**Пояснювальна записка  
до курсової роботи**

на тему: Веб-сервіс прогнозування погоди, шляхом агрегації даних онлайн гідрометцентрів

КПІ.ІП-3104.045440.02.81

Київ – 2024

**ЗМІСТ**

[**ВСТУП** 4](#_Toc198228486)

[1 ПЕРЕДПРОЄКТНЕ ОБСТЕЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ 5](#_Toc198228487)

[1.1 Аналіз предметної області 5](#_Toc198228488)

[1.2 Аналіз існуючих рішень 7](#_Toc198228489)

[1.2.1 Аналіз відомих програмних продуктів 7](#_Toc198228490)

[1.2.2 Аналіз відомих алгоритмічних та технічних рішень 16](#_Toc198228491)

[1.3 Опис бізнес-процесів 17](#_Toc198228492)

[Висновки до розділу 19](#_Toc198228493)

[2 РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 20](#_Toc198228494)

[2.1 Варіанти використання програмного забезпечення 20](#_Toc198228495)

[2.2 Аналіз системних вимог 23](#_Toc198228496)

[2.3 Розроблення функціональних вимог 26](#_Toc198228497)

[2.4 Розроблення нефункціональних вимог 30](#_Toc198228498)

[2.5 Постановка задачі 32](#_Toc198228499)

[Висновки до розділу 33](#_Toc198228500)

[3 КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 34](#_Toc198228501)

[3.1 Архітектура програмного забезпечення 34](#_Toc198228502)

[3.2 Обґрунтування вибору засобів розробки 38](#_Toc198228503)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 40](#_Toc198228504)

**ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ**

API – Application Programming Interface, прикладний програмний інтерфейс.

REST – Representational State Transfer, передача репрезентованого стану

UI – User Interface, інтерфейс користувача.

BPMN – Business Process Model and Notation, нотація моделювання бізнес-процесів.

VPS – Virtual Private Server, віртуальний приватний сервер

Frontend – частина програми, з якою взаємодіє користувач.

Backend – серверна частина програми, що відповідає за обробку даних і логіку

**ВСТУП**

У сучасному світі прогнозування погоди стало дуже важливим для загальної життєдіяльності суспільства, оскільки від нього залежить безліч аспектів життя: від планування аграрних робіт до організації масових заходів. Актуальність даної роботи обумовлена необхідністю забезпечення легкого доступу до найбільш актуальних метеорологічних даних, які можуть бути отримані шляхом агрегації інформації.

На сьогоднішній день використання онлайн веб-сервісів стало дуже популярним і сервіси прогнозування погоди не є виключенням [5]. Країни по всьому світу активно впроваджують новітні технології для поліпшення точності прогнозів. Відомі наукові установи та дослідницькі організації активно розробляють нові установи та методи обробки метеорологічних даних, що дозволяє покращити не лише точність, але й швидкість отримання актуальних прогнозів.

На даний момент існує досить велика кількість платформ, які надають метеорологічну інформацію, проте взагалі не всі з них забезпечують інтеграцію даних з кількох джерел, що може призвести до менш точної інформації. У зв'язку з цим, розробка веб-сервісу, що агрегує дані з різних онлайн-центрів, є актуальною задачею.

Розроблений веб-сервіс буде мати широкий спектр застосувань, охоплюючи практично всі сфери життя людини. Наприклад, у сільському господарстві точні прогнози погоди допомагають ефективно планувати посіви та збори врожаю, зменшуючи ризики втрат. У туризмі сервіс дозволяє туристам обирати оптимальні дати для поїздок, враховуючи погодні умови, що підвищує комфорт і безпеку. У повсякденному житті актуальна інформація про погоду допомагає людям планувати проведення свого дня та організовувати різні заходи. Отже, надійні метеорологічні дані сприяють прийняттю обґрунтованих рішень та підвищують ефективність планування в різних аспектах людської діяльності.

1. ПЕРЕДПРОЄКТНЕ ОБСТЕЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ
   1. Аналіз предметної області

У сучасному світі веб-застосунки стали дуже популярними, оскільки вони забезпечують зручний доступ до різноманітних сервісів. Веб-застосунок [1] визначається як програмне забезпечення, яке працює на веб-сервері і доступне для користувачів через браузер. Це дозволяє користуватися сервісом без необхідності його встановлення на локальний пристрій. Веб-застосунки охоплюють широкий спектр функцій, від простих форм для заповнення до складних систем для управління великими об’ємами даних, що робить їх важливими інструментами у різних сферах, включаючи бізнес, освіту та науку.

Прогнозування погоди, як предметна область, є складною і багатогранною [6,7]. Вона вимагає обробки та аналізу великих обсягів метеорологічних даних, зібраних з різних джерел, таких як метеостанції, супутники та радіолокаційні системи. Сучасні IT-технології пропонують різноманітні способи обробки даних, включаючи використання штучного інтелекту та машинного навчання для покращення точності прогнозів.

На поточному етапі розвитку технологій процес створення агрегаторів метеорологічних даних охоплює кілька важливих етапів [6,7]. Першим етапом є автоматизований збір даних з різноманітних джерел, таких як національні гідрометцентри, приватні метеостанції або супутникові системи. Далі слідує обробка та уніфікація отриманих даних, оскільки різні джерела можуть використовувати різні формати. Об’єднані дані піддаються аналізу за допомогою алгоритмів, що дозволяє виявляти можливі помилки та створювати прогнози на основі агрегованої інформації. Останнім етапом є візуалізація - представлення даних у вигляді інтерактивних графіків, карт або таблиць, доступних через веб-інтерфейс.

Типові можливості агрегаторів включають відображення погодних умов у реальному часі, довгострокові прогнози, аналіз історичних даних, сповіщення про зміни погоди, і налаштування під користувацькі запити [8].

Однак існує низка проблем, з якими стикаються сучасні агрегатори [6,7]. Відсутність стандартизації даних з різних джерел може призводити до розбіжностей у прогнозах. Затримки в оновленні інформації через складність інтеграції систем також знижують актуальність даних. Крім того, обмеження алгоритмів обробки великих обсягів даних можуть впливати на швидкість роботи під час пікових навантажень.

Шляхи покращення можуть включати вдосконалення алгоритмів агрегації, підвищення швидкості обробки даних за допомогою розподілених систем і використання додаткових методів валідації даних для покращення якості прогнозів [6].

У рамках цієї курсової роботи буде розроблено веб-сервіс, який об’єднуватиме дані з різних метеорологічних центрів. Основна мета - забезпечити користувачів більш повною та комплексною інформацією про погодні умови, об’єднавши різні джерела в одному інтерфейсі. На відміну від існуючих рішень, сервіс зосереджуватиметься на ефективній агрегації даних без розробки нових алгоритмів аналізу. Це дозволить мінімізувати залежність від окремих джерел та підвищити надійність і точність прогнозів.

* 1. Аналіз існуючих рішень

Проаналізуємо відомі на сьогодні програмні продукти та технічні рішення, що допоможуть у реалізації веб-сервісу прогнозування погоди шляхом агрегації даних різних гідрометцентрів.

* + 1. Аналіз відомих програмних продуктів

Спочатку проаналізуємо існуючи аналоги по агрегації погоди.

Першим прикладом відомих аналогів є Weather [2]. Цей сервіс надає детальну інформацію про погоду, включаючи прогнози на кілька днів наперед та можливість перегляду мап погоди. Інтерфейс сайту (рисунки 1.1 – 1.3) досить зручний і зрозумілий, що дозволяє легко отримувати необхідну інформацію.

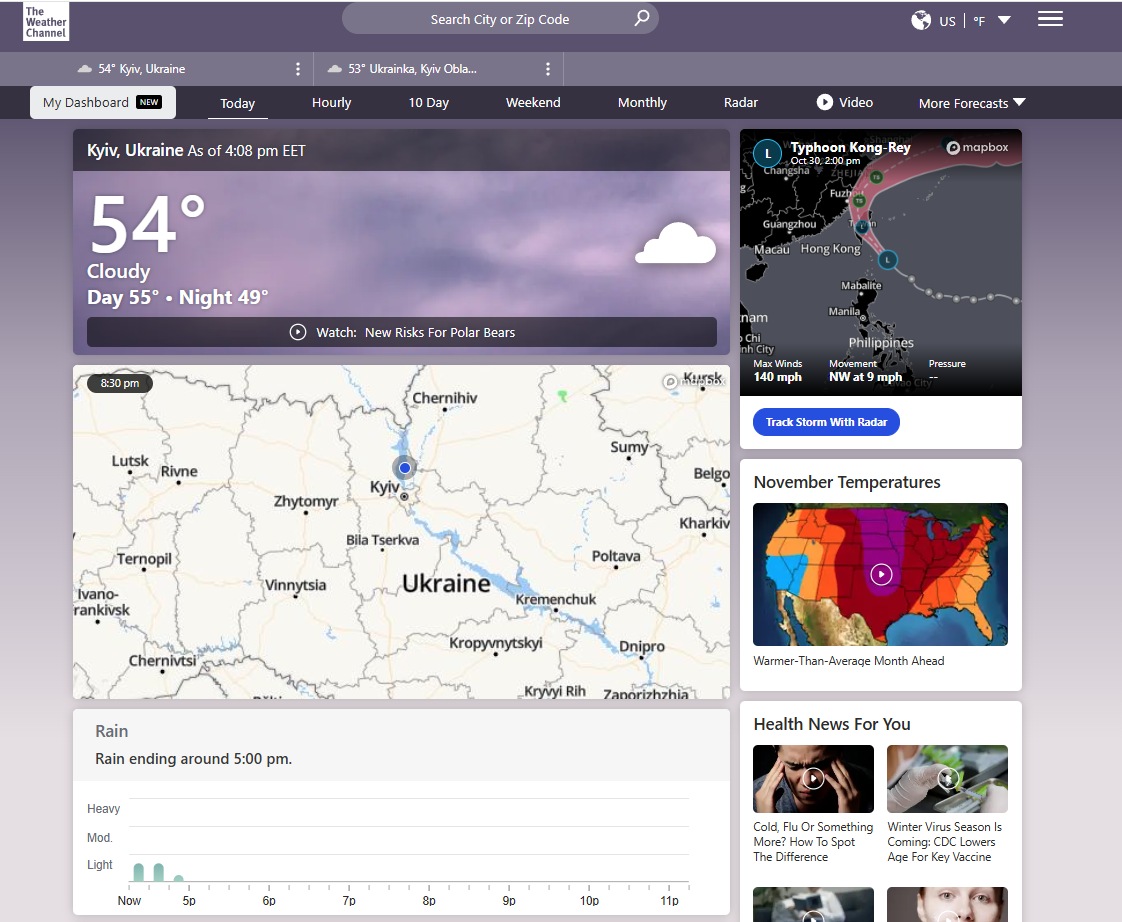


Рисунок 1.1 – UI сайту Weather

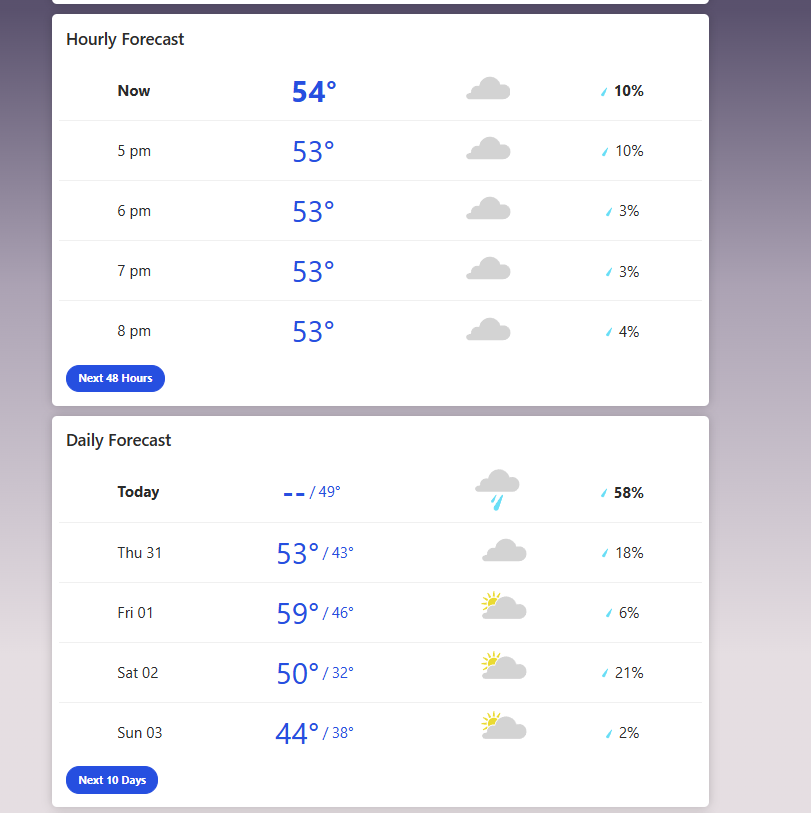


Рисунок 1.2 – Продовження UI сайту Weather

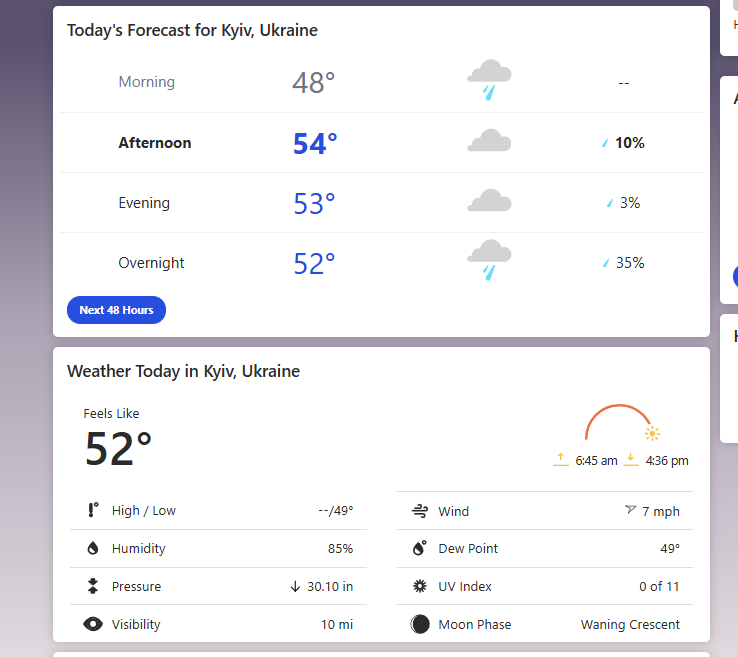


Рисунок 1.3 – Продовження UI сайту Weather

Другим продуктом є Accuweather [3]. Цей сервіс також є дуже зручним у використанні (рисунки 1.4-1.6) та надає детальні прогнози погоди, включаючи спеціалізовані прогнози для різних сфер життєдіяльності, таких як, господарство, туризм та багато іншого.

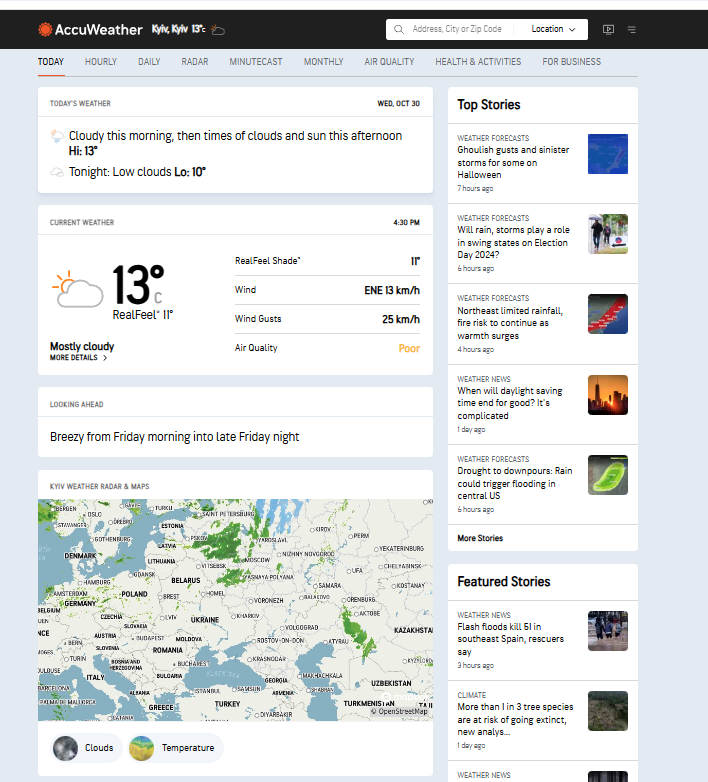


Рисунок 1.4 - UI сайту Accuweather

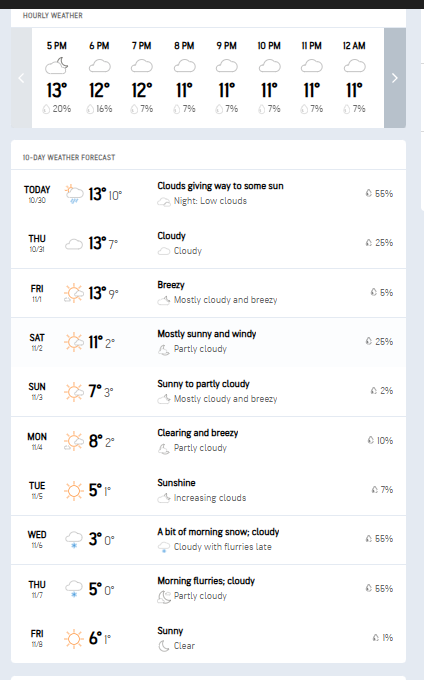


Рисунок 1.5 – Продовження UI сайту Accuweather

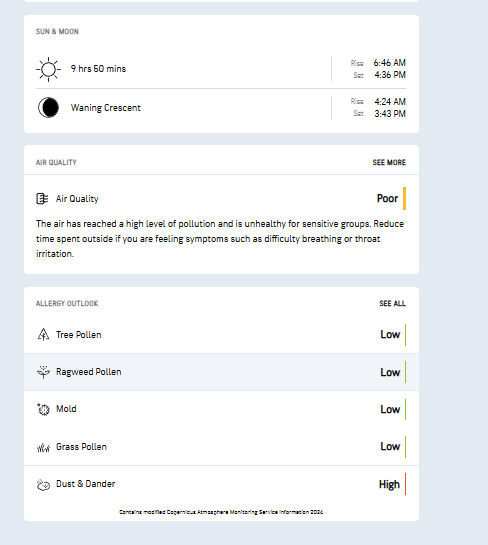


Рисунок 1.6 – Продовження UI сайту Accuweather

Третім сервісом є Worldweather [4], який є офіційним ресурсом Всесвітньої метеорологічної організації. Він акцентує увагу на актуальних погодних умовах та тенденціях. Інтерфейс користувача також досить зрозумілий (рисунок 1.7).

