НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«До захисту допущено»
Завідувач кафедри
_____ Євгенія СУЛЕМА
«___» _____ 2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»

спеціальності 121 Інженерія програмного забезпечення

на тему: «Вебзастосунок для інформаційної підтримки роботи метеостанції»

Виконав:
студентка IV курсу, групи КП-11
Пісоцька Софія Олександрівна _____

Керівник:
доцент кафедри ПЗКС, к.т.н., доцент,
Новак Дмитро Сергійович _____

Консультант з нормоконтролю:
доцент кафедри ПЗКС, к.т.н., доцент,
Онай Микола Володимирович _____

Рецензент:
доцент кафедри ІТШІК, к.т.н., доцент,
Мошенський Андрій Олександрович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____

Київ – 2025 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Євгенія СУЛЕМА
"___" _____ 2024 р.

ЗАВДАННЯ

на дипломний проєкт студентці

Пісоцькій Софії Олександрівні

1. Тема проєкту «Вебзастосунок для інформаційної підтримки роботи метеостанції», керівник проєкту Дмитро Сергійович Новак, к.т.н., доцент, затверджені наказом по університету №1808-С від «29» травня 2025 р.

2. Термін подання студентом проєкту «13» червня 2025 р.

3. Вихідні дані до проєкту: див. Технічне завдання.

4. Зміст пояснювальної записки:
   - аналіз предметної області та існуючих рішень;
   - аналіз мов програмування та технологій розроблення вебзастосунків та систем для метеостанції;
   - структурна організація розроблених програмних засобів;
   - аналіз реалізації програмного забезпечення;

5. Перелік обов'язкового графічного матеріалу:
   - функціональність вебзастосунку (креслення);
   - структура бази даних (креслення);
   - алгоритми вебзастосунку для інформаційної підтримки метеостанції (плакат);
   - структурна схема вебзастосунку (плакат).

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент		

7. Дата видачі завдання «31» жовтня 2024 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Вивчення літератури за тематикою проєкту	13.11.2024	
2.	Розроблення та узгодження технічного завдання	22.11.2024	
3.	Розроблення структури web-ресурсу	15.12.2024	
4.	Підготовка матеріалів першого розділу дипломного проєкту	28.12.2024	
5.	Розроблення дизайну сторінок та графічних елементів	01.02.2025	
6.	Підготовка матеріалів другого розділу дипломного проєкту	19.02.2025	
7.	Програмна реалізація web-ресурсу	11.03.2025	
8.	Тестування web-ресурсу	18.03.2025	
9.	Підготовка матеріалів третього розділу дипломного проєкту	27.03.2025	
10.	Підготовка матеріалів четвертого розділу дипломного проєкту	15.04.2025	
11.	Підготовка графічної частини дипломного проєкту	22.04.2025	
12.	Оформлення документації дипломного проєкту	27.05.2025	

Студент Софія ПІСОЦЬКА

Керівник проєкту Дмитро НОВАК

АНОТАЦІЯ

Дипломний проєкт присвячений розробці вебзастосунку для інформаційної підтримки роботи метеостанції, призначеного для збору, аналізу, візуалізації та прогнозування метеорологічних даних. Система орієнтована на потреби аграрного сектору, енергетики та служби з надзвичайних ситуацій. Основною метою є створення платформи, яка забезпечує доступ до актуальної та історичної погодної інформації з метеостанцій.

Функціонал системи охоплює відображення поточних погодних умов, побудову графіків довгострокових кліматичних змін, карту з метеоданими (супутникові знімки, радари), а також систему сповіщень про небезпечні погодні явища. У веб-застосунку реалізовано підключення до сторонніх погодних API, а також можливість обробки локальних даних з фізичної метеостанції. Інформаційна безпека реалізована через розмежування доступу: гості мають обмежений перегляд, а зареєстровані користувачі можуть користуватись розширеними функціями, зокрема адмініструванням та завантаженням архівних даних.

У рамках дипломного проєкту розроблено: архітектуру системи, алгоритми збору та синхронізації даних, систему авторизації, динамічний інтерфейс користувача та модулі візуалізації. Проєкт демонструє сучасний підхід до створення інформаційних систем для реального моніторингу клімату.

ABSTRACT

This diploma project is dedicated to the development of a web application for weather station, designed for the collection, analysis, visualization, and forecasting of meteorological data. The system is focused on the needs of the agricultural sector, energy industry, and emergency response services. Its main goal is to provide a platform that delivers access to current and historical weather data from a meteorological stations.

