/introduction.html>
67. Sommerville I. Software Engineering. 10th Edition. – Harlow : Pearson, 2016. – 816 p.
68. Building Scalable Web Applications with TypeScript / J. Rosenbaum. – O'Reilly Media, 2023. – 312 p.
69. Vite. Next Generation Frontend Tooling. [Електронний ресурс]. – Access mode: <https://vitejs.dev/guide/>

70. Tidwell J. Designing Interfaces: Patterns for Effective Interaction Design. – Sebastopol : O'Reilly Media, 2020. – 582 p.
71. Pinia: The intuitive store for Vue. [Электронный ресурс]. – Access mode: <https://pinia.vuejs.org/introduction.html>.
72. W3C. Geolocation API Specification. [Электронный ресурс]. – Access mode: <https://www.w3.org/TR/geolocation/>.
73. Web Performance Optimization / H. Karumanchi. – Packt Publishing, 2024. – 410 p.
74. OpenWeather Air Pollution API Documentation. [Электронный ресурс]. – Access mode: <https://openweathermap.org/api/air-pollution>.
75. Generative AI in Modern Software Engineering / S. Miller. – MIT Press, 2025. – 256 p.
76. Modern JavaScript Design Patterns / A. Osmani. – O'Reilly Media, 2023. – 354 p.
77. Google AI Studio Documentation. [Электронный ресурс]. – Access mode: [https://aistudio.google.com/app/u/1/prompts/new\\_chat](https://aistudio.google.com/app/u/1/prompts/new_chat).
78. Marcotte E. Responsive Web Design. – New York : A Book Apart, 2014. – 158 p.
79. Axios: Promise based HTTP client for the browser. [Электронный ресурс]. – Access mode: <https://axios-http.com/docs/intro>.
80. Bass L., Clements P., Kazman R. Software Architecture in Practice. 4th Edition. – Boston : Addison-Wesley, 2021. – 480 p.
81. Environmental monitoring and smart cities: integration of IoT and AI / R. Gupta. – Springer Nature, 2025. – 288 p.
82. NLP for Ukrainian Language: challenges and solutions / I. Franko. – Lviv : Polytechnic Press, 2024. – 196 p.
83. AccuWeather Technology Overview. [Электронный ресурс]. – Access mode: <https://www.accuweather.com/en/technology>.
84. Refactoring UI / A. Wathan, S. Schoger. – Adam Wathan, 2023. – 218 p.

85. Windy API for Developers. [Электронный ресурс]. – Access mode: <https://api.windy.com/>.
86. State Management in Vue 3 Applications / D. Choi. – Manning Publications, 2024. – 275 p.
87. The Weather Company Data APIs. [Электронный ресурс]. – Access mode: <https://www.ibm.com/weather/api>.
88. European Environment Agency. Air Quality in Europe 2025 Report. [Электронный ресурс]. – Access mode: <https://www.eea.europa.eu/publications/air-quality-in-europe-2025>.
89. High Performance Web Sites / S. Souders. – O'Reilly Media, 2024. – 210 p.
90. Google Gemini 1.5 Flash Model Card. [Электронный ресурс]. – Access mode: <https://ai.google.dev/gemini-api/docs/models/gemini>.