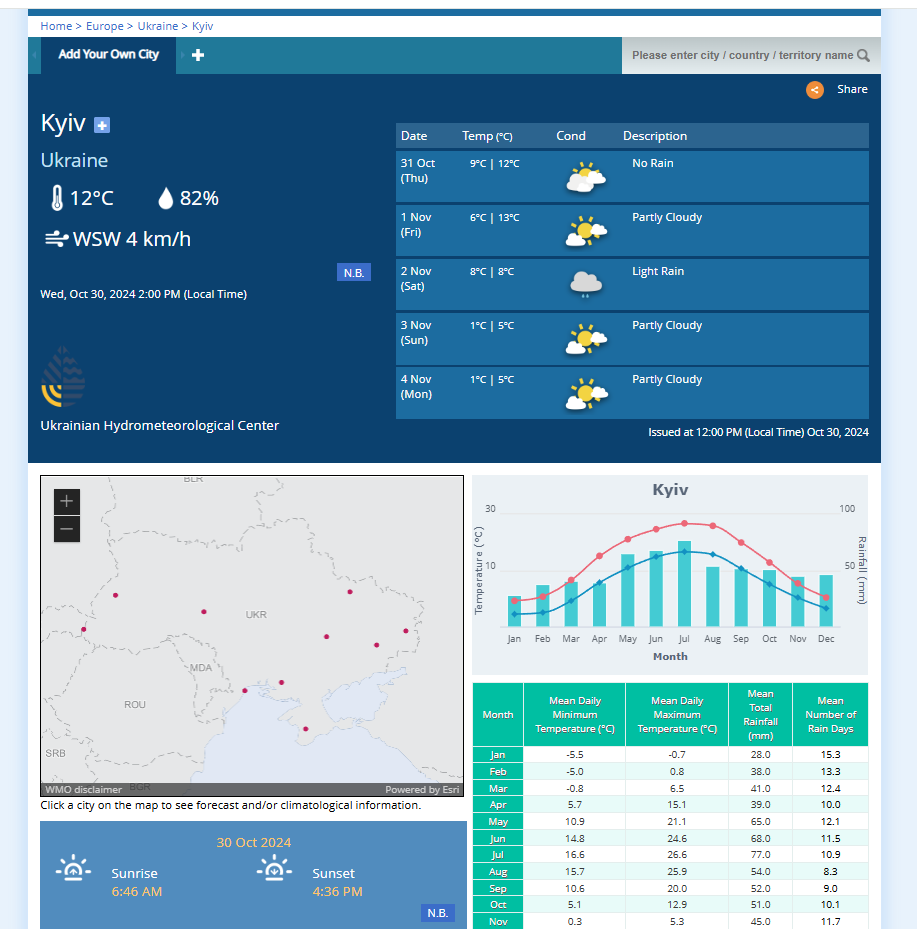


Рисунок 1.7 - UI сайту Worldweather

Більш узагальнене порівнняня основних функцій сервісів наведено в таблиці 1.1.

Таблиця 1.1 – Порівняння сервісів

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Функціонал | Проєкт в рамках курсової роботи «Веб-сервіс прогнозування погоди, шляхом агрегації даних онлайн гідрометцентрів» | Weather | Accuweather | Worldweather | Пояснення |
| Агрегація даних різних центрів | Так | Ні | Ні | Ні | Проєкт в рамках курсової роботи об’єднує дані з різних джерел, що забезпечує ширший прогноз. |

Продовження таблиці 1.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Функціонал | Проєкт в рамках курсової роботи «Веб-сервіс прогнозування погоди, шляхом агрегації даних онлайн гідрометцентрів» | Weather | Accuweather | Worldweather | Пояснення |
| Поточна погода | Так | Так | Так | Так | Всі сервіси надають поточну погоду. |
| Прогноз погоди на кілька днів | Так | Так | Так | Так | Всі сервіси надають прогноз на кілька днів. |

Продовження таблиці 1.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Функціонал | Проєкт в рамках курсової роботи «Веб-сервіс прогнозування погоди, шляхом агрегації даних онлайн гідрометцентрів» | Weather | Accuweather | Worldweather | Пояснення |
| Прогноз погоди по годинам на кілька днів | Так | Так | Так | Ні | Один із сервісів не надає прогноз по годиннам на кілька днів. |

* + 1. Аналіз відомих алгоритмічних та технічних рішень

У процесі розробки веб-сервісу необхідно розглянути різні архітектурні підходи та типи хостингу.

Серед архітектурних моделей [9] було проаналізовано монолітну, мікросервісну, сервіс-орієнтовану та багаторівневу. Монолітна архітектура передбачає розміщення всієї логіки застосунку в одному кодовому блоку, що спрощує розгортання, але ускладнює масштабування та підтримку. Мікросервісна архітектура розділяє застосунок на незалежні сервіси, кожен з яких відповідає за окрему функціональність, що полегшує масштабування та обслуговування, але вимагає складнішої інфраструктури. Сервіс-орієнтована архітектура фокусується на наданні функціональності через сервіси з чітко визначеними інтерфейсами, що дозволяє повторне використання та інтеграцію. Багаторівнева архітектура розділяє застосунок на рівні, такі як презентаційний, логіки застосунку та доступу до даних, що сприяє модульності та гнучкості.

Для реалізації веб-сервісу з агрегації прогнозів погоди було обрано мікросервісну архітектуру з поділом на клієнт-сервер. Це означає, що система має чіткий поділ на клієнтську частину та серверну частину, які взаємодіють через мережу за допомогою HTTP(S)-запитів. У той же час, серверна частина складається з декількох незалежних компонентів.

Щодо вибору хостингу, було розглянуто різні варіанти [10], включаючи віртуальний хостинг, VPS, хмарний хостинг та виділений сервер. Спільний хостинг є економічним варіантом, але обмежує контроль над серверним середовищем. Хмарний хостинг забезпечує високу масштабованість та доступність, але може бути дорожчим. Виділений сервер надає повний контроль, але вимагає значних ресурсів для обслуговування. Зрештою, було обрано VPS-хостинг, який забезпечує баланс між контролем, продуктивністю та вартістю. VPS дозволяє мати власне ізольоване середовище з можливістю налаштування системи, що відповідає потребам проєкту.

Таким чином, обрана архітектура та тип хостингу забезпечують ефективну реалізацію веб-сервісу з агрегації прогнозу погоди, дозволяючи досягти високої продуктивності, масштабованості та зручності в обслуговуванні.

* 1. Опис бізнес-процесів

Для опису основного бізнес-процесу отримання прогнозу погоди використовується BPMN [39] модель (рисунок 1.8).

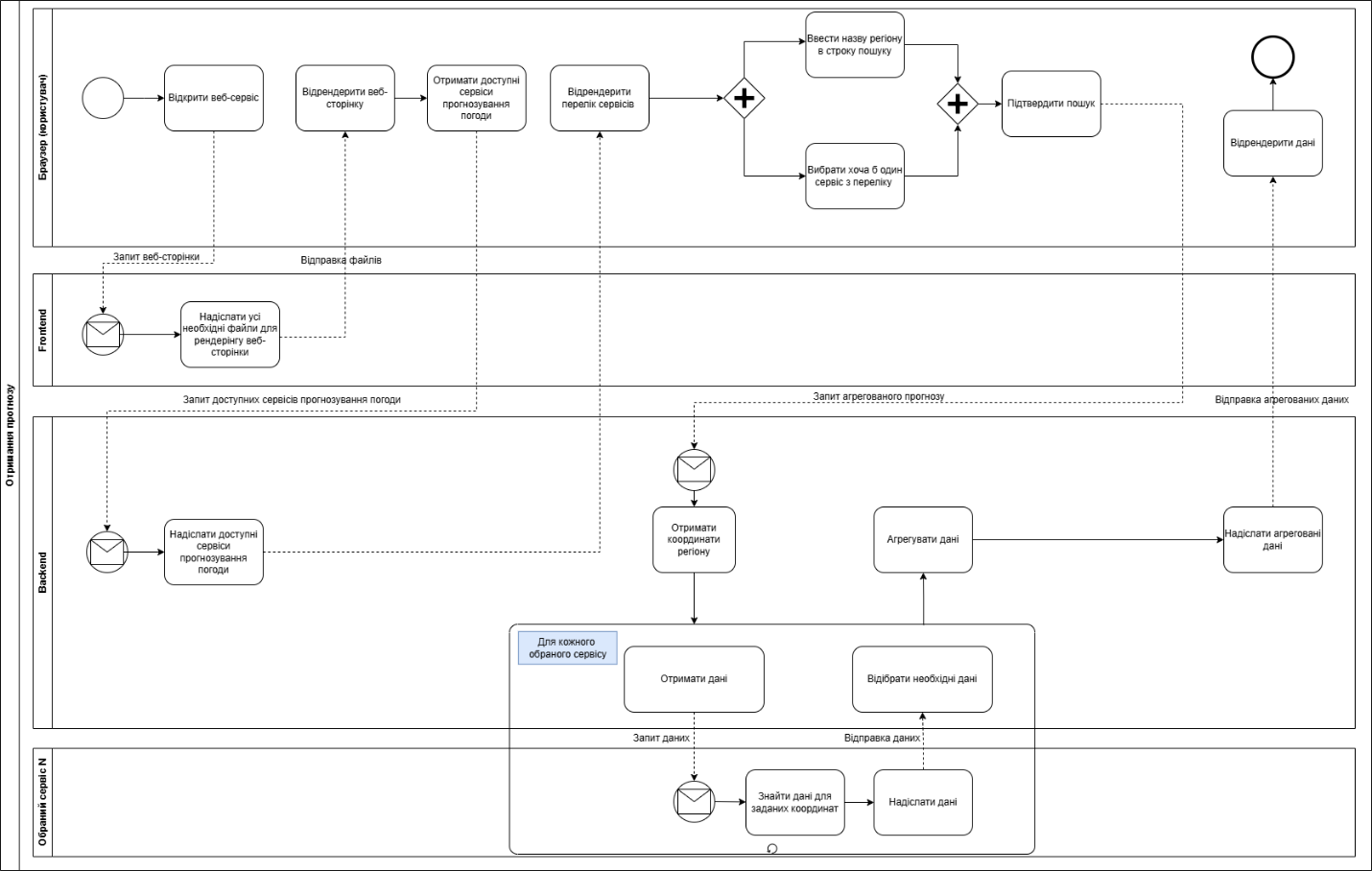


Рисунок 1.8 – Схема бізнес-прцесу отримання прогнозу

Опис послідовності отримання прогнозу:

* користувач відкриває веб-сторінку сервісу (браузер робить запит до frontend-частини);
* веб-сторінка сервісу відображається на екрані користувача (браузер отримує файли від frontend-частини і рендерить їх);
* користувач отримує перелік доступних сервісів прогнозування погоди (браузер робить запит до backend-частини та рендерить відповідь);
* користувач вводить необхідне місто, обирає певні сервіси та підтверджує запит (браузер робить запит до backend-частини);
* backend-частина отримує назву міста, конвертує її в координати;
* backend-частина для кожного обраного сервісу робить запит до відповідного API, отримує дані та фільтрує тільки необхідну їй інформацію;
* backend-частина агрегує оброблені дані та відправляє користувачу (браузеру);
* прогноз відображається на екрані користувача (браузер рендерить отримані агреговані дані).

Висновки до розділу

У даному розділі було проведено детальне передпроєктне дослідження предметної області, що стало основою для створення веб-сервісу з агрегації погодних даних.

Спершу було виконано аналіз предметної області, де було описано актуальність використання веб-застосунків. Було обґрунтовано необхідність у створенні додатку інтеграції різних сервісів прогнозування даних, що забезпечить більш точний і ширший прогноз погоди.

Наступним кроком був аналіз існуючих рішень, що включав огляд відомих сервісів, таких як Weather, Accuweather, Worldweather. Проаналізовано їхні можливості та недоліки з точки зору агрегації даних і представлено їх у таблиці. Це дало змогу визначити основні відмінності між сервісами та переконатися в унікальності запропонованого рішення.

Далі, для забезпечення стабільної роботи веб-сервісу, було обрано мікросервісну архітектуру із поділом на клієнтську та серверну частину, задля забезпечення масштабованості та контролю окремих сервісів. Як хостинг було обрано VPS, що надає баланс між контролем, продуктивністю та вартістю.

Крім того, у наступному розділі було описано бізнес-процесисервісу за допомогою BPMN-моделі, що ілюструє повний цикл взаємодії користувача з додатком – від відкриття сторінки до відображення прогнозу. Ця модель показує логіку обробки запиту користувача, вибір джерел, агрегацію та виведення результатів

Отже, проведене передпроєктне обстеження дозволило сформувати чітке уявлення про предметну область, існуючі аналоги, технічні рішення та бізнес-процеси, що стануть основою для подальшого формулювання вимог до програмного забезпечення.

1. РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
   1. Варіанти використання програмного забезпечення

Для опису системи прогнозування погоди, шляхом агрегації даних різних гідрометцентрів, було побудовано (рисунок 2.1) та описано (таблиця 2.1) діаграму варіантів використання в UML-нотації.

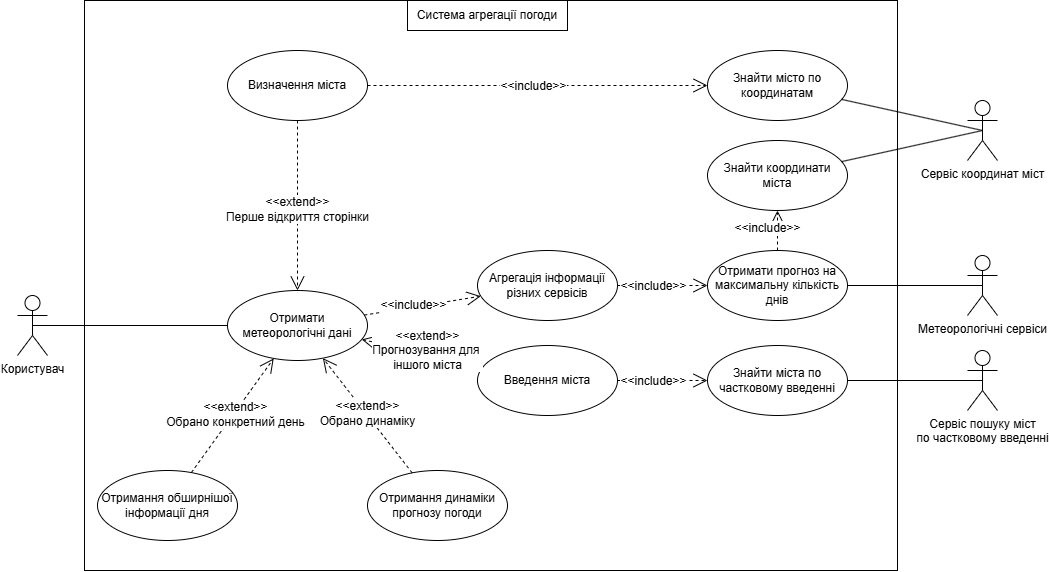


Рисунок 2.1 – Діаграма варіантів використання системи агрегації погоди

Таблиця 2.1 – Опис основного варіанту використання

|  |  |
| --- | --- |
| Use case name | Отримати метеорологічні дані |
| Use case id | UC-00 |
| Goals | * отримання поточної або прогнозованої погоди, агрегуючи дані різних сервісів; * отримання як загальної короткої поденної інформації, так і більш обширної для конкретного дня; * передбачення можливості перегляду динаміки прогнозу погоди. |

Продовження таблиці 2.1

|  |  |
| --- | --- |
| Actors | Користувач, зовнішні метеорелогічні сервіси, сервіс координат міст та сервіс пошуку міст по частковому введенні |
| Trigger | Користувач відкриває сторінку, підтверджуючи доступ до геолокації, або вводить місто вручну. |
| Pre-conditions | * доступ до Інтернету; * веб-сервіс запущено. |
| Flow of events | * користувач відкриває головну сторінку веб-сервісу; * система визначає або запитує місто користувача; * система надсилає запити до зовнішніх метеорологічних сервісів; * отримані дані агрегуються та форматуються; * результати виводяться у вигляді короткої інформації днів. |

Продовження таблиці 2.1

|  |  |
| --- | --- |
| Extension | * можливість автоматичного визначення міста, якщо користувач дозволить доступ; * можливість введення міста, якщо користувач не дозволив доступ до геолокації або потребує інше місто; * можливість перегляду більш обширної інформації для конкретного дня; * можливість перегляду динаміки прогнозу у вигляді графіків. |
| Post-Condition | Користувач бачить актуальну або прогнозовану погоду у вигляді короткої інформації для кожного дня, або більш обширної для конкретного дня, або динамічних графіків. |

Для подальшого уточнення варіантів використання, також надамо унікальний ідентифікатор кожному із них (таблиця 2.2).

Таблиця 2.2 – Унікальні ідентифікатори варіантів використання

|  |  |
| --- | --- |
| Назва | Унікальний ідентифікатор |
| Введення міста | UC-01 |
| Визначення міста | UC-02 |
| Отримання більш обширної інформації дня | UC-03 |

Продовження таблиці 2.2

|  |  |
| --- | --- |
| Назва | Унікальний ідентифікатор |
| Отримання динаміки прогнозу погоди | UC-04 |
| Агрегація інформації різних сервісів | UC-05 |
| Отримати прогноз на максимальну кількість днів | UC-06 |
| Знайти координати міста | UC-07 |
| Знайти міста по частковому введенні | UC-08 |
| Знайти місто по координатам | UC-09 |

* 1. Аналіз системних вимог

З огляду на обрані технічні рішення для реалізації веб-сервісу прогнозування погоди, до серверного середовища висуваються певні мінімальні та рекомендовані вимоги. Ці вимоги обумовлені використанням фреймворку FastAPI [15], кешуючого сховища Redis [27], системи контейнеризації Docker [24], а також зворотного серверного проксі Nginx [26], що виконує роль зворотного проксі-сервера та обслуговує статичні файли frontend-частини React [19].

Враховуючи ці компоненти, обґрунтуємо системні вимоги до серверу розгортання (таблиці 2.3-2.4).

Таблиця 2.3 – Мінімільні системні вимоги серверу

|  |  |  |
| --- | --- | --- |
| Компонент системи | Мінімальні вимоги | Обґрунтування |
| Процесор | 1 vCPU | Мінімально необхідна кількість віртуальних процесорів для підтримки роботи компонентів, що мають високу швидкодію та не потребують значних ресурсів у базовій конфігурації. |
| Оперативна пам’ять | 1 ГБ | Основні компоненти системи споживають незначну кількість пам’яті в стані простою, чого достатньо для прототипу. |
| Місце на диску | 5 ГБ | Достатньо для роботи Docker, а також зберігання кешу, логів та інших даних контейнеризованих сервісів. |
| Операційна система | Ubuntu 20.04 або новіша | Легка, стабільна та поширена система з повною підтримкою Docker та зручністю адміністрування. |

Продовження таблиці 2.3

|  |  |  |
| --- | --- | --- |
| Компонент системи | Мінімальні вимоги | Обґрунтування |
| Мережеве з’єднання | 20 Мбіт/c | Забезпечує базову швидкість для передачі статичних файлів і обміну з зовнішніми сервісами. |

Таблиця 2.4 – Рекомендовані системні вимоги серверу

|  |  |  |
| --- | --- | --- |
| Компонент системи | Рекомендовані вимоги | Обрґрунтування |
| Процесор | 4 vCPU | Оптимальна кількість віртуальних процесорів для забезпечення високої продуктивності Docker і всіх сервісів у паралельній роботі. |
| Оперативна пам’ять | 4 ГБ | Додаткова пам’ять необхідна при активному використанні кешування в Redis та під час обробки запитів. |
| Місце на диску | 10 ГБ | Розширене сховище для зберігання кешу, логів та інших даних. |

Продовження таблиці 2.4

|  |  |  |
| --- | --- | --- |
| Мережеве з’єднання | 100 Мбіт/c | Забезпечує швидку передачу даних і мінімізацію затримок при зверненні до сторонніх API. |

* 1. Розроблення функціональних вимог

Програмне забезпечення розділене на модулі. Кожен модуль має свій певний набір функцій. В таблиці 2.5 наведено загальну модель вимог, а в таблиці 2.6 наведений опис функціональних вимог до програмного забезпечення. Матрицю трасування вимог можна побачити на рисунку 2.2.