The system's functionality includes real-time weather updates, long-term climate change visualizations, a map with integrated meteorological data (such as satellite imagery and radar), and alerts for hazardous weather conditions. The application integrates third-party weather APIs and supports processing of local data from a physical weather station. Information security is ensured by access level separation: guest users have limited viewing rights, while registered users have extended functionality including administration and archive data management.

Within the scope of this diploma project, the following were developed: system architecture, data collection and synchronization algorithms, user authorization system, dynamic user interface, and visualization modules. The project demonstrates a modern approach to building information systems for real-time climate monitoring.

ДП.045440-01-90 Вебзастосунок для інформаційної підтримки роботи метеостанції. Відомість проєкту

Позначення	Найменування	Кіл-ть	Примітка
	Документація проєкту		
ДП.045440-02-91	Вебзастосунок для	5	
	інформаційної підтримки		
	роботи метеостанції.		
	Технічне завдання		
ДП.045440-03-81	Вебзастосунок для	67	
	інформаційної підтримки		
	роботи метеостанції.		
	Пояснювальна записка		
ДП.045440-04-51	Вебзастосунок для	4	
	інформаційної підтримки		
	роботи метеостанції.		
	Програма та		
	методика тестування		
ДП.045440-05-34	Вебзастосунок для	10	
	інформаційної підтримки		
	роботи метеостанції.		
	Керівництво		
	користувача		
ДП.045440-06-99	Вебзастосунок для	1	
	інформаційної підтримки		
	роботи метеостанції.		
	Структура бази		
	даних. ER-діаграма		

Позначення	Найменування	Кіл-ть	Примітка
ДП.045440-07-99	Вебзастосунок для	1	
	інформаційної підтримки		
	роботи метеостанції.		
	Функціональність		
	вебзастосунку. UML-		
	діаграма прецедентів		
ДП.045440-08-98	Вебзастосунок для	1	
	інформаційної підтримки		
	роботи метеостанції.		
	Компакт-диск		

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Євгенія СУЛЕМА

“___” _____ 2024 р.

ВЕБЗАСТОСУНОК ДЛЯ ІНФОРМАЦІЙНОЇ ПІДТРИМКИ РОБОТИ МЕТЕОСТАНЦІЇ

Технічне завдання

ДП.045440-02-91

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Дмитро НОВАК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Софія ПІСОЦЬКА

2024

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ…….3
2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ……3
3. ПРИЗНАЧЕННЯ РОЗРОБКИ…….3
4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ……..3
5. ВИМОГИ ДО ПРОЄКТНОЇ ДОКУМЕНТАЦІЇ……4
6. ЕТАПИ ПРОЄКТУВАННЯ……5
7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ……5

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: вебзастосунок для інформаційної підтримки роботи метеостанції.

Галузь застосування: інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проєктування, затверджене кафедрою програмного забезпечення комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для використання як інформаційна система моніторингу метеорологічних умов, що надає користувачам доступ до актуальних, архівних та прогнозованих кліматичних даних. Система орієнтована на фахівців аграрного сектору, енергетики, екологічного моніторингу та служб з надзвичайних ситуацій. Вебзастосунок забезпечує централізований доступ до метеоданих із сторонніх погодних API.

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Вебзастосунок для інформаційної підтримки роботи метеостанції повинен забезпечувати такі основні функції:

1) відображення поточних погодних умов у режимі реального часу;
2) візуалізація історичних кліматичних змін у вигляді інтерактивних графіків;
3) побудова метеокарт на основі супутникових знімків та радарів;

4) підтримка функцій сповіщення про небезпечні погодні явища;
5) динамічне оновлення інформації без перезавантаження сторінки;
6) авторизація та розмежування прав доступу (гості, зареєстровані користувачі, преуміум користувачі);
7) можливість завантаження персоналізованих даних зареєстрованими користувачами;
8) інтеграція з зовнішніми API для метеоданих.

Розробку виконати на платформі Node.js з використанням фреймворку React для клієнтської частини та бібліотек Chart.js для побудови графіків.

Додаткові вимоги:

1) наявність динамічного інтерфейсу (SPA);
2) адаптивна верстка для коректного відображення на різних пристроях;
3) локалізація вебзастосунку українською мовою;
4) наявність анімованих кнопок;
5) використання дизайну з заспокійливими блакитним та зеленим кольорами.

5. ВИМОГИ ДО ПРОЄКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проєкту повинна бути розроблена наступна документація:

1) пояснювальна записка;
2) програма та методика тестування;
3) керівництво користувача;
4) креслення:
   – «Структура бази даних. ER-діаграма»;
   – «Функціональність веб-додатка. UML-діаграма прецедентів».