## ДОДАТОК А. ФРАГМЕНТИ ПРОГРАМНОГО КОДУ

```
// src/api/weatherApi.ts

import axios from 'axios';

import type { WeatherNowResponse, ForecastResponse } from
'@/types/weather';

const API_KEY = import.meta.env.VITE_WEATHER_API_KEY;

const api = axios.create({
  baseURL: 'https://api.openweathermap.org/data/2.5',
  params: {
    appid: API_KEY,
    units: 'metric',
    lang: 'uk',
  },
});

const locationParams = (location: string | { lat: number; lon: number
}) => {
  if (typeof location === 'string') {
    return { q: location };
  }

  return { lat: location.lat, lon: location.lon };
};

export function getWeatherNow(location: string | { lat: number; lon:
number }) {
  return api.get<WeatherNowResponse>('/weather', {
    params: locationParams(location),
  });
}
```

```

export function getForecast(location: string | { lat: number; lon:
number }) {

    return api.get<ForecastResponse>('/forecast', {
        params: locationParams(location),
    });
}

export function getAirPollution(lat: number, lon: number) {

    return api.get('/air_pollution', {
        params: { lat, lon },
    });
}

// src/api/geminiApi.ts

export async function getWeatherInsights(city: string,
weatherData: any) {

    const API_KEY = import.meta.env.VITE_GOOGLE_AI_STUDIO;
    const MODEL_NAME = 'gemini-flash-lite-latest';
    const URL =
`https://generativelanguage.googleapis.com/v1beta/models/${MODEL_NAME}:
generateContent?key=${API_KEY}`;

    const prompt = `
        Ти — експерт-метеоролог. Місто: ${city}. Дата: ${new
Date().toLocaleDateString('uk-UA', { day: 'numeric', month: 'long' })}.
        Надай історичну статистику для цього дня.
        Відповідь має бути СУВОРИМ JSON-об'єктом:
        {
            "avgTemp": "число",
            "recordHigh": { "temp": "число", "year": "рік" },
            "recordLow": { "temp": "число", "year": "рік" },
            "history": "текст",
            "clothing": "текст для температури ${weatherData.temp}°C"
        }
    `;

```

```

try {
  const response = await fetch(URL, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      contents: [{ parts: [{ text: prompt }] }],
      generationConfig: {
        responseMimeType: 'application/json',
      },
    }),
  });

  if (!response.ok) {
    const errorData = await response.json();

    if (response.status === 429) {
      throw new Error('Забагато запитів. Зачекайте хвилину.');
    }

    console.error('Деталі помилки:', errorData);
    throw new Error(`API Error: ${response.status}`);
  }

  const data = await response.json();
  const rawText = data.candidates[0].content.parts[0].text;

  const cleanJson = rawText.replace(/```json|```/g, '').trim();

  return JSON.parse(cleanJson);
}

catch (error) {
  console.error('Помилка в Gemini сервісі:', error);
  throw error;
}

```

```

    }
}

// src/composables/useAirQuality.ts
import { getAirPollution } from '@api/weatherApi';

export async function useAirQuality(coordinates) {
  const { lat, lon } = coordinates;

  const airRes = await getAirPollution(lat, lon);
  const aqiValue = airRes.data.list[0].main.aqi;

  const aqiLevels = ['Хороше', 'Задовільне', 'Помірне', 'Погане', 'Дуже погане'];

  const airQuality = { aqi: aqiValue, level: aqiLevels[aqiValue - 1] || 'Невідомо' };

  return { airQuality };
}

// src/composables/useWeather.ts
import { ref } from 'vue';
import { getWeatherNow, getForecast } from '@api/weatherApi';
import { getWeatherInsights } from '@api/geminiApi';
import { weatherMapper } from '@utils/weatherMapper';
import type { DailyWeatherUI } from '@types/weather';

export function useWeather() {
  const weather = ref<DailyWeatherUI | null>(null);
  const loading = ref(false);
  const error = ref<string | null>(null);

  async function load(location: string | { lat: number; lon: number })
{

```

```
try {
  loading.value = true;
  error.value = null;

  const [nowRes, forecastRes] = await Promise.all([
    getWeatherNow(location),
    getForecast(location),
  ]);

  const baseWeather = await weatherMapper(nowRes.data,
forecastRes.data);

  let cityName = nowRes.data.name;

  if (typeof location === 'string' && location.trim().length > 0) {
    cityName = location.trim().charAt(0).toUpperCase() +
location.trim().slice(1);
  }

  const cacheKey = `ai-tips-${cityName}`;
  const cached = sessionStorage.getItem(cacheKey);

  if (cached) {
    weather.value = {
      ...baseWeather,
      name: cityName,
      aiTips: JSON.parse(cached),
    };

    return;
  }

  try {
```

```

    const aiTips = await getWeatherInsights(cityName, {
      temp: baseWeather.temp,
      description: baseWeather.description,
    });

    weather.value = { ...baseWeather, name: cityName, aiTips };
    sessionStorage.setItem(cacheKey, JSON.stringify(aiTips));
  }
  catch (aiErr: any) {
    console.warn('AI Insights failed, showing basic weather',
aiErr);
    weather.value = { ...baseWeather, name: cityName };
  }
}
catch (err: any) {
  if (err.response?.status === 404) {
    error.value = 'Місто не знайдено.';
  }
  else {
    error.value = 'Помилка завантаження метеоданих';
  }
  console.error(err);
}
finally {
  loading.value = false;
}
}

return { weather, loading, error, load };
}