Таблиця 2.5 – Загальна модель вимог

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Назва | ID вимоги | Пріоретети | Ризики |
| 1 | Збір даних з різних джерел. | FR-1 | Високий | Високий |
| 2 | Агрегація даних різних джерел. | FR-2 | Високий | Високий |
| 3 | Отримання погодних даних за назвою населеного пункту. | FR-3 | Високий | Високий |
| 4 | Перегляд узагальненого прогнозу. | FR-4 | Високий | Високий |
| 5 | Перегляд прогнозу для конкретного дня. | FR-5 | Високий | Високий |

Продовження таблиці 2.5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Назва | ID вимоги | Пріоретети | Ризики |
| 6 | Візуалізація динаміки прогнозу. | FR-6 | Середній | Середній |
| 7 | Можливість вибору періоду візуалізації динаміки. | FR-7 | Середній | Середній |
| 8 | Можливість отримання підказок при введенні міста. | FR-8 | Середній | Середній |
| 9 | Автоматичне визначення місцезнаходження. | FR-9 | Низький | Низький |

Таблиця 2.6 – Перелік функціональних вимог

|  |  |
| --- | --- |
| Назва | Опис |
| FR-1 | Збір даних з різних джерел.  Автоматизувати збір даних із кожного використаного метеорологічних сервісів у певному форматі. |
| FR-2 | Агрегація даних різних джерел.  Автоматизувати збір, агрегацію та форматування результатів прогнозу із різних джерел. |

Продовження таблиці 2.6

|  |  |
| --- | --- |
| Назва | Опис |
| FR-3 | Отримання погодних даних за назвою населеного пункту.  Дозволити користувачу введення міста та обрання сервісів, використаних для агрегації. |
| FR-4 | Перегляд узагальненого прогнозу.  Можливість отримання користувачем короткої узагальненої інформації для кожного доступного дня. |
| FR-5 | Перегляд прогнозу для конкретного дня.  Можливість отримання користувачем обширного прогнозу для конкретного обраного дня. |
| FR-6 | Візуалізація динаміки прогнозу.  Можливість перегляду графіків динаміки прогнозу за різними метеорологічними параметрами. |
| FR-7 | Можливість вибору періоду візуалізації динаміки.  Можливість отримання графіків не для усіх прогнозовних днів, а для конкретного обраного періоду. |

Продовження таблиці 2.6

|  |  |
| --- | --- |
| Назва | Опис |
| FR-8 | Можливість отримання підказок при введенні міста.  Можливість відображення усіх, схожих по введеній назві, міст, якщо такі існують. |
| FR-9 | Автоматичне визначення місцезнаходження.  Можливість визначення місцезнаходження користувача та автоматичного пошуку даних при першому запуску веб-сторінки та підтвердженні доступу до геолокації. |



Рисунок 2.2 – Матриця трасування

* 1. Розроблення нефункціональних вимог

Сервіс повинен задовільняти певним нефункціональним вимогам, тобто характеристикам, які визначають не функціональність, а властивості роботи системи. Основні нефункціональні вимоги до системи агрегації погоди наведено в таблиці 2.7.

Таблиця 2.7 – Нефункціональні вимоги системи

|  |  |
| --- | --- |
| Тип | Вимоги |
| Доступність | * веб-сервіс має бути доступним не менше 99% часу; * середній час відновлення сервісу або його частини має виконуватися до 5 секунд. |
| Продуктивність | * коректна робота при навантаженні до 100 одночасних користувачів; * для швидкого Інтернет з’єдання середній час відповіді – 1 секунда, для повільного – до 10 секунд. |
| Обмеження | * система має забезпечити коректне функціонування для користувача зі швидкістю Інтернет-з’єднання не менше 8 Мбіт/c. |
| Масштабованість | * можливість ручного горизонтального мастштабування сервісів за допомогою Docker Compose; * можливість ручного надання більших ресурсів кожному сервісу за допомогою Docker Compose. |

Продовження таблиці 2.7

|  |  |
| --- | --- |
| Тип | Вимоги |
| Вимоги до зручності використання | * інтерфейс користувача має бути інтуїтивним та зрозумілим, тобто не вимагати попереднього навчання; * для доступу до будь-якого функціоналу веб-застосунку має бути достатньо не більше 3 кроків. |

* 1. Постановка задачі

Задача даної курсової роботи полягає в розробці веб-сервісу для прогнозування погоди шляхом агрегації даних з різних онлайн-сервісів. На вхід система приймає запит користувача з обраною локацією, а також список метеосервісів, які користувач бажає використати для отримання прогнозу. У результаті користувач отримує узагальнений прогноз погоди, що включає різні метеорологічі показники, такі як температура, вологость, швидкість вітру, тиску, погодні умови тощо.

Метою даної розробки є створення зручного інструменту для надання інтегрованих погодних прогнозів, який дозволяє ефективніше планувати діяльність, залежну від погодних умов. Застосування декількох джерел даних підвищує точність і повноту інформації, а їхнє комбінування дає змогу формувати більш достовірний прогноз.

Ключовими задачами розробки є забезпечення автоматичного визначення місцезнаходження користувача, введення та допомога вибору населеного пункту, надсилання запитів до зовнішніх API, агрегація та обробка даних, отриманих із різних джерел, а також відображення результатів у зрозумілому та зручному для користувача вигляді із можливістю отримання динамічних графіків. Важливою є й реалізація швидкодії, стабільності та інтуїтивно зрозумілого інтерфейсу користувача.

Висновки до розділу

У даному розділі було сформовано повний набір вимог до майбутнього веб-сервісу агрегації погодних даних, що є ключовим етапом у процесі його розробки. Спочатку було представлено варіанти використання програмного забезпечення у вигляді діаграми та опису основних сценаріїв взаємодії користувача із системою. Це дозволило структурувати функціональність додатку та визначити спосіб взаємодії.

Далі проведено аналіз системних вимог, з урахуванням обраних технологій, таких як FastAPI, React, Docker, Redis, Nginx. Було чітко окреслено мінімальні та рекомендовані параметри серверного середовища, необхідні для стабільної та ефективної роботи сервісу.

Окремо було опрацьовано функціональні вимоги, які охоплюють усі ключові можливості системи. Вимоги було систематизовано у таблицях з вказаними пріоритетами та потенційними ризиками, що дозволяє ефективніше планувати реалізацію кожного компонента. Також було створено матрицю трасування для контролю відповідності реалізації заявленим вимогам.

Далі у розділі було визначено нефункціональні вимоги до системи, що забезпечують створення надійного, швидкого та інтуїтивно зрозумілого продукту, здатного відповідати очікуванням кінцевих користувачів.

Окрему увагу було приділено постановці задачі, де було узагальнено принцип роботи веб-застосунку, його мету та коротко описано вимоги для подальшої розробки.

Таким чином, у цьому розділі було повністю сформовано технічну, функціональну та нефункціональну основу програмного забезпечення, яка стане фундаментом для подальшого проєктування архітектури, реалізації логіки та тестування веб-сервісу прогнозування погоди.

1. КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
   1. Архітектура програмного забезпечення

Архітектура програмного забезпечення побудована з урахуванням поділу на frontend та backend частини, взаємодіючі між собою у рамках клієнт-серверної моделі. Такий підхід дозволяє чітко розмежувати функціональність, що відповідає вимогам масштабованості, підтримуваності та продуктивності системи.

Frontend реалізований як односторінковий застосунок на основі бібліотеки React, що забезпечує динамічну роботу з інтерфейсом користувача. Backend реалізовано за допомогою FastAPI і відповідає за головну логіку: допомога автоматичного визначення міста, підказки при введенні міста та агрегація погодних даних із різних джерел, а також працює із кешуванням. Для зберігання проміжних результатів використовується Redis, що знижує навантаження та прискорює відповіді. Усі компоненти працюють в окремих контейнерах з використанням Docker, що спрощує тестування та розгортання. Nginx виконує роль зворотного проксі-сервера та забезпечує маршрутизацію запитів до відповідних сервісів.

Для представлення архітектури розроблені діаграми моделі C4. Високорівнева структура системи представлена на рівнях 1 та 2 (рисунки 3.1-3.2), більш детальна організація логіки на рівні 3 (рисунок 3.3), а на рівні 4 (рисунок 3.4) зображено модель коду backend-частини.

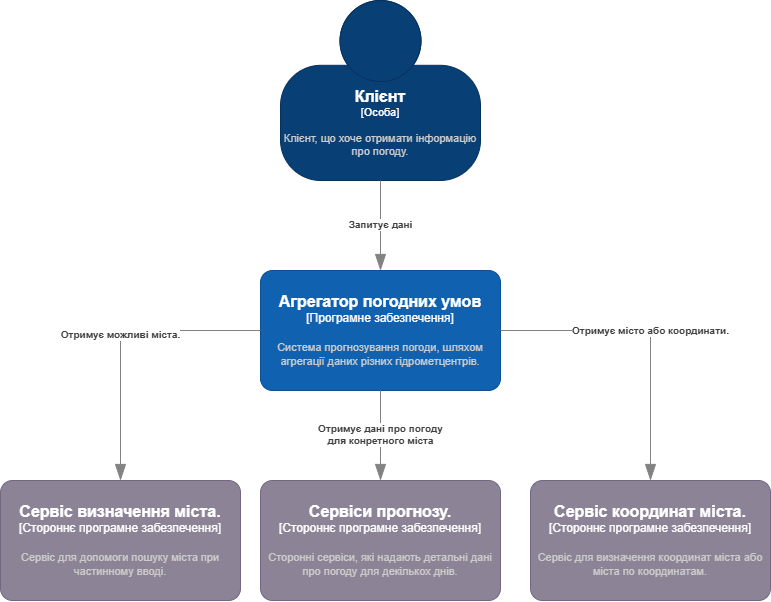


Рисунок 3.1 – Діаграма C4, рівень 1

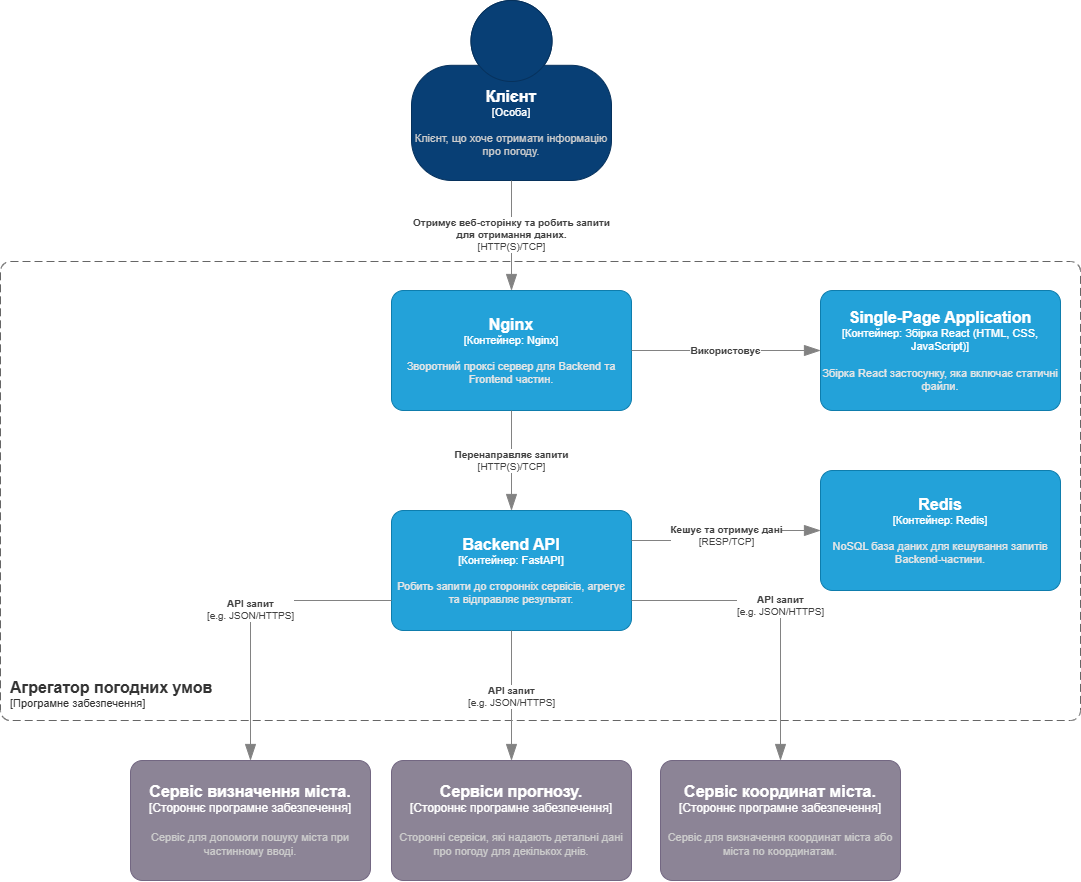


Рисунок 3.2 – Діаграма C4, рівень 2

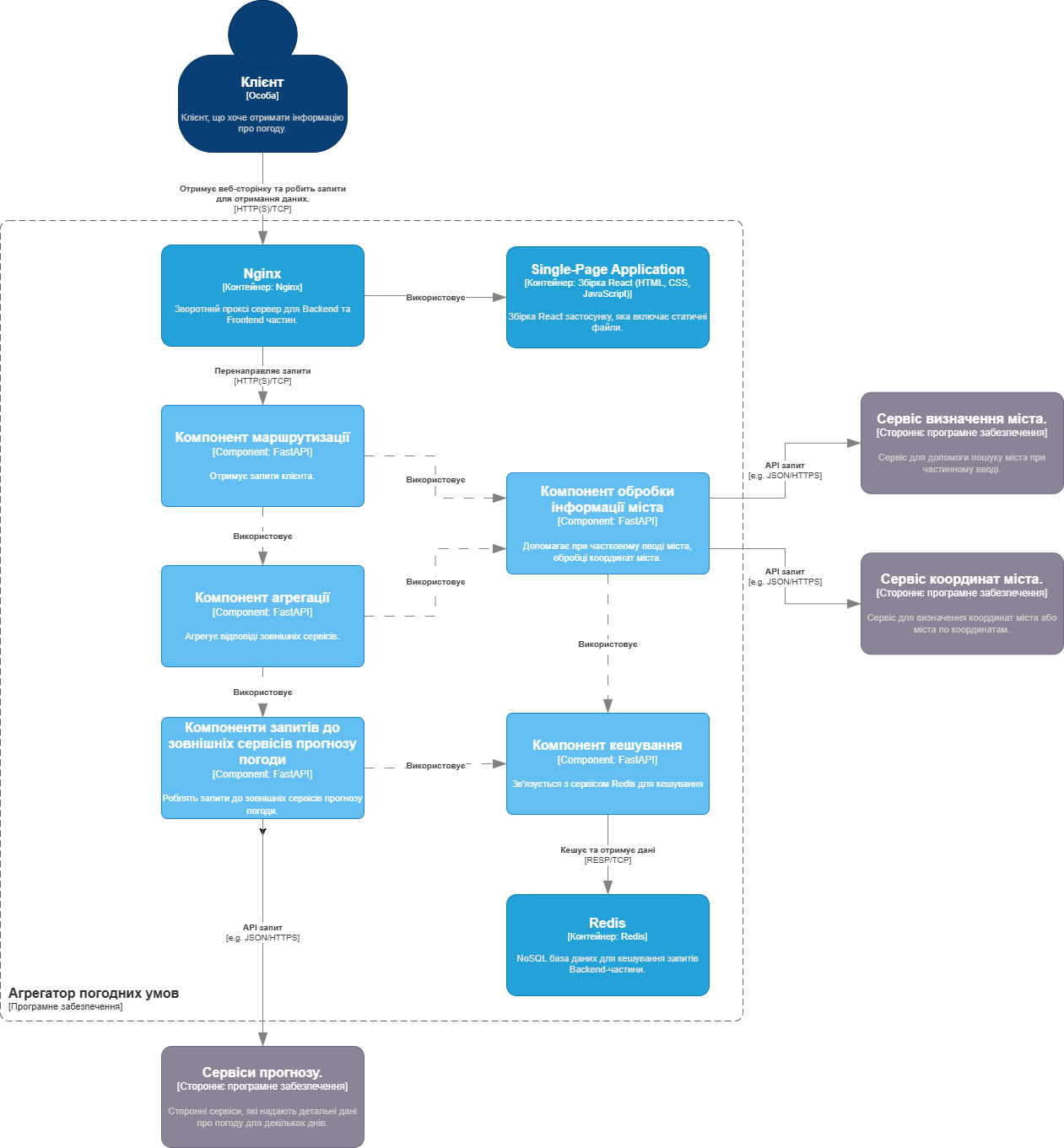


Рисунок 3.3 – Діаграма C4, рівень 3

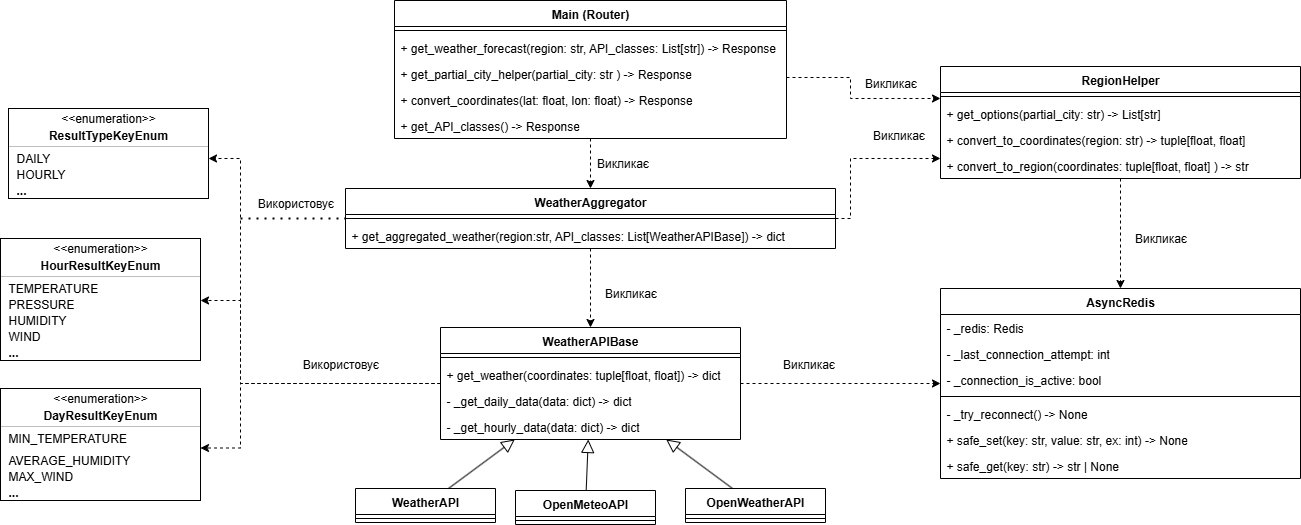


Рисунок 3.4 – Діаграма C4, рівень 4

* 1. Обґрунтування вибору засобів розробки

Для розробки веб-сервісу було обрано технології, що забезпечують високу продуктивність, зручність у розробці та масштабованість.

Основною мовою програмування для серверної частини є Python [11], оскільки він має велику різноманітність бібліотек і фреймворків для будь-яких задач. Python також підтримує асинхронне програмування, що є важливим для роботи із запитами до сторонніх сервісів. Інші варіанти, такі як Java [12], PHP [14], Go [13], хоча і є популярними для серверної частини веб-додатків, виявились не такими швидкими в плані розробки.

Для розробки веб-сервісу було обрано FastAPI [15] як основний фреймворк для Python. FastAPI дозволяє швидко створювати ефективні REST API з високою продуктивністю завдяки асинхронному підходу, що робить його оптимальним вибором для проєктів, які обробляють багато запитів одночасно. Альтернативи, такі як Flask [16] або Django [17], не забезпечують таке ж поєднання продуктивності та зручності розробки API.

Для клієнтської частини веб-сервісу був обраний JavaScript [18] з використанням бібліотеки React [19]. Це одна з найбільш популярних технологій для створення інтерактивних користувацьких інтерфейсів, що дозволяє ефективно працювати з компонентами та оновленням даних у реальному часі. Vue[20] і Angular[21] також є хорошими альтернативами, але React має більшу екосистему та кращу підтримку серед розробників, що робить його найбільш універсальним варіантом для цього проєкту. Для стилізації було обрано Tailwind [22], оскільки цей фреймворк дозволяє швидко та гнучко створювати адаптивний дизайн без потреби написання великих обсягів кастомних стилів, що є перевагою порівняно з використанням Bootstrap [23], що пропонує обмежені готові компоненти.