6. ЕТАПИ ПРОЄКТУВАННЯ

Вивчення літератури за тематикою роботи……..……16.11.2024

Розроблення та узгодження технічного завдання……...……27.11.2024

Розроблення структури вебзастосунку.……..……...……15.12.2024

Розроблення дизайну сторінок та графічних елементів…….…….05.02.2025

Програмна реалізація вебзастосунку..……..…….....…15.03.2025

Тестування вебзастосунку…….…….....08.04.2025

Підготовка матеріалів текстової частини проєкту……..…….29.04.2025

Підготовка матеріалів графічної частини проєкту…….....……15.05.2025

Оформлення технічної документації проєкту……..……28.05.2025

7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до «Програми та методики тестування».

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Євгенія СУЛЕМА

“___” _____ 2025 р.

ВЕБЗАСТОСУНОК ДЛЯ ІНФОРМАЦІЙНОЇ ПІДТРИМКИ РОБОТИ МЕТЕОСТАНЦІЇ

Пояснювальна записка

ДП.045440-03-81

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Дмитро НОВАК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Софія ПІСОЦЬКА

2025

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ 4
ВСТУП 6
1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ 8
1.1. Аналіз особливостей моніторингу клімату та погодних умов 8
1.2. Обґрунтування ідеї створення вебзастосунку для метеостанції 8
1.3. Аналіз існуючих програмних рішень 11
1.4. Результати аналізу 17
2. АНАЛІЗ МОВ ПРОГРАМУВАННЯ, ТЕХНОЛОГІЙ РОЗРОБЛЕННЯ ВЕБЗАСТОСУНКІВ ТА СИСТЕМ ДЛЯ МЕТЕОСТАНЦІЇ 21
2.1. Переваги вебзастосунка над іншими типами програмного забезпечення у вирішенні поставленої задачі 21
2.2. Вибір мови програмування та технологій для розроблення серверної частини вебзастосунку 24
2.3. Вибір технологій для розроблення клієнтської частини 28
2.4. Вибір СУБД та технологій взаємодії з обраною мовою програмування 32
2.5. Вибір системи для обробки даних з метеостанції 34
3. СТРУКТУРНА ОРГАНІЗАЦІЯ РОЗРОБЛЕНИХ ПРОГРАМНИХ ЗАСОБІВ 37
3.1. Загальна структура програмного забезпечення 37
3.2. Структура бази даних 42
3.3. Модуль взаємодії із системою для отримання даних з метеостанції та їх обробки 46
3.4. Модуль для організації робочого процесу користувача 48
4. АНАЛІЗ РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 50
4.1. Особливості реалізації програмного забезпечення 50
4.2. Тестування вебзастосунку 55

4.3. Рекомендації щодо використання розробленого програмного забезпечення 58
4.4. Рекомендації щодо подальшого вдосконалення вебзастосунку..... 59
ВИСНОВКИ..... 63
СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ..... 65
ДОДАТКИ..... 67

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

MVC (Model-View-Controller) – архітектурний шаблон, що розділяє програму на три основні компоненти: Модель (дані), Представлення (інтерфейс) і Контролер (логіка обробки запитів).

RESTful API – інтерфейс програмування додатків, який використовує HTTP-запити для доступу і маніпуляцій ресурсами.

Серверна частина (Backend) – частина вебзастосунку, що відповідає за логіку, обробку запитів, роботу з базами даних і взаємодію з API.

Клієнтська частина (Frontend) – частина вебзастосунку, з якою безпосередньо взаємодіє користувач через інтерфейс.

Node.js – середовище виконання JavaScript на стороні сервера.

Express.js – вебфреймворк для Node.js, який спрощує створення серверних додатків.

MongoDB – документоорієнтована NoSQL база даних, що зберігає дані у форматі BSON (подібному до JSON).

Mongoose – об'єктно-документний мапер (ODM) для MongoDB, який дозволяє працювати з даними як з об'єктами JavaScript.

CRUD – абревіатура для операцій створення (Create), читання (Read), оновлення (Update) та видалення (Delete) даних.

JWT (JSON Web Token) – стандарт для безпечного обміну інформацією у форматі JSON між сторонами через токени.

bcrypt – бібліотека для хешування паролів з метою безпечного зберігання в базі даних.

Nodemailer – модуль для Node.js, що дозволяє надсилати електронні листи через SMTP.