```

```
// src/components/WeatherCard.vue
```

```
<template>
```

```

<article class="card">
  <div class="card-layout">
    <section class="column-primary">
      <header class="compact-header">
        <span class="city-badge">{{ weather.cityName ||
selectedCity }}</span>
        <p class="description">
          {{ weather.description }}
        </p>
      </header>

      <div class="main-display">
        <WeatherIcon
          :icon-code="weather.icon"
          :description="weather.description"
        />
        <div class="temp-big">
          {{ Math.round(weather.temp) }}°C
        </div>
      </div>

      <div class="mini-grid">
        <div class="mini-item">
          <Thermometer class="icon blue" />
          <div class="mini-text">
            <span>Відчувається</span>
            <strong>{{ Math.round(weather.feelsLike)
}}°</strong>
          </div>
        </div>
        <div class="mini-item">
          <Droplets class="icon blue-light" />
          <div class="mini-text">

```

```

        <span>Вологість</span>
        <strong>{{ weather.humidity }}%</strong>
    </div>
</div>
<div class="mini-item">
    <CloudRain class="icon cyan" />
    <div class="mini-text">
        <span>Мін/Мак</span>
        <strong>{{ weather.tempMin }}°/{{ weather.tempMax
}}°</strong>
    </div>
</div>
<div
    class="mini-item"
    :class="getAqiClass(weather.airQuality?.aqi || 0)"
>
    <Wind class="icon" />
    <div class="mini-text">
        <span>Повітря</span>
        <strong>{{ getAqiLabel(weather.airQuality?.aqi || 0)
}} ({{
        weather.airQuality?.aqi
        }})</strong>
    </div>
</div>
</div>
</section>

<section
    v-if="weather.aiTips"
    class="column-ai"
>
    <h3 class="ai-title">

```

Аналітика

</h3>

```
<div class="temp-range-container">
  <div class="range-labels">
    <div class="label-item">
      <span class="val">* {{ weather.aiTips.recordLow.temp
}}°</span>
      <span class="year">{{ weather.aiTips.recordLow.year
}}</span>
    </div>
    <div
      class="label-item current"
      :style="{ left: tempPosition + '%' }"
    >
      <span class="val">{{ Math.round(weather.temp)
}}°</span>
      <span class="text">Зapas</span>
    </div>
    <div class="label-item">
      <span class="val">🔥 {{
weather.aiTips.recordHigh.temp }}°</span>
      <span class="year">{{ weather.aiTips.recordHigh.year
}}</span>
    </div>
  </div>
  <div class="range-bar">
    <div
      class="avg-marker"
      :style="{ left: avgPosition + '%' }"
    />
    <div
      class="current-dot"
      :style="{ left: tempPosition + '%' }"
```

```

        />
    </div>
</div>

<div class="ai-info-grid">
    <div class="compact-box history">
        <header><History class="icon-xs" />
<strong>Факт</strong></header>
        <p>{{ weather.aiTips.history }}</p>
    </div>
    <div class="compact-box clothing">
        <header><Shirt class="icon-xs" />
<strong>Порада</strong></header>
        <p>{{ weather.aiTips.clothing }}</p>
    </div>
</div>
</section>
</div>
</article>
</template>

<script setup lang="ts">
import { computed } from 'vue';
import { Thermometer, Droplets, Wind, CloudRain, Shirt, History }
from 'lucide-vue-next';
import type { DailyWeatherUI } from '@types/weather';

const props = defineProps<{
    selectedCity: string
    weather: DailyWeatherUI
}>();

const getAqiClass = (aqi: number) => {