Для кешування даних і забезпечення високої швидкості доступу до часто використовуваної інформації був обраний Redis [27]. Він надає можливість швидкого збереження та доступу до даних у пам'яті, що значно прискорює роботу веб-сервісу. В порівнянні з такими альтернативими, як Memcached [28] або MongoDB [29], Redis є більш популярним та швидкісним варіантом.

Для керування веб-запитами, забезпечення високої доступності та безпеки був обраний Nginx [26]. Цей веб-сервер є потужним інструментом для балансування навантаження, розподілу запитів та обробки статичних файлів. Nginx забезпечує високу продуктивність при великій кількості одночасних з’єднань. Він дозволяє оптимізувати використання серверних ресурсів, оскільки працює з асинхронною обробкою запитів, що зменшує навантаження на сервер і підвищує швидкість реакції. При порівнянні із аналогами, такими як Apache [35], Nginx сильно виграє в швидкодії та легкості налаштування.

Для контейнеризації застосунку було обрано Docker [24], оскільки він дозволяє створювати ізольовані середовища для запуску різних частин сервісу, що спрощує налаштування та розгортання проєкту на різних системах. Інші рішення, такі як Kubernetes [25], є надмірними для цього проєкту, оскільки потребують складної оркестрації контейнерів, що не є необхідним для поточного рівня складності.

Для розробки веб-сервісу було обрано середовище розробки Visual Studio Code [36]. Це популярний редактор коду, що пропонує потужний інструмент, з можливістю встановлюти безліч плагінів, для підтримки розробки на мовах Python, JavaScript та відповідних бібліотек, а також інтеграцією з Docker. Інші IDE, такі як PyCharm [37] чи WebStorm [38], можуть бути потужнішими в певних аспектах, проте для даного проєкту, з використанням різних технологій, будуть не найкращим вибором.

Вибір сервісів для агрегації прогнозів погоди був обумовлений необхідністю забезпечити функціональність без значних обмежень по кількості запитів. Було обрано три погодні сервіси: Openweathermap [30], Weatherapi [31] та Open-meteo [32], які надають подібні необхідні можливості, але мають безкоштовну версію з достатьно хорошими обмеженнями в порівнянні із такими аналогами, як наприклад Tomorrow [33] або Weatherstack [34]. Також було обрано сервіс Geocodemaps, який надає безкоштовну можливість як звичайного геокодування, так і зворотнього, а також може допомагати при частковому введені міста.

* 1. Конструювання програмного забезпечення
     1. Опис сторонніх сервісів

Функціонування веб-сервісу базується на взаємодії з кількома зовнішніми API для отримання метеорологічних даних та інформації про географічне розташування.

Сервіс Geocodemaps [47] використовується для виконання операцій геокодування: перетворення назви міста на географічні координати, отримання підказок при введенні часткової назви міста та зворотнього геокодування (визначення назви міста по координатам). Це ключовий сервіс, що дозволяє уніфікувати запити до погодних API, перетворюючи назву в координати, що однаково сприймаються кожним із сервсів. Інтерфейс Geocodemaps зображено на рисунку 3.5. Обов’язковим параметром API є api\_key – унікальний ключ наданий для кожного користувача сервісу. Для зворотнього геокодування надаються також lat (широта) та lon (довгота), а в результаті отримаємо назву міста та іншу інформацію (рисунок 3.6). Для отримання можливих варіантів міст та їх координат по частковому введенні назви, необхідно передати її як параметр q. В результаті отримаємо варіанти можливих міст та координати для кожного з них (рисунок 3.7).

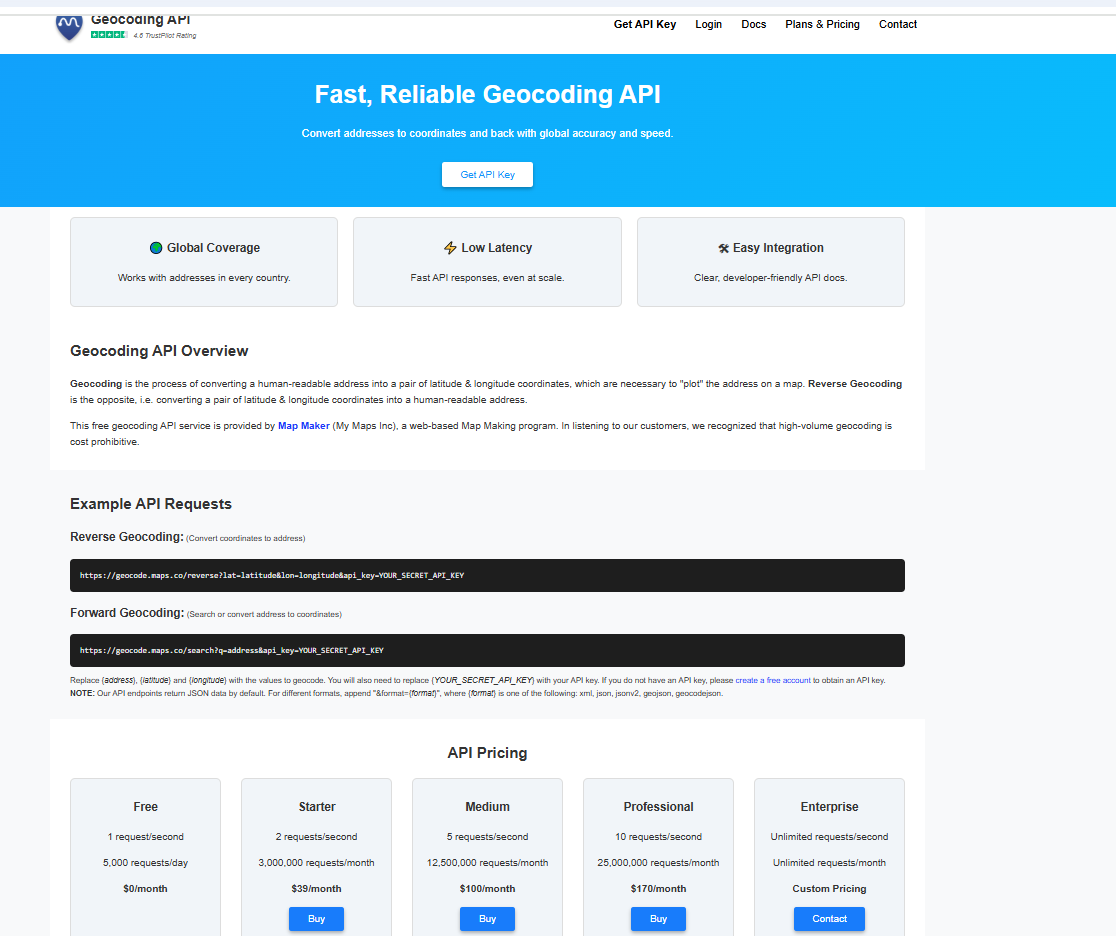


Рисунок 3.5 – Інтерфейс Geocodemaps



Рисунок 3.6 – Результат зворотнього геокодування

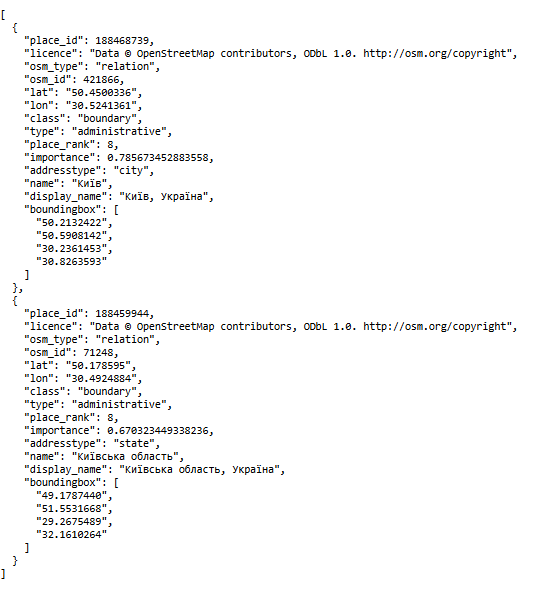


Рисунок 3.7 – Результат геокодування по частковому введенні

Наступним сервісом є Openweathermap – одним із найбільш популярних сервісів, що надає широкий спектр метеорологічних даних. У проєкті використовується його кінцева точка «One Call API 3.0», яка за одним запитом надає поточну погоду, погодинний прогноз на 48 годин та денний прогноз на 8 днів. Інтерфейс сервісу представлений на рисунку 3.8. Обов’язковими параметрами для отримання прогнозу погоди є appid, що є унікальним ключом, наданим для кожного користувача, lat (широта) та lon (довгота). Приклад результату наведено на рисунках 3.9-3.10.

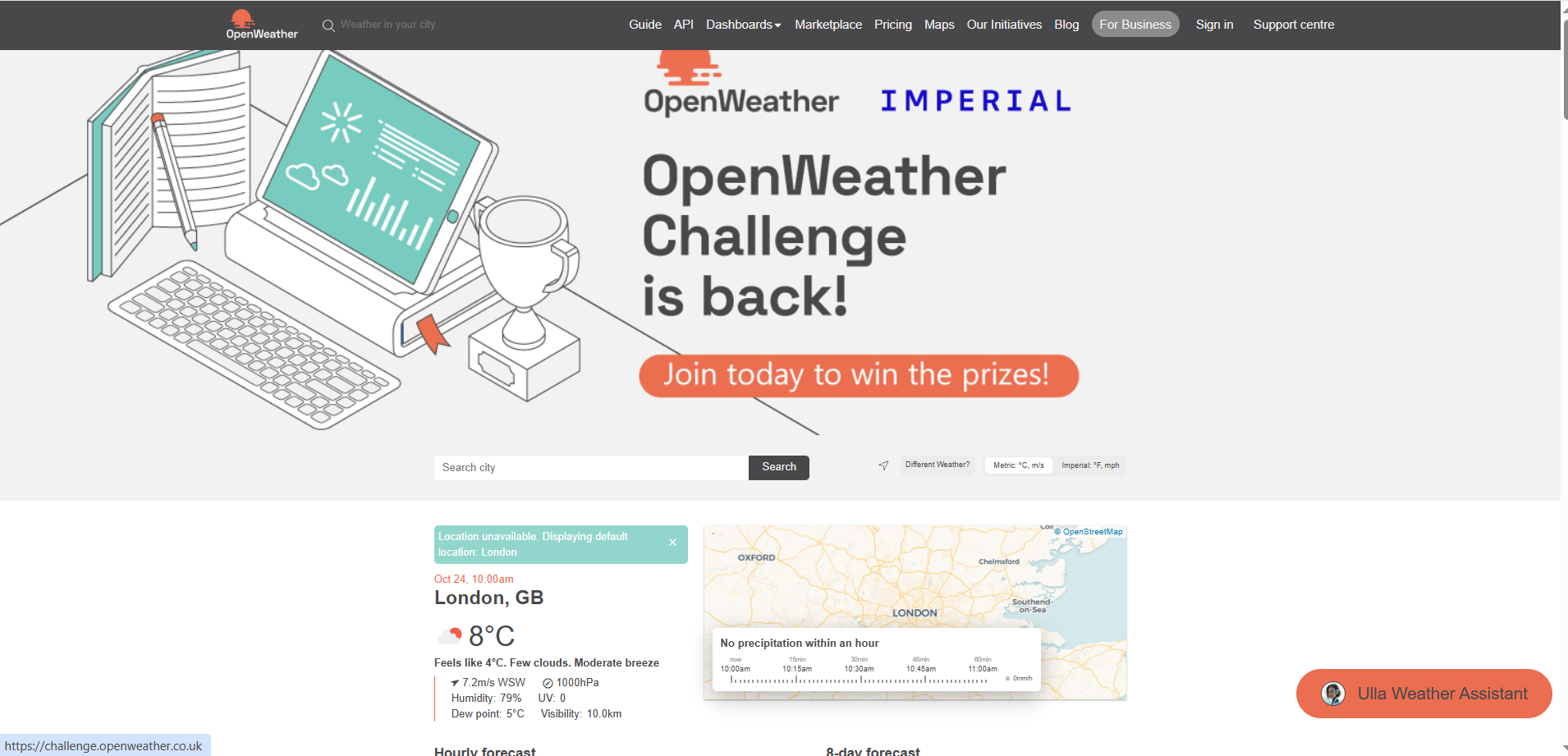


Рисунок 3.8 – Інтерфейс Openweathermap.

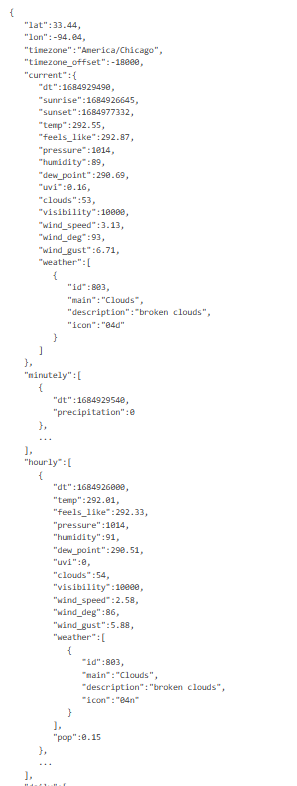


Рисунок 3.9 – Результат прогнозу Openweatheramap (частина 1)

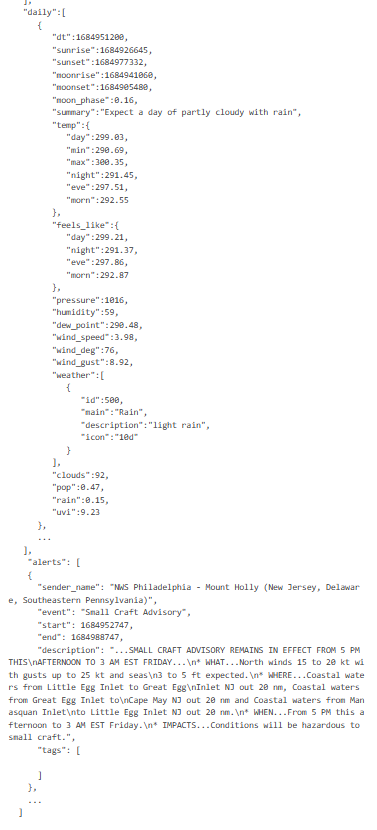


Рисунок 3.10 - Результат прогнозу Openweatheramap (частина 2)

Weatherapi – це ще один комплексний сервіс, що надає доступ до поточних погодних умов та прогнозів. У веб-застосунку використовується кінцева точка «Forecast API», яка дозволяє отримати детальний прогноз на термін до 14 днів, включаючи погодинні дані. Інтерфейс сервісу представлено на рисунку 3.11. Основними параметрами знову є унікальний ключ – key, а також q, що відповідає або за координати, або за назву. Приклад результату представлено на рисунках 3.12-3.14.

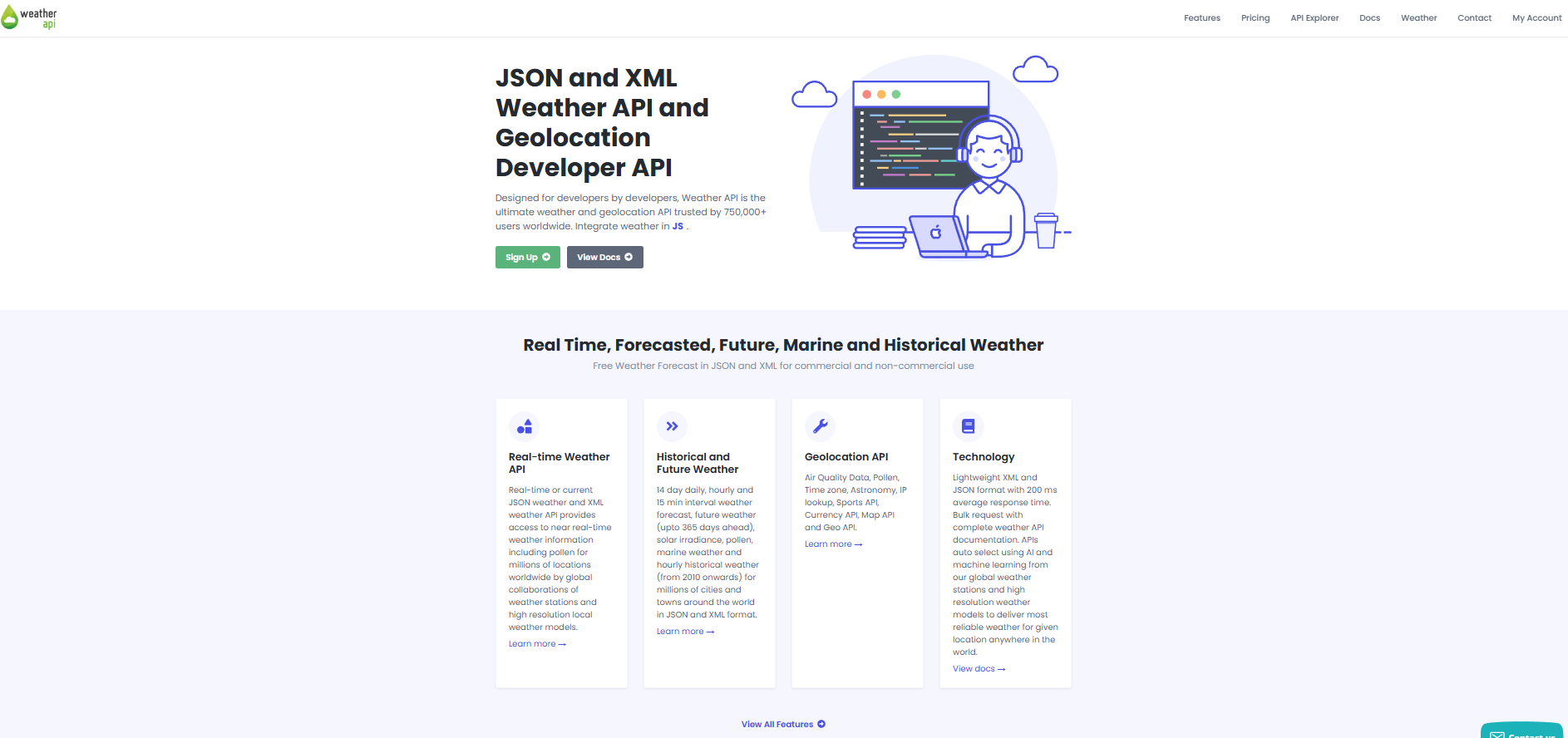


Рисунок 3.11 – Інтерфейс Weatherapi

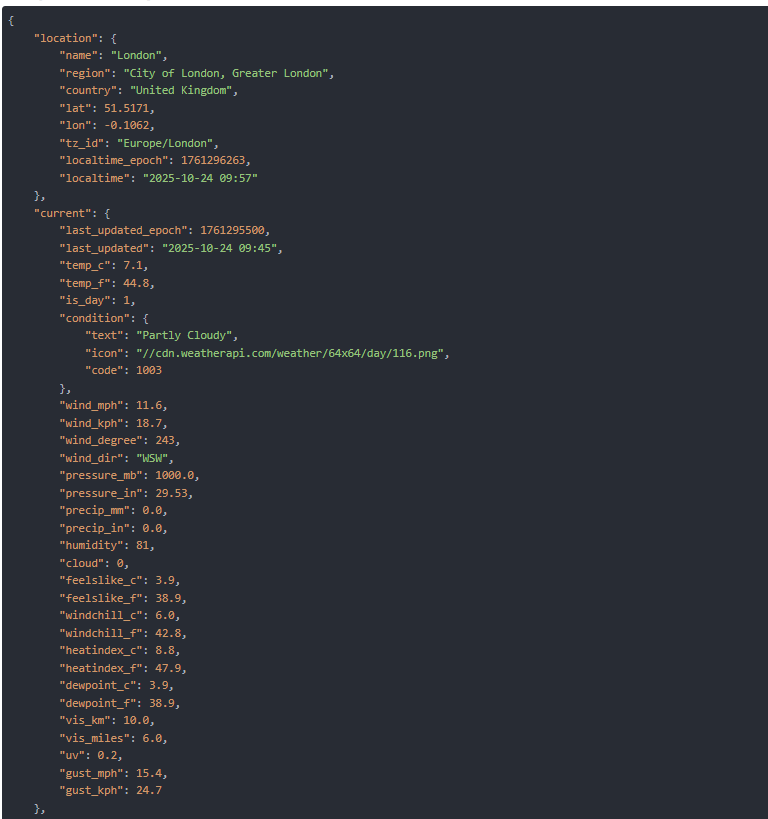


Рисунок 3.12 – Результат прогнозу Weatherapi (частина 1)

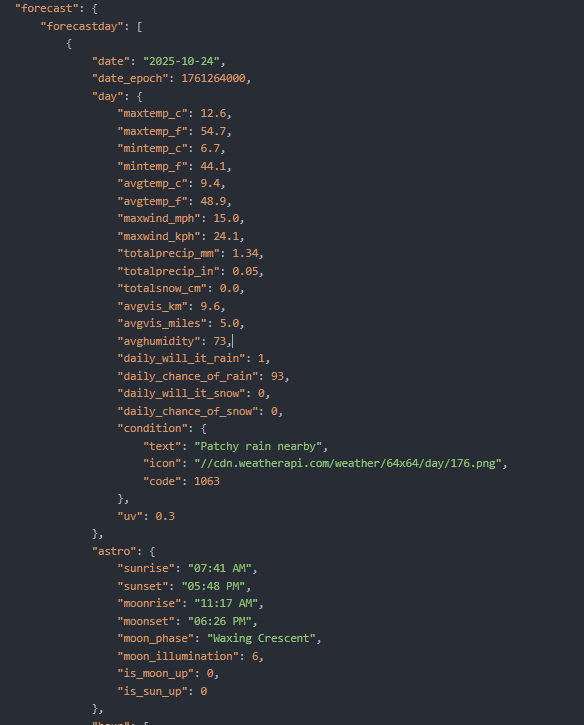


Рисунок 3.13 – Результат прогнозу Weatherapi (частина 2)

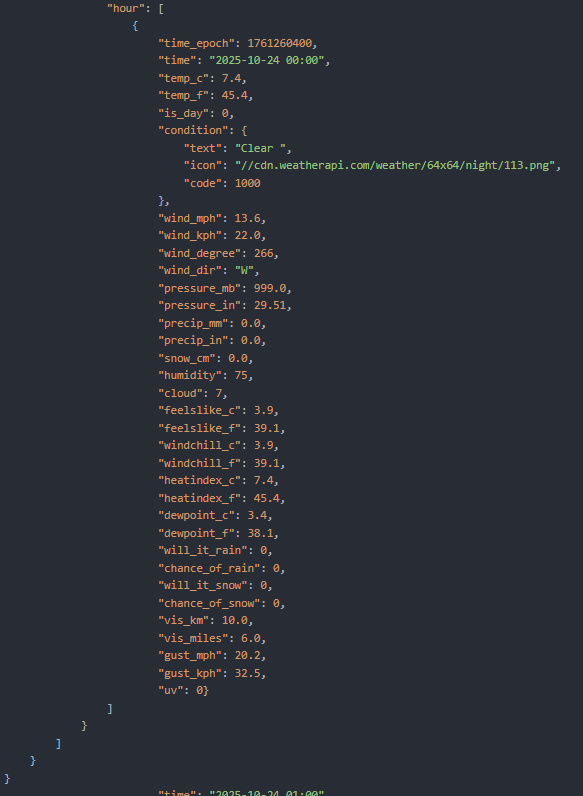


Рисунок 3.14 – Результат прогнозу Weatherapi (частина 3)

Останнім сервісом є Open-meteo – повністю безкоштовний погодний сервіс з відкритими даними, що надає високу гнучкість у налаштуванні запитів. На відміну від інших сервісів, він дозволяє користувачу самому обирати, які саме метеорологічні параметри включати у відповідь, що дозволяє оптимізувати обсяг отриманих даних. Інтерфейс сервісу представлено на рисунку 3.15. Сервіс не потребує API-ключа, тому необхідними є тільки параметри lat (широта) та lon (довгота). Приклад резульату наведено на рисунках 3.16-3.17.

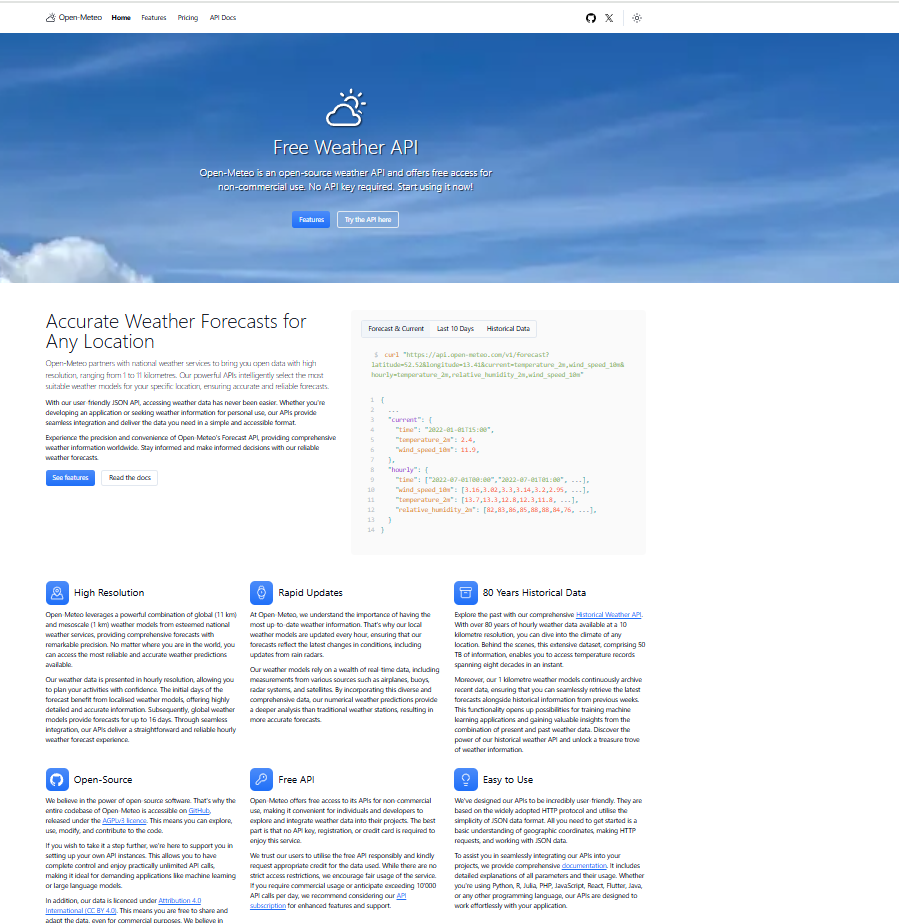


Рисунок 3.15 – Інтерфейс Open-meteo

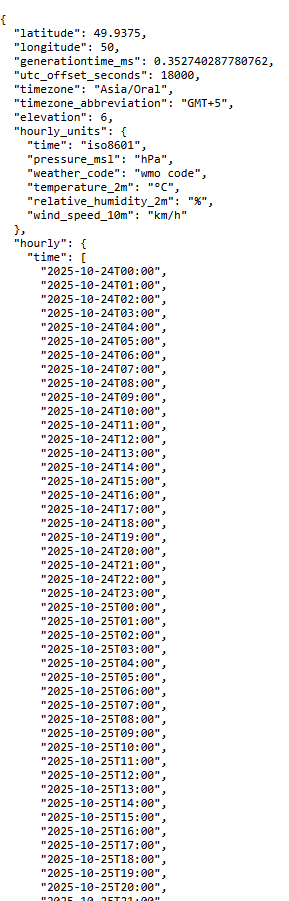


Рисунок 3.16 - Результат прогнозу Open-meteo (частина 1)

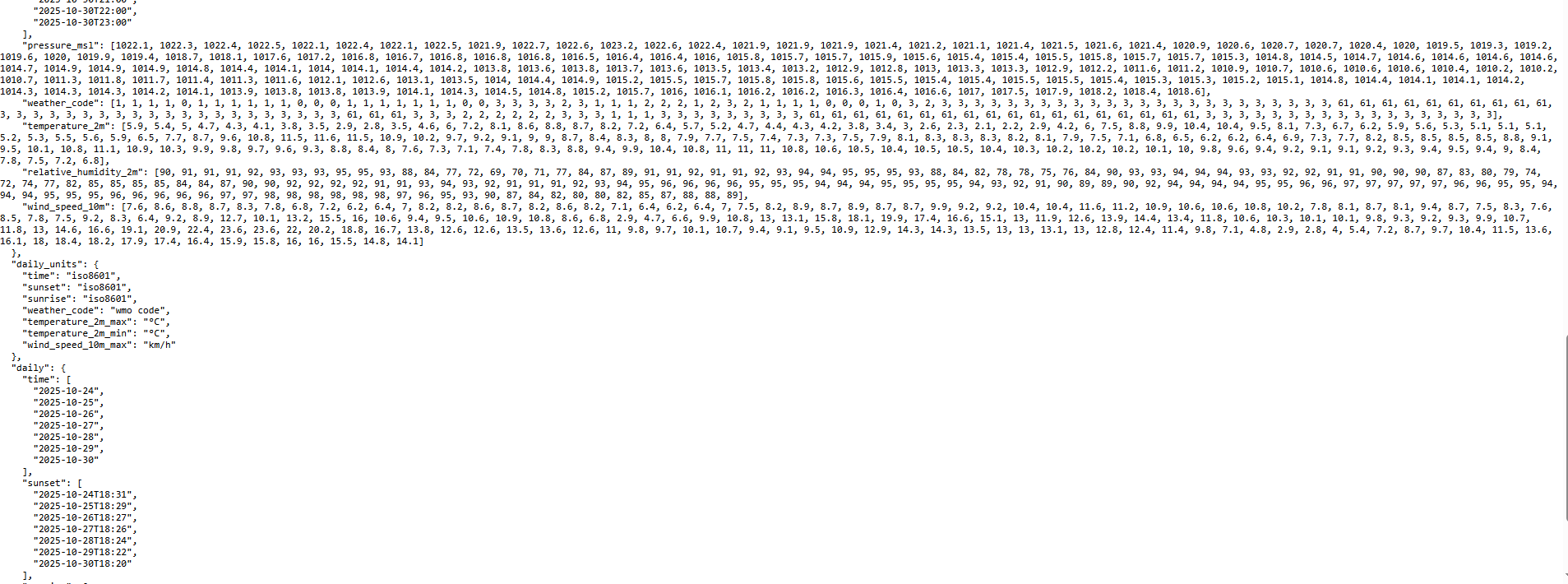


Рисунок 3.17 - Результат прогнозу Open-meteo (частина 2)

* + 1. Опис утіліт, бібліотек та іншого стороннього програмного забезпечення

Опис основних утиліт, бібліотек та іншого стороннього програмного забезпечення, що використовується у розробці наведено в таблиці 3.1.

Таблиця 3.1 – Опис утиліт, бібліотек та іншого стороннього ПЗ

|  |  |  |  |
| --- | --- | --- | --- |
| Частина застосунку | №  п/п | Назва | Опис застосування |
| Backend | 1 | FastAPI | Асинхронний веб-фреймворк для створення REST API на мові Python. Використовується для обробки HTTP-запитів від клієнтської частини. |
| 2 | Httpx [41] | Асинхронна бібліотека для виконання HTTP-запитів до зовнішніх сервісів. |

Продовження таблиці 3.1

|  |  |  |  |
| --- | --- | --- | --- |
| Частина застосунку | №  п/п | Назва | Опис застосування |
| Backend | 3 | Redis [42] | Бібліотека, що включає асинхронний модуль для вза’ємодії з in-memory сховищем Redis, що використовується для реалізації механізму кешування. |
| Frontend | 4 | React | Бібліотека JavaScript для створення динамічних та інтерактивних користувацький інтерфейсів. |
| 5 | React Router [43] | Бібліотека для реалізації маршрутизації в односторінковому додатку, що дозволяє навігацію між сторінками прогнозу, динаміки тощо. |
| 6 | Recharts [44] | Бібліотека для створення діаграм та графіків, що використовується для відображення динаміки погодних показників. |
| 7 | Tailwind CSS | CSS-фреймворк для швидкої та гнучної стилізації інтерфейсу користувача без написання додаткового коду в CSS-файлах. |

Продовження таблиці 3.1

|  |  |  |  |
| --- | --- | --- | --- |
| Частина застосунку | №  п/п | Назва | Опис застосування |
| Розгортання | 8 | Docker (з Docker Compose) | Платформа для контейнерізації, що дозволяє ізолювати та запускати frontend, backend, redis у вигляді окремих сервісів, які запускаються однаково на будь-якій машині із встановленним Docker, що спрощує розгортання. |
| 9 | Nginx | Веб-сервер, що використовується як зворотній проксі для маршрутизації запитів до backend-частини та для віддача статичних файлів frontend-частини у робочому середовищы |
| Середовище розробки та тестування | 10 | VS Code | Головне середовище розробки програмного забезпечення. |
| 11 | Postman [45] | Програмне забезпечення для тестування запитів до кінцевих точок backend-частини. |
| 12 | SonarQube [46] | Програмне забезпечення для тестування якості коду проєкту. |

Висновки до розділу

У даному розділі було детально описано процес конструювання та розробки програмного забезпечення для веб-сервісу прогнозування погоди, що є ключовим етапом реалізації проєкту.

Спочатку було представлено архітектуру програмного забезпечення, що базується на клієнт-серверній моделі з чітким поділом на frontend та backend частини. Для візуалізації архітектури на різних рівнях абстракції було використано діаграми моделі С4, які ілюструють як загальну структуру системи та її взаємодію із зовнішніми сервісами (рівні 1-2), так і внутрішню компонентну будову та модель коду backend-частини (рівні 3-4).

Далі було наведено обґрунтування вибору технологічного стеку. Для серверної частини було обрано Python з асинхронним фреймворком FastAPI, що забезпечує високу продуктивність. Клієнтська частина реалізована на JavaScript з використанням бібліотеки React для створення динамічних інтерфейсів. Для кешування даних було інтегровано Redis, а для розгортання та керування інфраструктурою – Docker та Nginx.

У рамках конструювання програмного забезпечення було надано детальний опис сторонніх сервісів, що є джерелами даних для системи: Geocodemaps, Openweathermap, Weatherapi та Open-meteo. Для кожного сервісу було описано його призначення, вхідні та вихідні дані, що дозволяє зрозуміти принципи інтеграції. На завершення було систематизовано опис ключових бібліотек, утиліт та іншого стороннього ПЗ, що було використано у проєкті.

Таким чином, у цьому розділі було представлено повну картину технічної реалізації проєкту: від архітектури до конкретних інструментів. Обрані рішення дозволили створити надійний, масштабований та продуктивний веб-сервіс, що повністю відповідає функціональним та нефункціональним вимогам, визначеним у попередньому розділі.

1. АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
   1. Аналіз якості ПЗ

Аналіз якості програмного забезпечення є комплексним процесом, що оцінює відповідність продукту як вимогам користувача, так і стандартам розробки.

Метриками для оцінки якості ПЗ обрано наступні:

* продуктивність (швидкість відповіді при звичайному режимі та під навантаженням);
* доступність (час безвідмовної роботи та швидкість відновлення, при відмові будь-якого з необхідних сервісів);
* зручність використання (інтуїтивність інтерфейсу);
* підтримуваність (якість коду).

Для перевірки продуктивності backend-частини застосуємо сервіс Postman. Спочатку протестуємо середній час відгуку без навантаження та отримаємо результати, які приблизно дорівнюють 1100 та 250 мс без заповненого кешу та із ним відповідно (рисунки 4.1-4.2). Далі протестуємо середній час відгуку та кількість помилок для 100 користувачів одночасно (рисунок 4.3) та отримаємо досить хороші результати, де кількість помилок, хоч й на початку досить велика, проте спадає пізніше й доходить до 0%. Усі помилки були викликані через обмеження безкоштовної версії API.



Рисунок 4.1 – Швидкість відповіді без кешу



Рисунок 4.2 – Швидкість відповіді з кешем

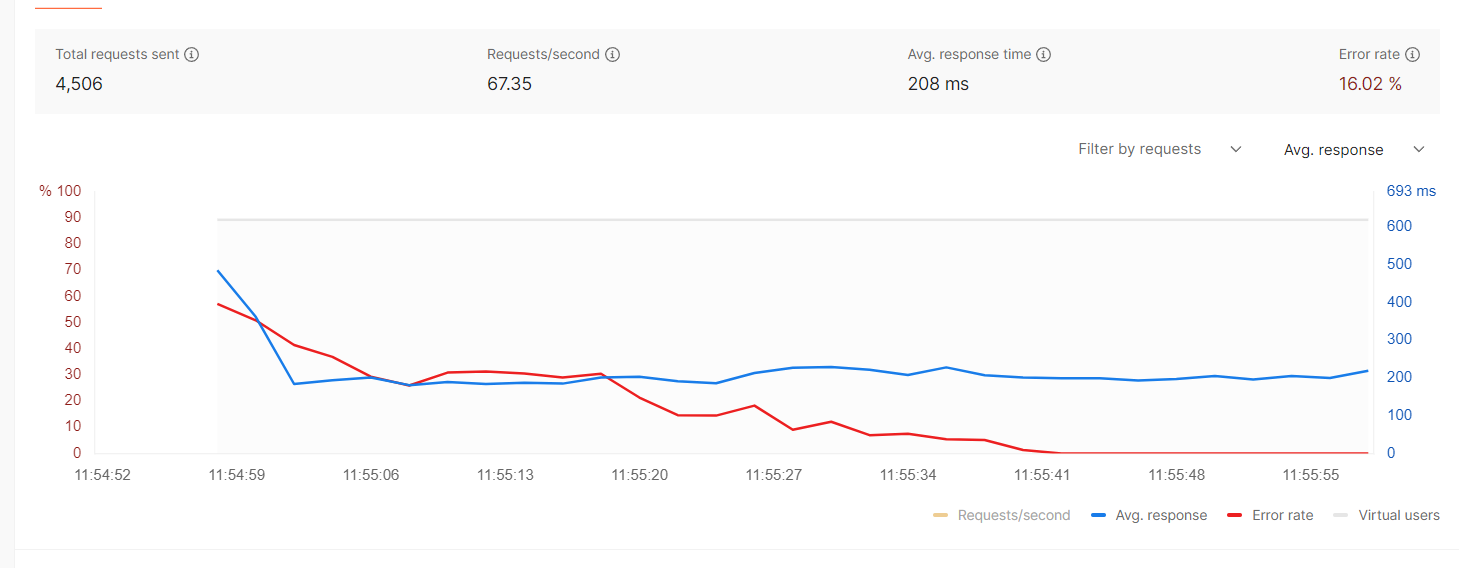


Рисунок 4.3 – Швидкість та кількість помилок при роботі під навантаженням

Тепер перевіримо frontend-частину як на швидкість, так і на зручність виокристання. Почнемо із аналізу швидкості та зручності, використовуючи вбудований механізм в браузер – Lighthouse. Після тестування отримано дуже хороші результати (рисунок 4.4). Також будь-який функціонал може бути досягненим за три кроки, що є досить зручним, а завантаження при повільному Інтернет з’єднанні в середньому займає 7.5 секунд (рисунок 4.5), що є достатнім результатом.