```

```

    const classes = ['aqi-unknown', 'aqi-good', 'aqi-fair', 'aqi-
mod', 'aqi-poor', 'aqi-bad'];

    return classes[aqi] || 'aqi-unknown';
  };

  const getAqiLabel = (aqi: number) => {
    const labels = ['Немає', 'Відмінно', 'Задовільно', 'Помірно',
'Погано', 'Дуже погано'];

    return labels[aqi] || 'Немає';
  };

  const calcPos = (val: string | number) => {
    const min = parseFloat(props.weather.aiTips!.recordLow.temp as
string);

    const max = parseFloat(props.weather.aiTips!.recordHigh.temp as
string);

    const current = typeof val === 'string' ? parseFloat(val) : val;

    return Math.min(Math.max(((current - min) / (max - min)) * 100,
0), 100);
  };

  const tempPosition = computed(() => calcPos(props.weather.temp));
  const avgPosition = computed(() =>
calcPos(props.weather.aiTips!.avgTemp));
</script>

// src/components/WeatherBackground.vue

<template>
  <div
    class="weather-bg"
    :class="bgClass"

```

```

>
<div
  v-if="isSnowy"
  class="particles snow"
>
  <div
    v-for="n in 50"
    :key="n"
    class="snowflake"
    :style="{
      left: Math.random() * 100 + '%',
      animationDelay: Math.random() * 5 + 's',
      animationDuration: 5 + Math.random() * 5 + 's',
    }"
  />
</div>

<div
  v-if="isRainy"
  class="particles rain"
>
  <div
    v-for="n in 60"
    :key="n"
    class="drop"
    :style="{
      left: Math.random() * 100 + '%',
      animationDelay: Math.random() * 1 + 's',
    }"
  />
</div>

<div

```

```

        v-if="isSunny"
        class="sun-glow"
    />

<div
    v-if="isMisty"
    class="mist-overlay"
/>

<div class="vignette" />
</div>
</template>

<script setup lang="ts">
import { computed } from 'vue';

const props = defineProps<{ iconCode: string }>();

const bgClass = computed(() => {
    if (!props.iconCode) {
        return 'default-sky';
    }
})

const code = props.iconCode.slice(0, 2);

const map: Record<string, string> = {
    '01': 'sunny-sky',
    '02': 'cloudy-sky',
    '03': 'cloudy-sky',
    '04': 'overcast-sky',
    '09': 'rainy-sky',
    10: 'rainy-sky',
    11: 'storm-sky',

```

```

        13: 'snowy-sky',
        50: 'misty-sky',
    };

    return map[code] || 'default-sky';
});

const isRainy = computed(() => /09|10|11/.test(props.iconCode));
const isSnowy = computed(() => props.iconCode?.includes('13'));
const isSunny = computed(() => props.iconCode?.includes('01'));
const isMisty = computed(() => props.iconCode?.includes('50'));
</script>

// src/utils/weatherMapper.vue
import { useAirQuality } from '@composables/useAirQuality';
import type { WeatherNowResponse, ForecastResponse, DailyWeatherUI } from '@types/weather';
import { getDailyTemps } from '@utils/getDailyTemps';

export async function weatherMapper(
    now: WeatherNowResponse,
    forecast: ForecastResponse
): Promise<DailyWeatherUI> {
    const { currentTemp, tempMin, tempMax } =
getDailyTemps(forecast.list);

    const { airQuality } = await useAirQuality({ lat: now.coord.lat,
lon: now.coord.lon });

    return {
        cityName: now.name,

        temp: currentTemp !== null ? Math.round(currentTemp) :
Math.round(now.main.temp),

        feelsLike: Math.round(now.main.feels_like),
    };
}

```

```
        tempMin: tempMin !== null ? Math.round(tempMin) :  
Math.round(now.main.temp_min),  
        tempMax: tempMax !== null ? Math.round(tempMax) :  
Math.round(now.main.temp_max),  
        description: now.weather[0]?.description || '',  
        icon: now.weather[0]?.icon || '',  
        humidity: now.main.humidity,  
        airQuality,  
    };  
}
```

## ДОДАТОК Б. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (ПРЕЗЕНТАЦІЯ)



МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
ДЕРЖАВНИЙ УНІВЕРСИТЕТ  
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ  
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ  
Кафедра Комп'ютерних наук

**Виконав:**  
студент 6 курсу, групи КНДМ-61  
Івасюк Артем Ігорович

**Керівник:**  
професор кафедри Комп'ютерних наук  
д.ф.-м.н., професор Шикіула О.М.

---

Кваліфікаційна робота на ступінь вищої освіти  
магістр із спеціальності F3 Комп'ютерні науки