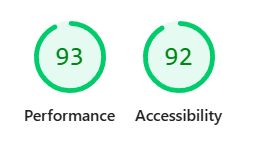


Рисунок 4.4 – Результати Lighthouse

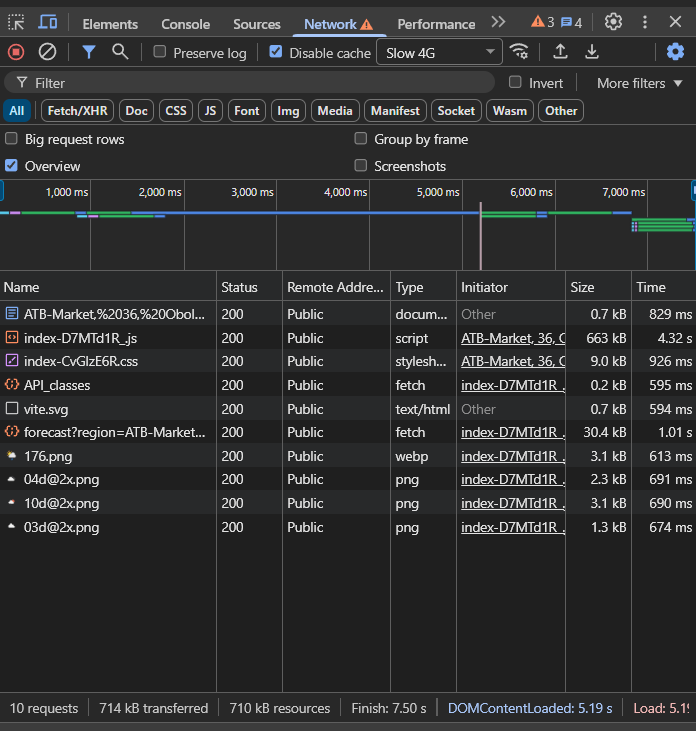


Рисунок 4.5 – Результати завантаження сторінки при повільному Інтернет з’єднанні

Наступним перевіримо веб-сторінку на доступність. Час за’явленої безвідомовної роботи серверу, на якому розгортається програмне забезпечення, 99.5%, що задовольняє ТЗ та нефункціональні вимоги. Отже, необхідно перевірити, що швидкість відновлення компонентів ПЗ достатня. Після перевірки за допомогою Postman, отримаємо результат, що приблизно дорвівнює п’яти секундам, тобто відповідає необхідним умовам (рисунок 4.6).

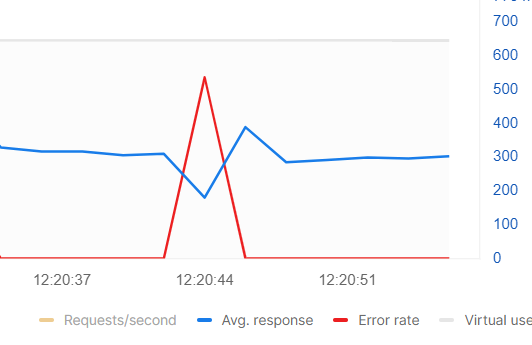


Рисунок 4.6 – Швидкість відновлення веб-сервісу

На останок перевіримо код на підтримуваність, що є важливим для подальшого вдосконалення ПЗ. Для цього застосуємо веб-застосунок SonarQube та отримаємо дуже хорошу загальну оцінку (рисунок 4.7), що підверджує якість коду.

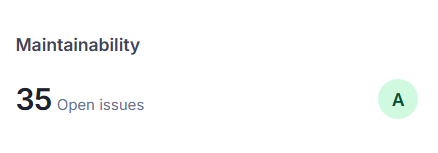


Рисунок 4.7 – Оцінка підтримуваності коду SonarQube

* 1. Опис процесів тестування

Тестування програмного забезпечення є критично важливим етапом життєвого циклу розробки, метою якого є перевірка відповідності продукту заявленим вимогам та виявлення дефектів. Існує багато типів тестування, які класифікують за різними критеріями. За рівнем тестування виділяють модульне, інтеграційне та системне. За метою – функціональне (перевірка відповідності функціональним вимогам) та нефункціональне (перевірка продуктивності, надійності, зручності використання, безпеки тощо). За ступенем автоматизації – мануальне та автоматизоване.

Як було детально продемонстровано у попередньому підрозділі (4.1 Аналіз якості ПЗ), оцінка ключових нефункціональних характеристик та аналіз якості коду вже були проведені з використанням спеціалізованих інструментів та аналізу сценаріїв. Ці дії фактично представляють собою результати нефункціонального тестування та статичного аналізу коду.

Даний підрозділ зосереджується на описі процесу функціонального тестування, спрямованого на перевірку коректності роботи всіх ключових функцій системи відповідно до сформованих функціональних вимог. Для даної курсової роботи було обрано саме мануальний підхід до функціонального тестування. Цей вибір обумовлений, по-перше, обсягом та складністю проєкту: розроблений веб-сервіс має чітко визначений та відносно невеликий набір основних функцій , що дозволяє ретельно перевірити кожен ключовий сценарій використання вручну. По-друге, академічний характер роботи передбачає демонстрацію розуміння процесу розробки та отримання працездатного прототипу, тоді як впровадження повноцінного автоматизованого тестування вимагало б значних додаткових зусиль. По-третє, на етапі розробки курсового проєкту частота змін функціоналу є помірною, що знижує потребу у частому регресійному тестуванні, де автоматизація найбільш ефективна.

Процес мануального функціонального тестування відбувався ітеративно паралельно з розробкою. Після реалізації кожної значної функціональної частини проводилося тестування відповідних сценаріїв. Для перевірки роботи кінцевих точок backend-частини використовувався інструмент Postman, а тестування frontend-частини здійснювалося безпосередньо у веб-браузері з використанням інструментів розробника.

В таблицях 4.1-4.7 наведено приклади ключових тест-кейсів, що були виконані для перевірки відповідності системи функціональним вимогам.

Таблиця 4.1 – Тест 1 Отримання прогнозу за геолокацією (FR-9)

|  |  |
| --- | --- |
| Початковий стан системи | Користувач вперше відкриває веб-сервіс у браузері. |
| Вхідні дані |  |
| Опис проведення тесту | 1. відкрити головну сторінку веб-сервісу;  2. дочекатися запиту браузеру на доступ до геолокації;  3. надати дозвіл натисканням «дозволити …». |
| Очікуваний результат | Після надання дозволу система автоматично визначає поточне місто користувача, завантажує та відображає агрегований прогноз погоди для нього. |
| Фактичний результат | Збігається з очікуваним. |

Таблиця 4.2 – Тест 2 Ручний пошук міста з підказками (FR-3, FR-8)

|  |  |
| --- | --- |
| Початковий стан системи | Головна сторінка веб-сервісу відкрита. |
| Вхідні дані | Часткова (або повна) назва міста, як наприклад, «Kyiv». |
| Опис проведення тесту | 1. у поле пошуку міста «City name» ввести назву існуючого міста;  2. дочекатися появи списку підказок під полем пошуку;  3. клікнути на відповідну підказку, яка найкраще підходить користувачу. |
| Очікуваний результат | Поле пошуку очищується. Система завантажує та відображає агрегований прогноз погоди для обраного міста. |
| Фактичний результат | Збігається з очікуваним. |

Таблиця 4.3 – Тест 3 Вибір/скасування вибору API (FR-1, FR-2, FR-3)

|  |  |
| --- | --- |
| Початковий стан системи | Відображено прогноз погоди для будь-якого міста. Усі доступні чекбокси API відмічені за замовчуванням. |
| Вхідні дані |  |
| Опис проведення тесту | 1. зняти позначку з одного із чекбоксів API;  2. зняти позначки з усіх API, крім одного;  3. спробувати зняти позначку з останнього; |
| Очікуваний результат | Після кроку 1 дані прогнозу оновлюються - у блоках зникають дані від деактивованого API, а також перераховуються усереднині результати, використовуючи результати активних API. Після кроку 2 відбувається той самий процес, що й піісля кроку 1. Крок 3 неможливий, оскільки система не дозволяє диактивувати усі чекбокси одночасно. |
| Фактичний результат | Збігається з очікуваним. |

Таблиця 4.4 – Тест 4 Перегляд узагальненого прогнозу (FR-4)

|  |  |
| --- | --- |
| Початковий стан системи | Відображено прогноз погоди для будь-якого міста та обрано секцію «Forecast» за замовчуванням. |
| Вхідні дані |  |
| Опис проведення тесту | 1. Переглянути секцію сторінки, де відображається прогноз погоди по днях |
| Очікуваний результат | На сторінці коректно відображаються блоки прогнозу для кожного доступного дня (згідно з даними API). Кожен блок містить основні денні показники (мін./макс. температура, вологість, швидкстіь вітру, іконки погоди, середні значення тощо). |

Продовження таблиці 4.4

|  |  |
| --- | --- |
| Фактичний результат | Збігається з очікуваним. |

Таблиця 4.5 – Тест 5 Перегляд детального прогнозу на день (FR-5)

|  |  |
| --- | --- |
| Початковий стан системи | Відображено прогноз погоди для будь-якого міста та обрано секцію «Forecast» за замовчуванням. |
| Вхідні дані |  |
| Опис проведення тесту | 1. клікнути на інтерактивний блок будь-якого дня з прогнозом. |
| Очікуваний результат | Відбувається перехід на нову сторінку. На екрані відображається детальний прогноз погоди для обраного дня, що містить показники, такі як температура, тиск, вологість, швидкість вітру тощо, для кожної години. |
| Фактичний результат | Збігається з очікуваним. |

Таблиця 4.6 – Тест 6 Перегляд та взаємодія з динамікою погоди (FR-6, FR-7)

|  |  |
| --- | --- |
| Початковий стан системи | Відображено прогноз погоди для будь-якого міст, обрано секцію «Forecast» за замовчуванням та всі API сервіси. |
| Вхідні дані |  |
| Опис проведення тесту | 1. клікнути на секцію «Dynamics»;  2. дочекатися завантаження сторінки з графіками;  3. клікнути на іконку для вибору іншого метеорологічного показника;  4. натиснути на початкову дату;  5. змінити початкову дату на наступну; |

Продовження таблиці 4.6

|  |  |
| --- | --- |
| Опис проведення тесту | 6. натиснути на кінцеву дату;  7. змінити кінцеву дату. |
| Очікуваний результат | Після кроку 1 відбувається перехід на сторінку динаміки та завантажується графік для температури, на якому зображені лінії усіх сервісів та усереднений варіант. Після кроку 3 відображається інший графік для відповідного показника. Після кроку 5 графік для будь-якого показника починається із відповідно обраної дати. Після кроку 7 графік будь-якого показника закінчується відповідно обраною датою. |
| Фактичний результат | Збігається з очікуваним. |

Таблиця 4.7 – Тест 7 Обробка некоректного введення міста (FR-3, FR-8)

|  |  |
| --- | --- |
| Початковий стан системи | Головна сторінка веб-сервісу відкрита. |
| Вхідні дані | Некоректна назва міста, як наприклад, «Test123». |
| Опис проведення тесту | 1. у поле пошуку міста «City name» ввести назву неіснуючого міста;  2. очікувати пошук можливих варіантів;  3. натиснути клавішу «Enter». |
| Очікуваний результат | Після кроку 1 з’являється повідомлення про завантаження варіантів «Loading…», після чого воно пропадає, а на його місці не з’являється жодного із варіантів. Після кроку 3 нічого не відбувається. |
| Фактичний результат | Збігається з очікуваним. |

* 1. Опис контрольного прикладу

Даний підрозділ ілюструє типовий сценарій взаємодії користувача з розробленим веб-сервісом прогнозування погоди. Метою є покрокова візуалізація основного функціоналу системи на конкретному прикладі, включаючи опис можливих варіантів дій користувача та особливостей роботи інтерфейсу.

При відкритті веб-сервісу користувачем (рисунок 4.8), система може запропонувати визначити місцезнаходження автоматично за допомогою геолокації браузера. Якщо користувач надає дозвіл, система одразу переходить до відображення прогнозу для поточного міста (рисунок 4.9). Якщо ж користувач користувач відмовляється від автоматичного визначення або бажає знайти погоду для іншого міста вручну. На екрані відображається поле для введення назви міста та перелік доступних для вибору API, усі з яких відмічені за замовчуванням (рисунок 4.10).

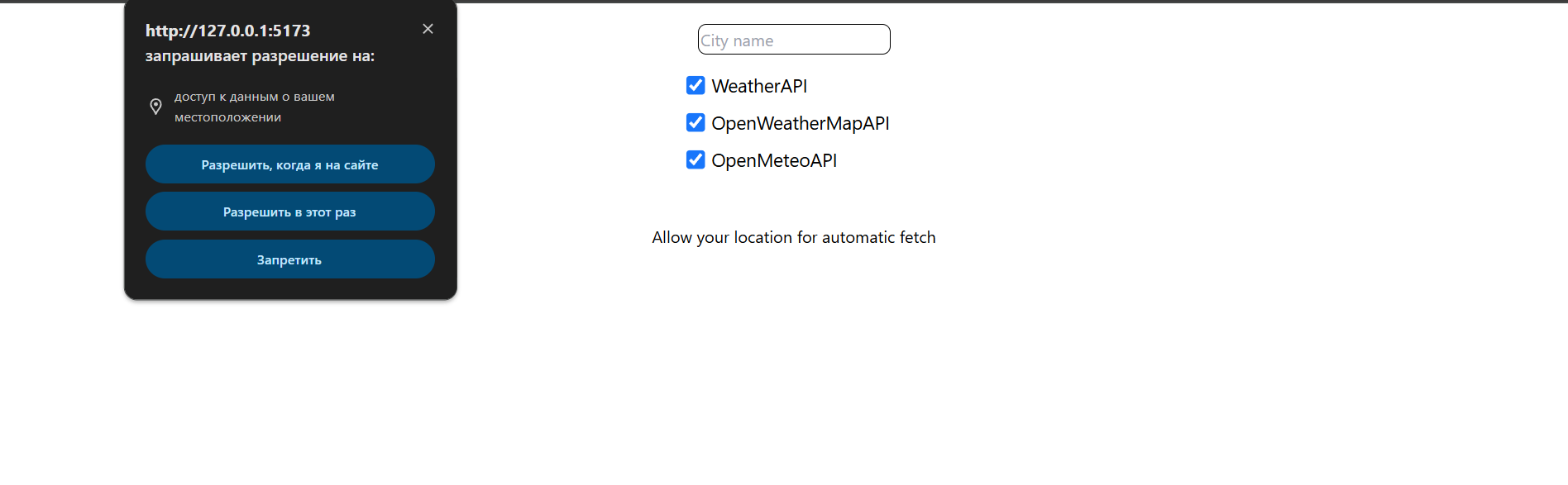


Рисунок 4.8 – Запит на геолокацію

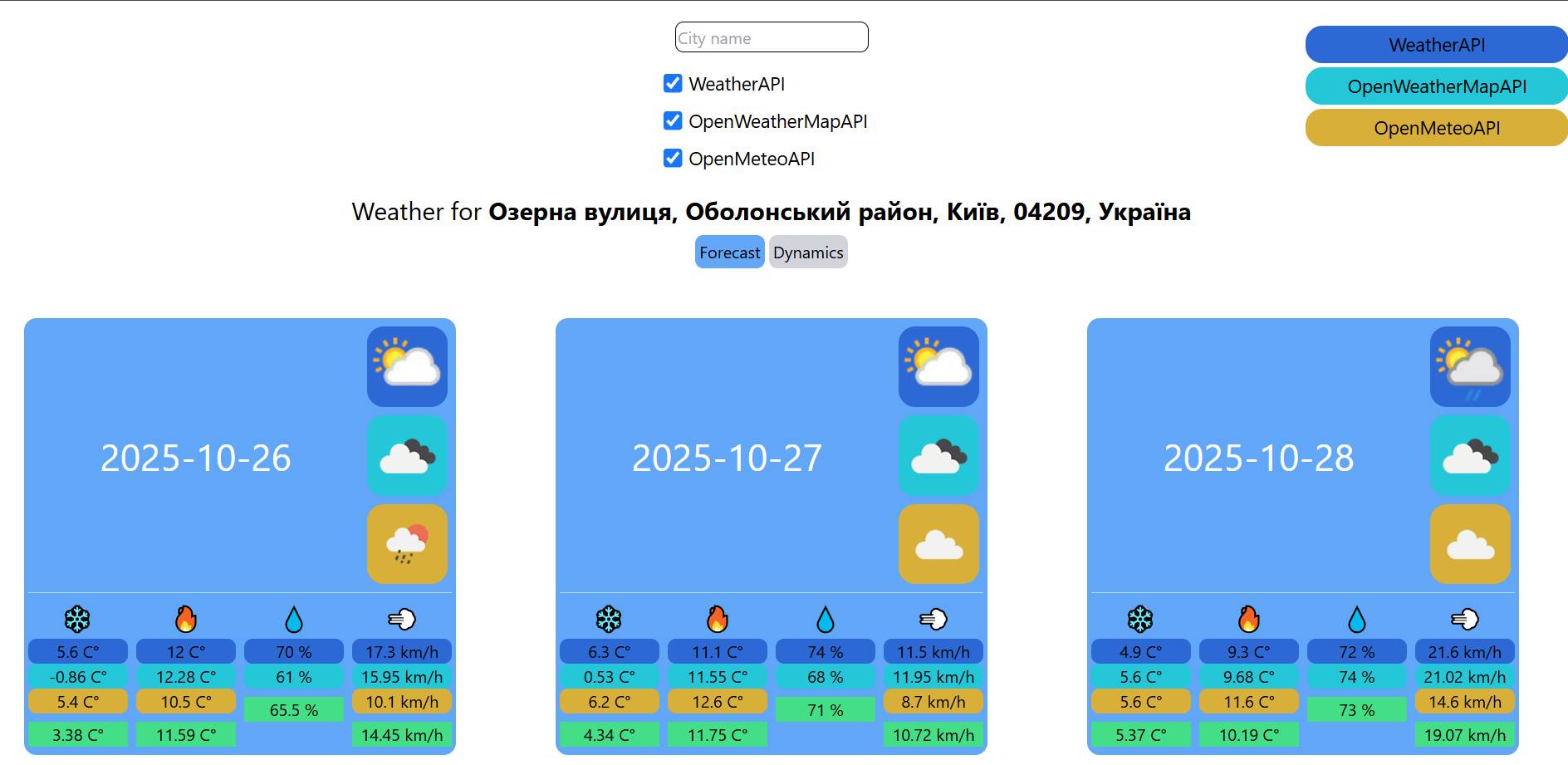


Рисунок 4.9 – Отримання прогнозу погоди при підтвердження доступу до геолокації

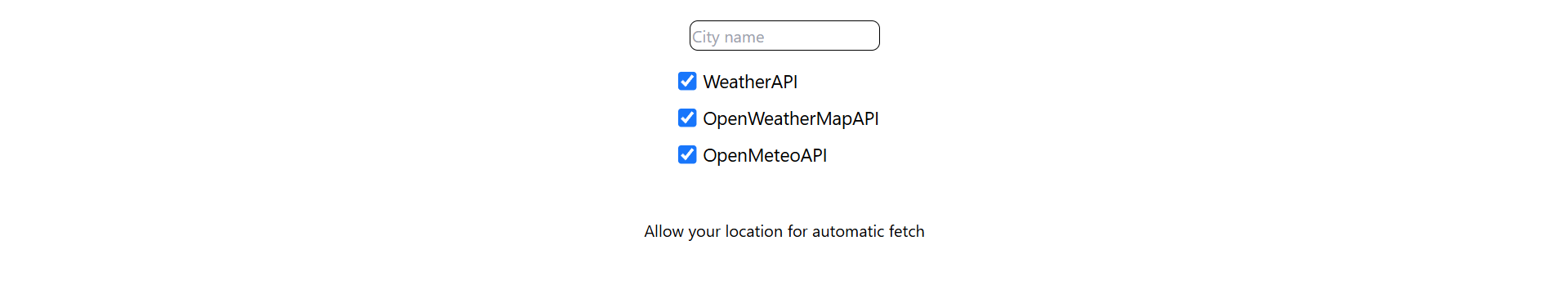


Рисунок 4.10 – Результат при відмові доступу до геолокації

Користувач може ввести назву міста у поле пошуку "City name". Система асинхронно надсилає запити до backend-сервісу для отримання варіантів автодоповнення. Під полем введення динамічно з'являється список підказок, що відповідають введеному тексту (рисунок 4.11). За умови введення некоректного міста, система не запропоновує варіантів, таким чином, користувач не може отримати прогноз для некоректного міста (рисунок 4.12).