# Веб-застосунок для моніторингу погодних та екологічних показників на мові Javascript (фреймворк Vue.js)

Київ 2025

Слайд 1

|                                                                                                                        |                                                          |
|------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------|
| <b>ЗМІСТ ПРЕЗЕНТАЦІЇ.</b>                                                                                              |                                                          |
| <b>Веб-застосунок для моніторингу<br/>погодних та екологічних показників на<br/>мові Javascript (фреймворк Vue.js)</b> | <b>01</b> АКТУАЛЬНІСТЬ РОБОТИ                            |
|                                                                                                                        | <b>02</b> ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ                  |
|                                                                                                                        | <b>03</b> МЕТА РОБОТИ                                    |
|                                                                                                                        | <b>04</b> АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ                         |
|                                                                                                                        | <b>05</b> ОБГРУНТУВАННЯ ВИБОРУ<br>ТЕХНОЛОГІЧНОГО СТЕКУ   |
|                                                                                                                        | <b>06</b> АРХІТЕКТУРА ТА ЛОПІКА ВЗАЄМОДІЇ<br>КОМПОНЕНТІВ |
|                                                                                                                        | <b>07</b> РЕАЛІЗАЦІЯ ШІН-МОДУЛЯ ТА ПРОМПТ-<br>ІНЖИНІРИНГ |
|                                                                                                                        | <b>08</b> КОРИСТУВАЦЬКИЙ ІНТЕРФЕЙС ТА<br>UX-ДИЗАЙН       |
|                                                                                                                        | <b>09</b> ДЕМОНСТРАЦІЯ РОБОТИ<br>(СКРИНШОТИ)             |
|                                                                                                                        | <b>10</b> ОПТИМІЗАЦІЯ ТА ТЕСТУВАННЯ<br>РЕЗУЛЬТАТІВ.      |
|                                                                                                                        | <b>11</b> ВИСНОВКИ.                                      |

Слайд 2

# АКТУАЛЬНІСТЬ

- Стрімкі кліматичні зміни та погіршення стану довкілля.
- Необхідність швидкої інтерпретації метеоданих для здоров'я та безпеки.
- Перехід від складних графіків до персоналізованих ШІ-порад.

Слайд 3

## ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

### ● ОБ'ЄКТ ДОСЛІДЖЕННЯ

Процес моніторингу та візуалізації екологічних показників у вебсередовищі.

### ● ПРЕДМЕТ ДОСЛІДЖЕННЯ

Методи та інструменти інтеграції штучного інтелекту в інтерфейси на базі Vue.js.

Слайд 4

# МЕТА РОБОТИ.



**МЕТА РОБОТИ:**  
Розробка інтелектуального вебзастосунку «EcoWatch» для надання точних метеоданих та контекстних екологічних рекомендацій.

Слайд 5

## АНАЛІЗ ІНСУЮЧИХ РІШЕНЬ І ОБГРУНТУВАННЯ РОЗРОБКИ

### 01 НЕДОЛІКИ АНАЛОГІВ.

- Складність інтерпретації метеоданих.
- Відсутність інтелектуальних рекомендацій.
- Низька адаптивність екологічних прогнозів.

### 02 ПЕРЕВАГИ EcoWatch.

- Інтеграція Gemini AI для генерації порад.
- Мінімалістичний інтерфейс (Glassmorphism).
- Фокус на екологічних індексах (AQI, UV).

| Функція                  | Аналоги (AccuWeather/Windy) | EcoWatch |
|--------------------------|-----------------------------|----------|
| Прогноз погоди           | ✓                           | ✓        |
| Екологічні індекси (AQI) | ⚠(часто платно)             | ✓        |

Слайд 6

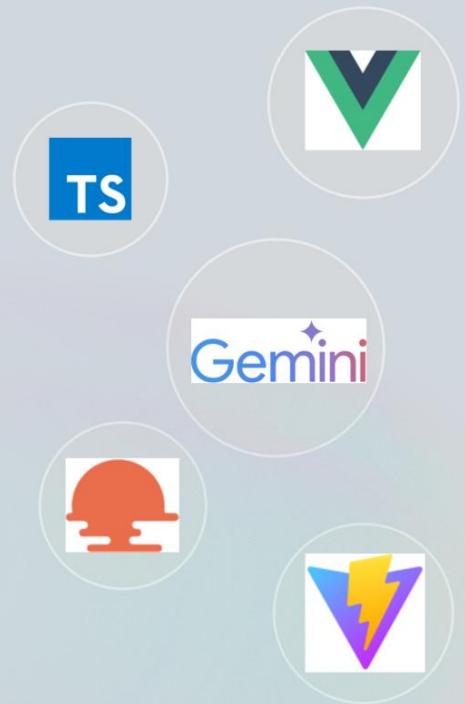
# Обґрунтування вибору технологічного стека

## Основа розробки:

- Vue 3 (Composition API): Гнучкість, модульність та реактивність інтерфейсу.
- TypeScript: Суворі типізація для мінімізації помилок при роботі з API.
- Інструментарій та збірка:
- Vite: Миттєве відображення змін (HMR) та оптимізація продуктивності.
- Axios: Стабільна обробка асинхронних запитів.

## Джерела даних та ШІ:

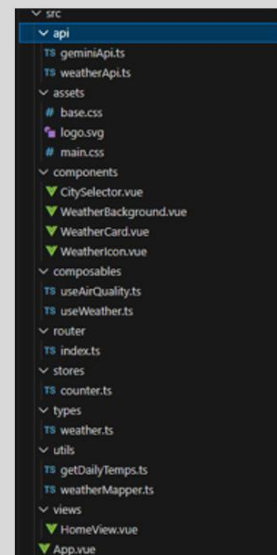
- Google Gemini 1.5 Flash: Швидка генерація контекстних порад.
- OpenWeather API: Верифіковані метеорологічні показники.



Слайд 7

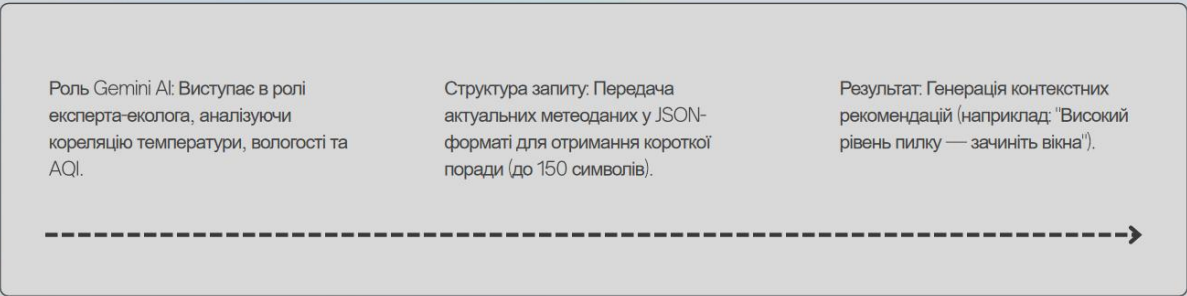
## АРХІТЕКТУРА ТА ЛОГІКА ВЗАЄМОДІЇ КОМПОНЕНТІВ

- src/api: Ізольовані методи для зовнішніх запитів.
- src/components: Перевикористовувані компоненти інтерфейсу.
- src/utills: Утиліти для мапінгу та форматування даних.
- src/types: TypeScript-інтерфейси для типізації погодних об'єктів.



Слайд 8

# Реалізація ШІ-модуля та Промпт-інжиніринг



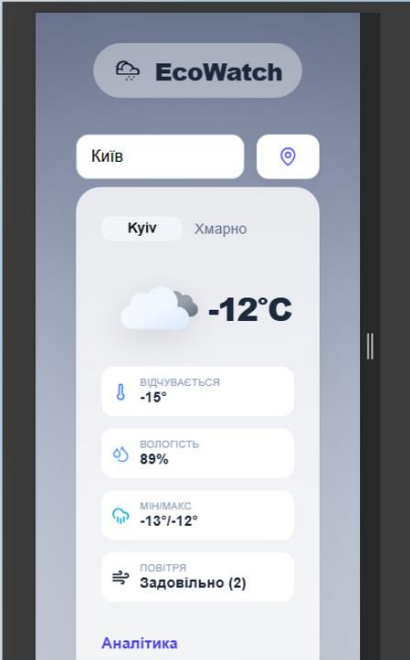
Слайд 9

# Користувацький інтерфейс та UX-дизайн

Стиль: Glassmorphism (ефект матового скла) для легкості інтерфейсу.