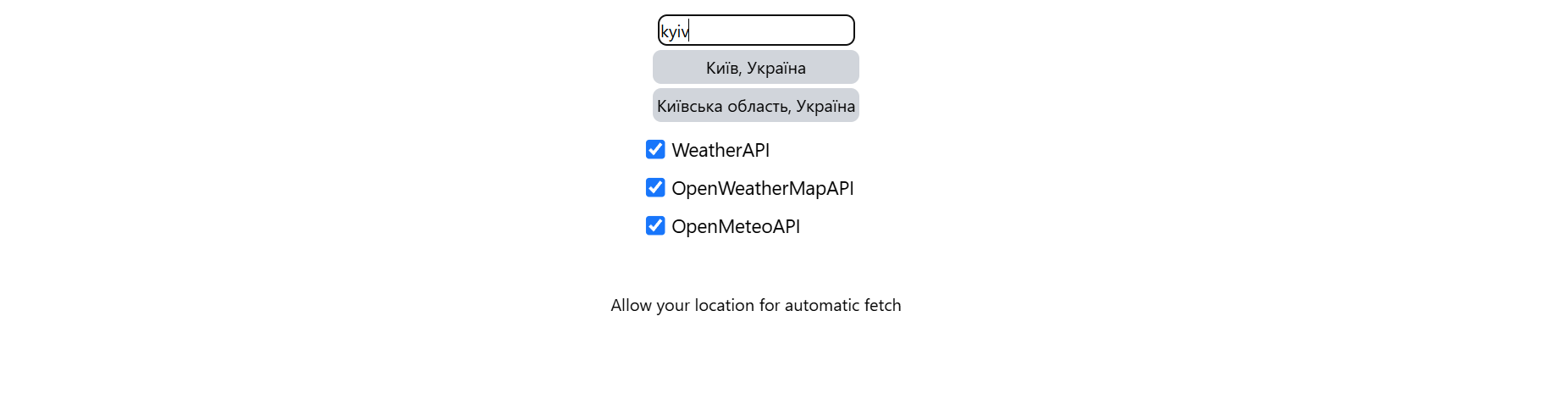


Рисунок 4.11 – Отримання варіантів пошуку по частковому введенні

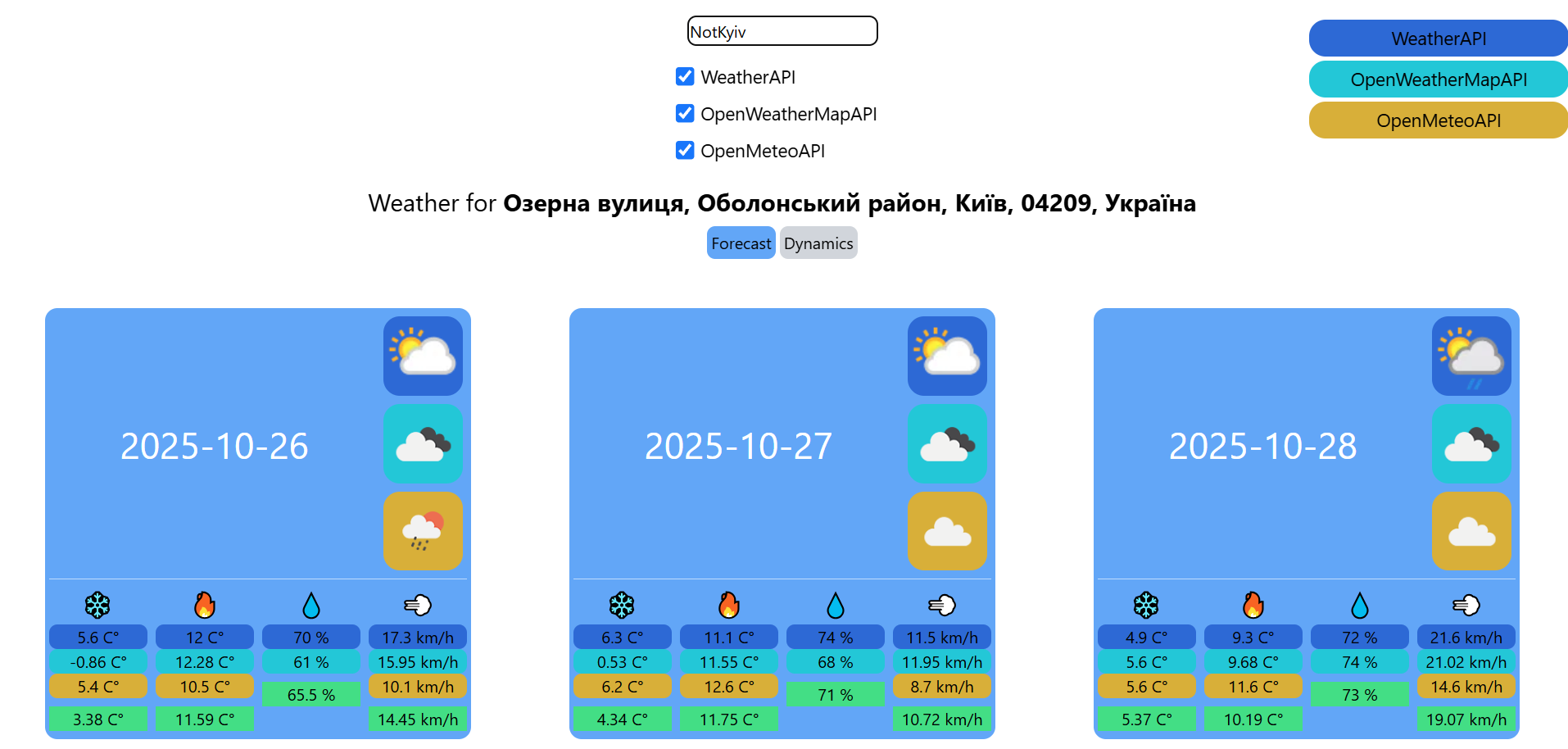


Рисунок 4.12 – Результат при введенні некоректного міста

При обранні варіанту із запрпонованих, система перенаправляє користувача на сторінку загального прогнозу по дням, видаляючи при цьому дані із поля «City name» (рисунок 4.13).

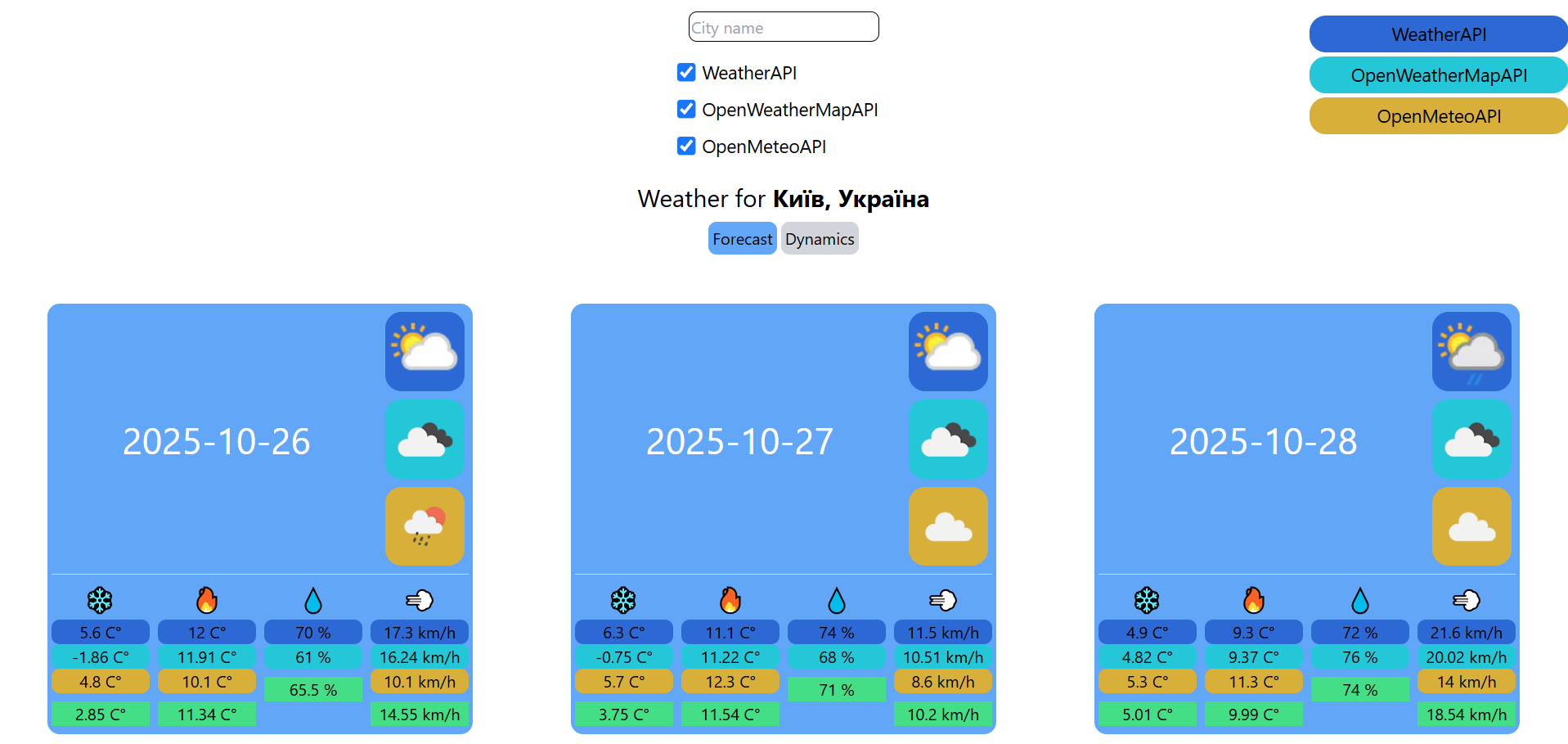


Рисунок 4.13 – Отримання прогнозу погоди після обрання варіанту

Для отримання інформації по годиннам, необхідно клікнути на будь-який денний блок, після чого відбудиться перенаправлення на відповідну сторінку із погодинної інформацією (рисунок 4.14). Для повернення на поденний прогноз можна використати відповідну кнопку «Back to daily forecast» в лівому верхньому куту застосунку.

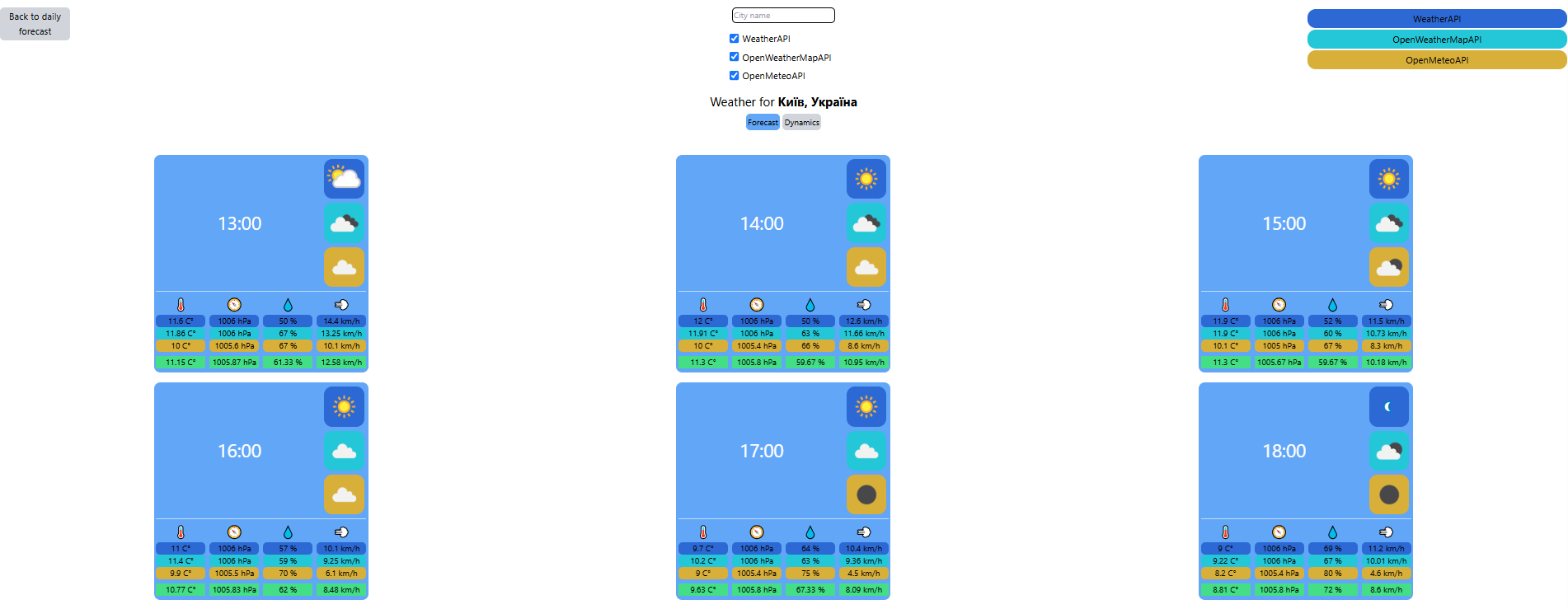


Рисунок 4.14 – Результати для конкретного обраного дня

Якщо користувач бажає використовувати тільки деякі сервіси прогнозу для агрегованого результату, він має можливість деактивувати відповідні чекбокси. При цьому результуючий прогноз буде змінений (рисунок 4.15). Вимикання усіх чекбоксів є неможливим, тому, при спробі вимкнути останній активний, система не дозволить це зробити.

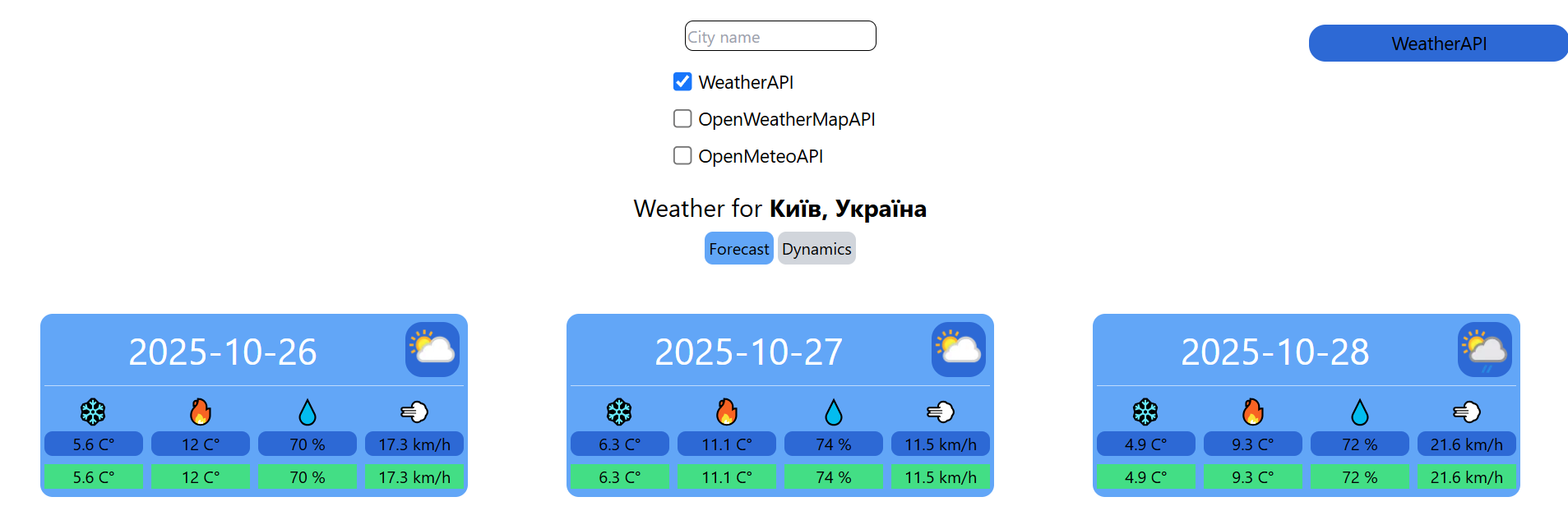


Рисунок 4.15 – Спроба деактивування усіх сервісів прогнозу

Для отримання погодинної динаміки параметрів прогнозу, необхідно вибрати секцію «Dynamics» натисканням на неї. Після чого відбудеться перехід на відповідну сторінку та рендеринг графіку із обраним індикатором за замовчуванням (рисунок 4.16). Для зміни індикатора можна натиснути на необхідну іконку справа і графік буде перебудовано (рисунок 4.17). Також можна змінити початкову та кінцеву дату. Для цього необхідно натиснути на відповідну дату під графіком (рисунок 4.18) та обрати необхідну, після чого графік буде перебудовано, починаючи та закінчуючи обраними датами (4.19-4.20)

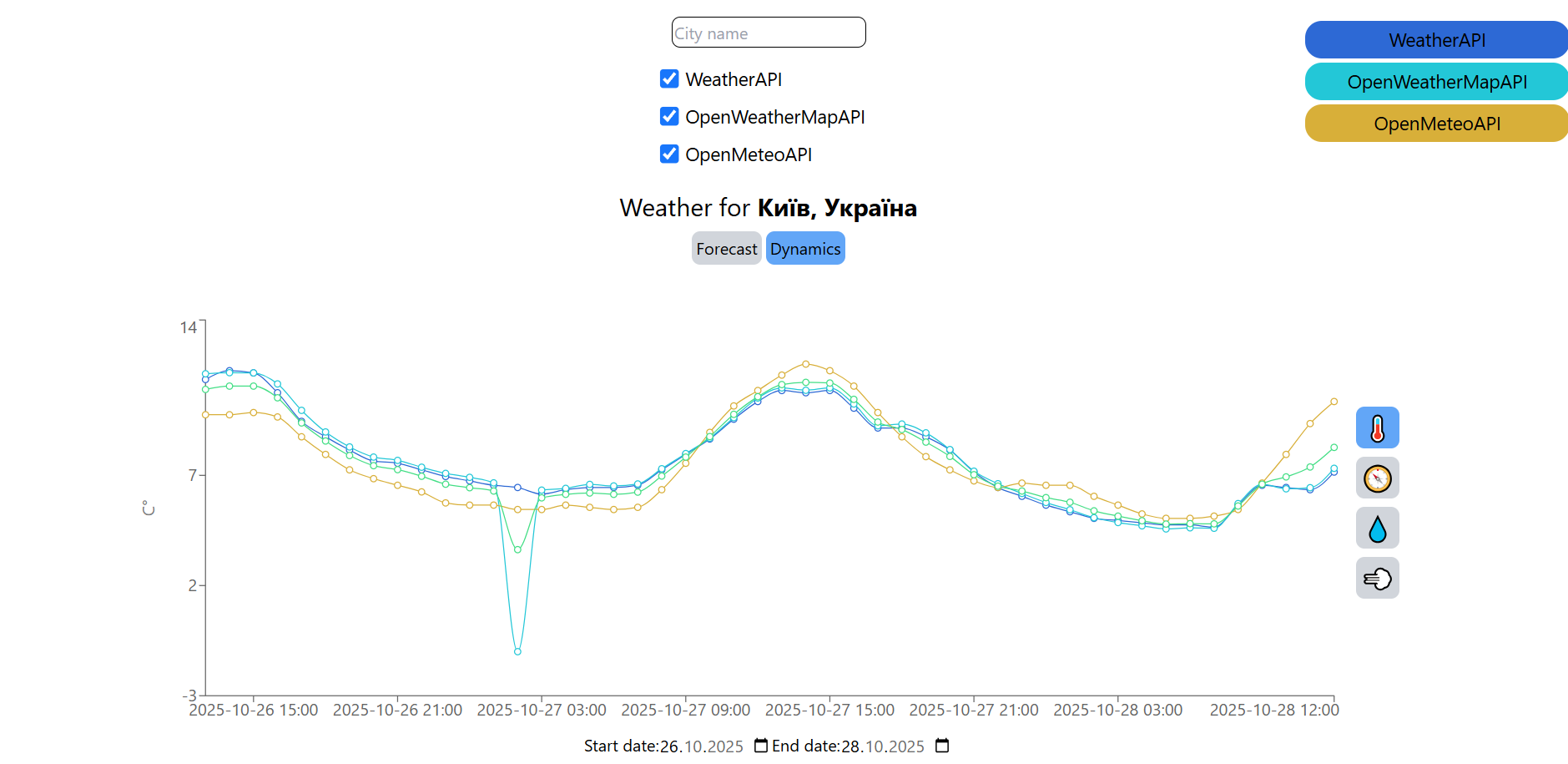


Рисунок 4.16 – Вибір секції динаміки прогнозу погоди

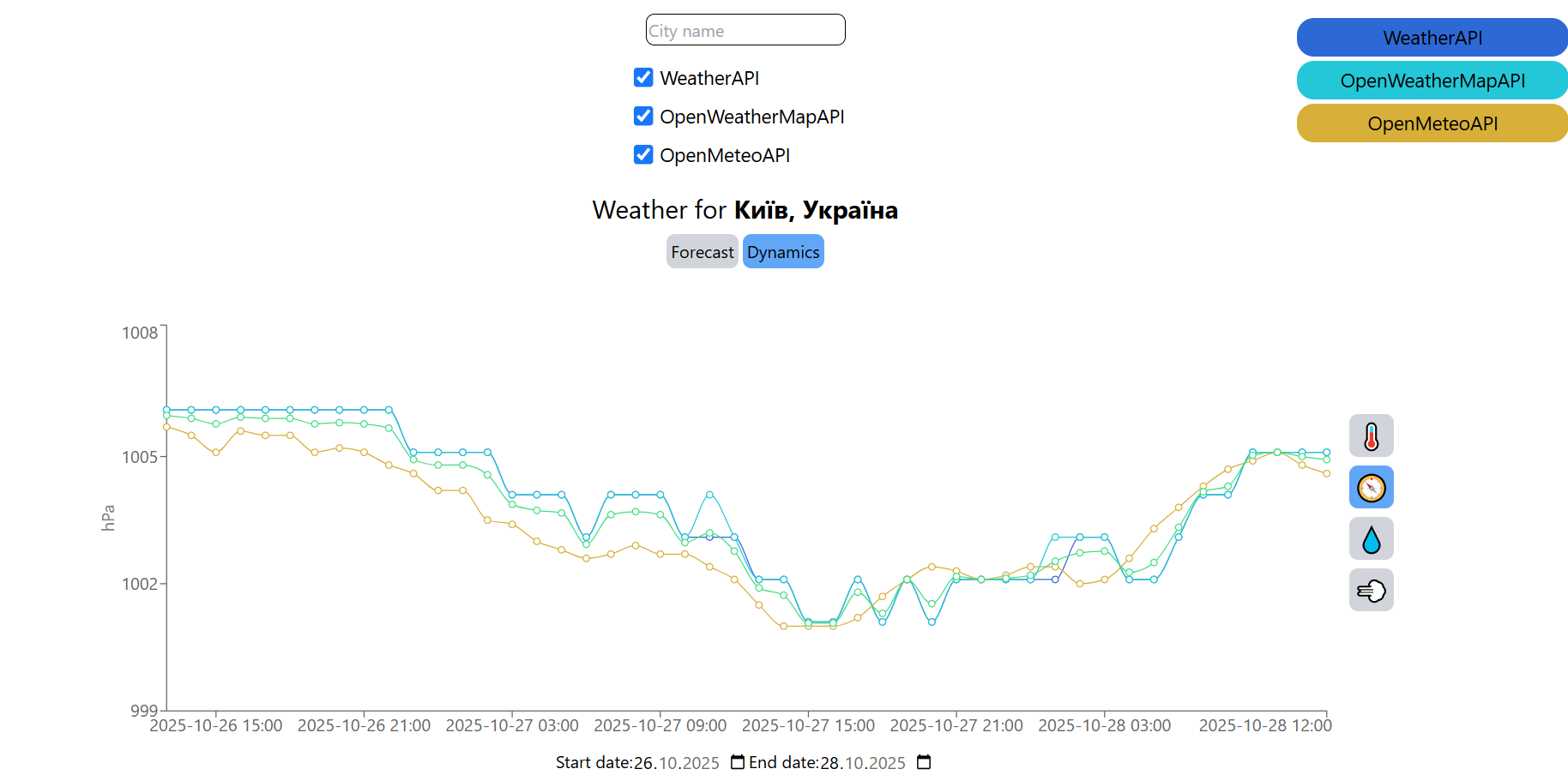


Рисунок 4.17 – Зміна індикатора погоди для динаміки

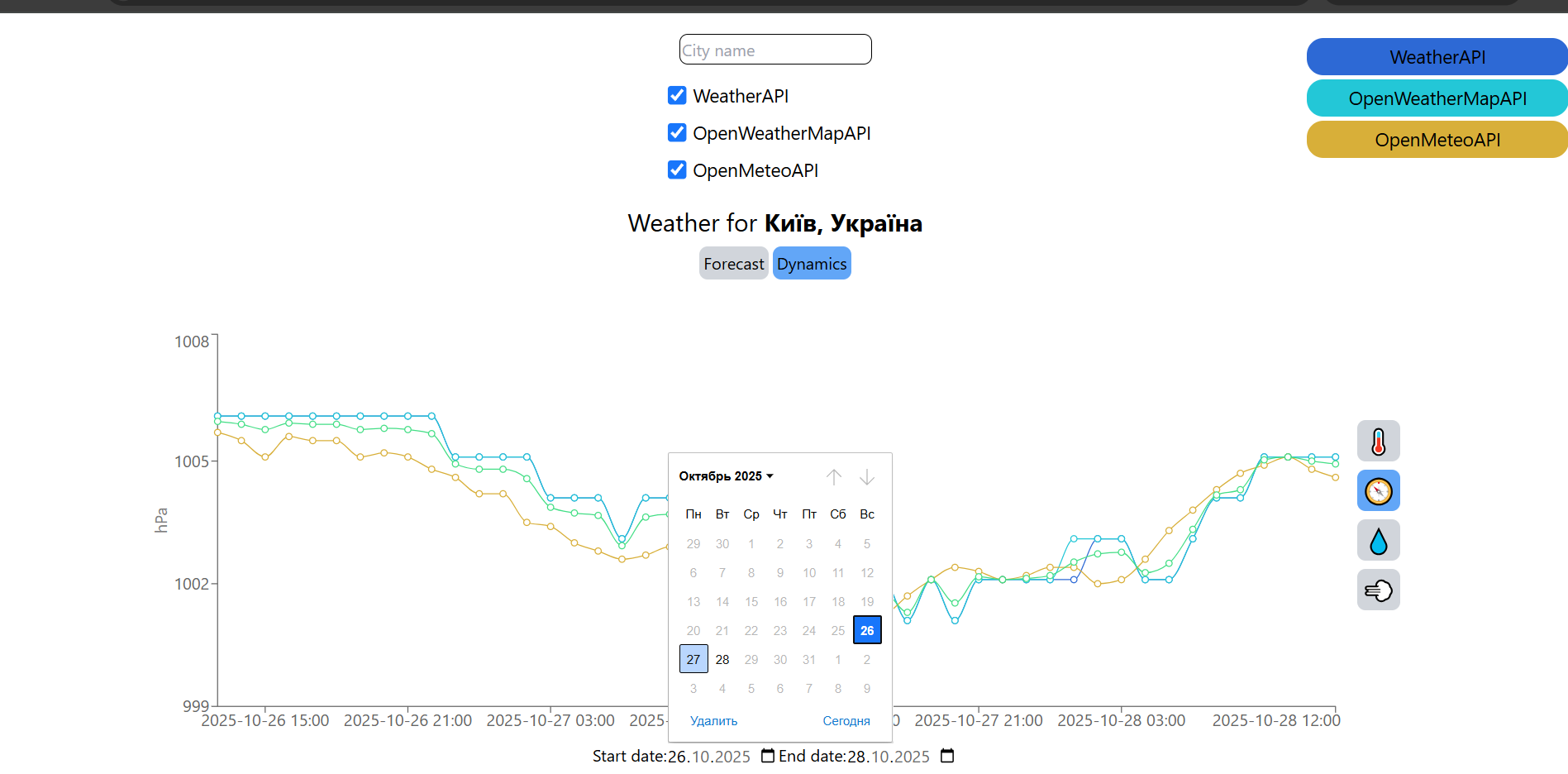


Рисунок 4.18 – Зміна початкової дати динаміки



Рисунок 4.19 – Результат після зміни початкової дати

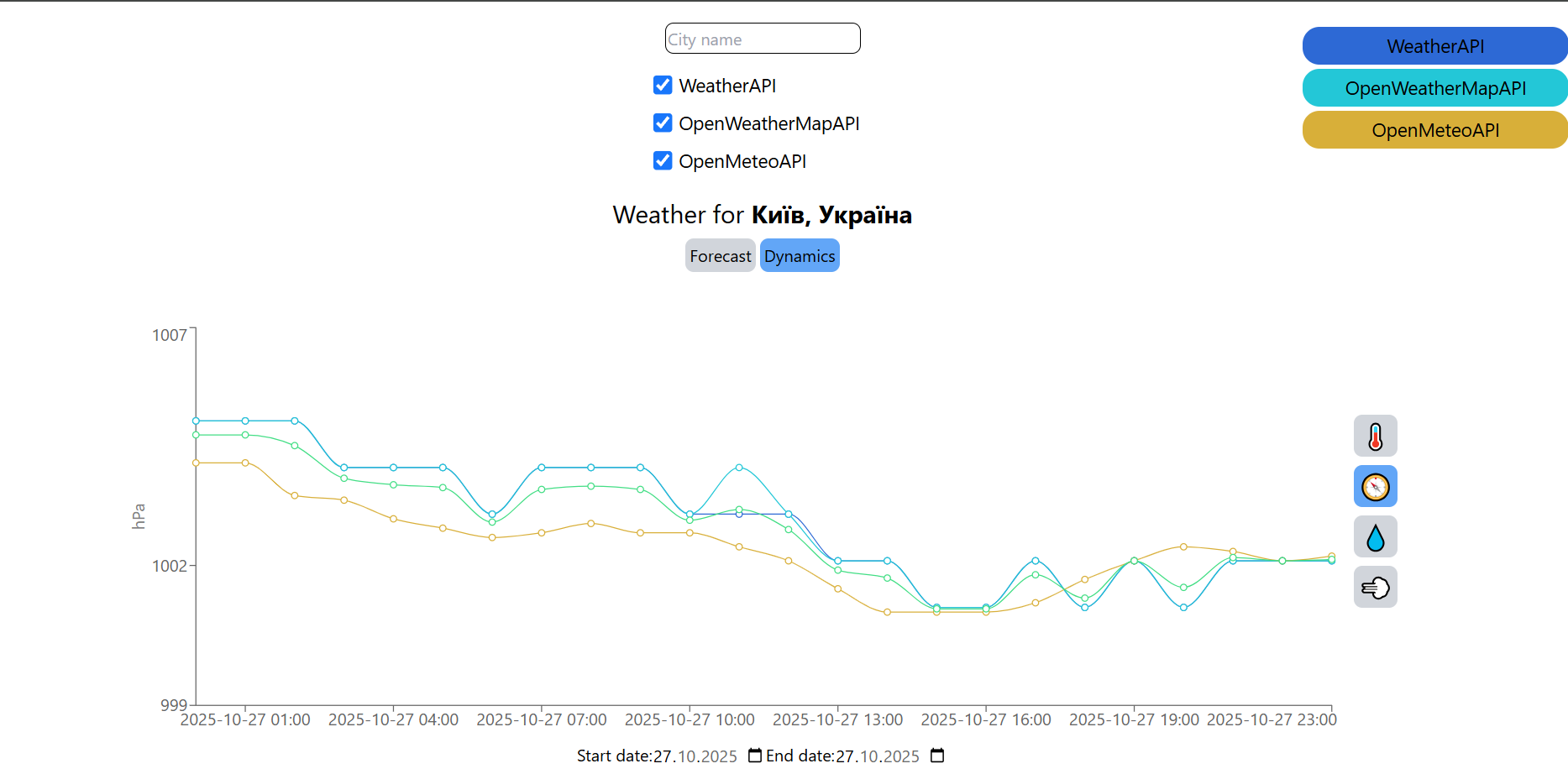


Рисунок 4.20 – Результат після зміни кінцевої дати динаміки

Висновки до розділу

У даному розділі було проведено аналіз якості розробленого програмного забезпечення та описано процеси його тестування, що є важливими етапами для підтвердження відповідності продукту встановленим вимогам.

Спочатку було виконано аналіз якості ПЗ за ключовими метриками: продуктивність, доступність, зручність використання та підтримуваність. Для кожної метрики було проведено оцінку з використанням відповідних інструментів, таких як: Postman для вимірювання часу відповіді backend-частини, Google Lighthouse для оцінки продуктивності та доступності frontend-частини, а також було зімітовано навантаження та збої для перевірки роботи з великою кількістю одночаних користувачів та швидкості відновлення. Також було використано SonarQube для аналізу якості коду та його підтримуваності.

Далі було надано короткий огляд основних типів тестування та обґрунтовано вибір мануального функціонального тестування як основного підходу для даної курсової роботи. Також було описано методику проведення тестування та представлено набір ключових тест-кейсів у вигляді таблиць, що охоплюють перевірку всіх основних функціональних вимог.

На завершення розділу було наведено опис контрольного прикладу, який покроково, з ілюстраціями, демонструє типовий сценарій взаємодії користувача з веб-сервісом: від пошуку міста до отримання агрегованого прогнозу, перегляду детальної інформації та аналізу динаміки погоди. У прикладі також було показано основні альтернативні шляхи та особливості роботи інтерфейсу.

Таким чином, у цьому розділі було представлено докази якості розробленого програмного забезпечення через аналіз ключових метрик та демонстрації коректності роботи основного функціоналу за допомогою тест-кейсів та детального контрольного прикладу. Результати підтверджують, що створений веб-сервіс є працездатним, відповідає поставленим вимогам та готовий до використання.

1. ПОВНА ІНСТРУКЦІЯ КОРИСТУВАЧА
   1. Встановлення програмного забезпечення

<У підрозділі викладають:

повний опис покрокового встановлення ПЗ. Кроки доповнюють ілюстраціями.>

* 1. Інструкція користувача

<У підрозділі викладають:

опис того, як користувач має працювати з програмним забезпеченням від установки до завершення роботи. Обов’язково мають бути ілюстрації.>

Висновки до розділу

< Необхідно стисло описати усе, що було виконано у даному розділі. Обсяг 0,75-1 сторінка>

**ВИСНОВКИ**

**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

1. Web-application. URL: <https://www.javatpoint.com/web-application> (дата звернення: 01.11.2024).
2. Weather. URL: <https://weather.com/> (дата звернення: 01.11.2024).
3. Accuweather. URL: <https://www.accuweather.com/> (дата звернення: 01.11.2024).
4. Worldweather. URL: <https://worldweather.wmo.int/> (дата звернення: 01.11.2024).
5. Weather app market size. URL: <https://www.grandviewresearch.com/industry-analysis/weather-app-market-report> (дата звернення: 01.11.2024).
6. Data-driven weather forecasting. URL: <https://www.mdpi.com/2073-4433/15/6/689> (дата звернення: 01.11.2024).
7. Forecast process. URL: <https://www.weather.gov/about/forecast-process> (дата звернення: 01.11.2024).
8. Introducing weather API. URL: [https://blog.weatherstack.com/blog/introducing-real-time-weather-data-api/](https://blog.weatherstack.com/blog/introducing-real-time-weather-data-api/%20) (дата звернення: 01.11.2024).
9. Architecture patterns. URL: <https://www.geeksforgeeks.org/types-of-software-architecture-patterns/> (дата звернення 25.04.2025).
10. Web hosting. URL: <https://www.geeksforgeeks.org/types-of-web-hosting> (дата звернення 25.04.2025).
11. Python. URL: <https://www.python.org/> (дата звернення 25.04.2025).
12. Java. URL: <https://www.java.com/> (дата звернення 25.04.2025).
13. Go. URL: <https://go.dev/> (дата звернення 25.04.2025).
14. PHP. URL: [https://www.php.net/](https://www.php.net/%20) (дата звернення 25.04.2025).
15. FastAPI. URL: <https://fastapi.tiangolo.com/> (дата звернення 25.04.2025).
16. Flask. URL: <https://flask.palletsprojects.com/en/stable/> (дата звернення 25.04.2025).
17. Django. URL: <https://www.djangoproject.com/> (дата звернення 25.04.2025).
18. JavaScript. URL: <https://developer.mozilla.org/ru/docs/Web/JavaScript> (дата звернення 25.04.2025).
19. React. URL: <https://react.dev/> (дата звернення 25.04.2025).
20. Vue. URL: <https://vuejs.org/> (дата звернення 25.04.2025).
21. Angular. URL: [https://vuejs.org/](https://vuejs.org/%20) (дата звернення 25.04.2025).
22. Tailwind. URL: <https://tailwindcss.com/> (дата звернення 25.04.2025).
23. Bootstrap. URL: <https://getbootstrap.com/> (дата звернення 25.04.2025).
24. Docker. URL: <https://docs.docker.com/> (дата звернення 25.04.2025).
25. Kubernetes. URL: <https://kubernetes.io/> (дата звернення 25.04.2025).
26. Nginx. URL: <https://nginx.org/> (дата звернення 25.04.2025).
27. Redis. URL: <https://redis.io/> (дата звернення 25.04.2025).
28. Memcached. URL: <https://memcached.org/> (дата звернення 25.04.2025).
29. MongoDB. URL: <https://www.mongodb.com/> (дата звернення 25.04.2025).
30. Openweathermap. URL: <https://openweathermap.org/> (дата звернення 25.04.2025).
31. Weatherapi. URL: <https://www.weatherapi.com/> (дата звернення 25.04.2025).
32. Open-meteo. URL: [https://open-meteo.com/](https://open-meteo.com/%20) (дата звернення 25.04.2025).
33. Tomorrow. URL: <https://www.tomorrow.io/> (дата звернення 25.04.2025).
34. Weatherstack. URL: <https://weatherstack.com/> (дата звернення 25.04.2025).
35. Apache. URL: <https://httpd.apache.org/> (дата звернення 25.04.2025).
36. Visual Studio Code. URL: <https://code.visualstudio.com/> (дата звернення 25.04.2025).
37. PyCharm. URL: <https://www.jetbrains.com/pycharm/> (дата звернення 25.04.2025).
38. WebStorm. URL: <https://www.jetbrains.com/webstorm/> (дата звернення 25.04.2025).
39. BPMN. URL: <https://en.wikipedia.org/wiki/Business_Process_Model_and_Notation> (дата звернення 25.04.2025).
40. С4. URL: [https://c4model.com/](https://c4model.com/%20) (дата звернення 25.04.2025).
41. Httpx Python library: <https://www.python-httpx.org/>
42. Redis Python library: <https://redis.io/docs/latest/develop/clients/redis-py/> (дата звернення 21.10.2025).
43. React Router JavaScript library: <https://reactrouter.com/> (дата звернення 21.10.2025).
44. Recharts JavaScript library: <https://recharts.github.io/> (дата звернення 21.10.2025).
45. Postman: <https://www.postman.com/> (дата звернення 21.10.2025).
46. SonarQube: <https://www.sonarsource.com/products/sonarqube/> (дата звернення 21.10.2025).
47. Geocodemaps: <https://geocode.maps.co/> (дата звернення 21.10.2025).