Адаптивність: Mobile-first стратегія, коректне відображення на будь-яких пристроях.

Динамічність: Реактивна зміна фонових анімацій (частки дощу, снігу, туману) на основі кодів стану погоди.



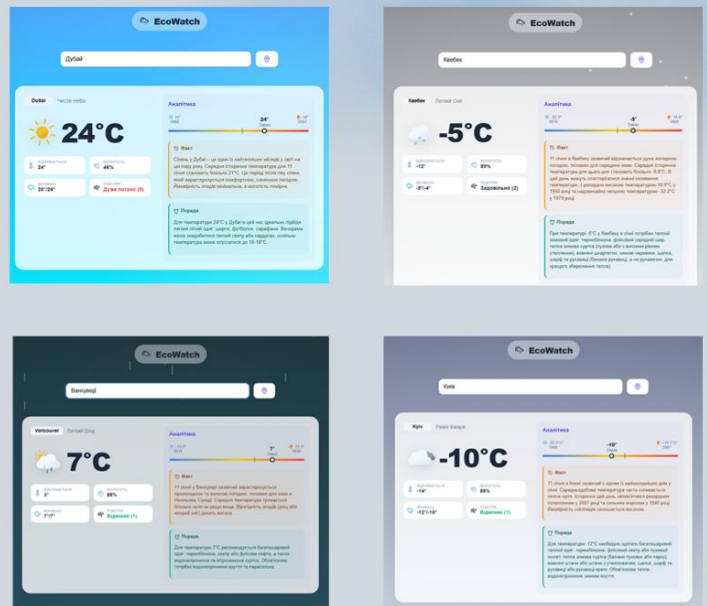
Слайд 10

## Демонстрація роботи (Скриншоти)

Основний екран: Пошук міста та картка поточної погоди.

Екологічний блок: Візуалізація індексу якості повітря (AQI).

Блок ШІ-поради: Демонстрація відповіді від моделі Gemini.



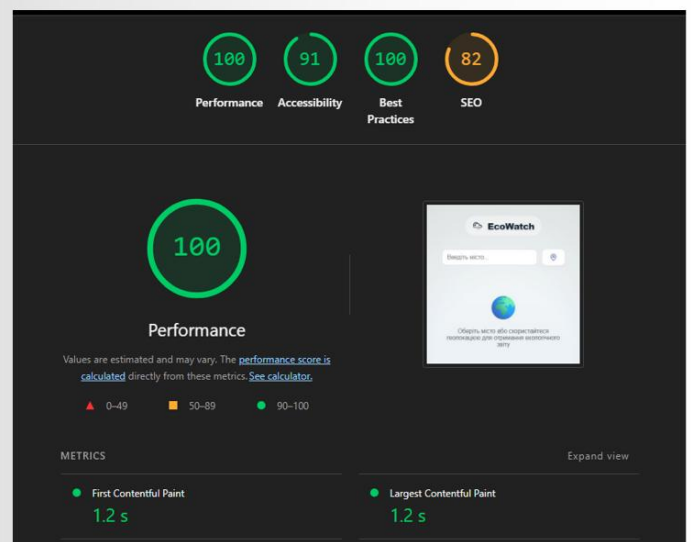
Слайд 11

## Оптимізація та тестування результатів.

Продуктивність: Показники Google Lighthouse > 90% (Performance, SEO, Best Practices).

Стабільність: Опрацювання помилок API (401, 404, 429) та станів завантаження (Loading states).

Оптимізація: Використання Debounce для пошукових запитів.



Слайд 12

# Висновки.

- Розроблено функціональний вебзастосунок з інтегрованим ШІ.
- Доведено ефективність використання Vue 3 для створення високопродуктивних метеорологічних сервісів.
- Підтверджено гіпотезу про спрощення сприйняття екологічних даних за допомогою великих мовних моделей.
- Мета роботи досягнута, завдання виконані повністю.