Рівень подання (Presentation Layer) – рівень, який відповідає за обробку HTTP-запитів і відповіді у форматі JSON.

Рівень фасадів (Facade Layer) – рівень, що відповідає за підготовку даних до передачі між клієнтом і сервером, приховуючи чутливу інформацію.

Рівень бізнес-логіки (Service Layer) – рівень, що містить основну логіку роботи застосунку, обробку даних і інтеграцію з зовнішніми сервісами.

Рівень зберігання та доступу до даних (Persistence Layer) – рівень, що відповідає за зберігання, вилучення і обробку даних у базі даних.

DAO (Data Access Object) – шаблон проєктування, який інкапсулює логіку доступу до бази даних.

fs (File System) – модуль Node.js для роботи з файловою системою: читання, запис, видалення файлів.

React – JavaScript бібліотека для створення користувацьких інтерфейсів, особливо односторінкових додатків (SPA).

SPA (Single Page Application) – тип вебзастосунку, що завантажує одну HTML-сторінку та динамічно оновлює контент без перезавантаження.

Action – об’єкт, який описує дію користувача у клієнтській частині застосунку.

Middleware – функція, що обробляє дії (Actions) до того, як вони потрапляють до редюсера або стору.

Dispatcher – центральний механізм, що керує передачею об'єктів Action до відповідних обробників.

State (стан) – структура, що зберігає поточну інформацію додатка для визначення його зовнішнього вигляду та поведінки.

ВСТУП

У сучасному світі кліматичні зміни стають дедалі помітнішими, спричиняючи серйозні виклики для багатьох сфер життєдіяльності людини. Часті аномальні погодні явища, підвищення середньорічної температури, зростання кількості природних катастроф та непередбачуваність погодних умов впливають не лише на екологічну ситуацію, а й на економіку, інфраструктуру, сільське господарство, логістику та безпеку. У цих умовах особливого значення набуває доступ до актуальної, достовірної та структурованої метеорологічної інформації.

Традиційні метеорологічні сервіси часто не забезпечують достатньої гнучкості у подачі даних, мають складний або перевантажений інтерфейс, а також не завжди дозволяють персоналізувати інформацію відповідно до потреб конкретного користувача чи галузі. Водночас стрімкий розвиток цифрових технологій, зокрема вебтехнологій та API-сервісів, створює умови для розробки ефективних рішень, що дозволяють здійснювати повноцінний моніторинг погодних умов у режимі реального часу та проводити аналітичні дослідження кліматичних змін.

У цьому контексті виникає ідея створення вебзастосунку MeteoHub – універсального цифрового інструменту, орієнтованого на різні категорії користувачів: аграріїв, науковців, логістів, працівників енергетичної галузі, туристичних агентств тощо. Застосунок дозволяє не лише переглядати поточні погодні показники, а й аналізувати історичні кліматичні тренди, будувати графіки, карти, отримувати попередження про екстремальні явища, а також адаптувати інформацію до конкретного регіону та потреб користувача.

Актуальність теми дипломної роботи зумовлена необхідністю впровадження інноваційних інструментів для екологічного моніторингу, підвищення рівня кліматичної обізнаності населення та підтримки прийняття рішень у галузях, чутливих до погодних умов. Розробка

MeteoHub демонструє можливості сучасних вебтехнологій у сфері кліматичного аналізу та стає вагомим внеском у розвиток екологічно орієнтованих ІТ-рішень.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

У сучасних умовах розвитку цифрових технологій та зростання потреби в оперативному аналізі кліматичних даних особливого значення набувають системи моніторингу погодних умов. Ефективне функціонування таких систем дозволяє не лише фіксувати зміни в атмосфері, а й своєчасно попереджати про можливі природні загрози. У цьому розділі розглянуто основні особливості кліматичного моніторингу, а також проаналізовано існуючі рішення.

1.1. Аналіз особливостей моніторингу клімату та погодних умов

Сучасні кліматичні зміни, викликані глобальним потеплінням, урбанізацією та людською діяльністю, значно впливають на повсякденне життя, аграрний сектор, інфраструктуру та економіку. Постійне зростання частоти екстремальних погодних явищ (буревії, повені, аномальні температури) підвищує актуальність моніторингу метеорологічної інформації в режимі реального часу.

Моніторинг погодних умов базується на зборі, аналізі та візуалізації даних із метеостанцій, супутників, сенсорних пристроїв та відкритих API. Такі дані використовуються для оперативного прийняття рішень у сферах транспорту, енергетики, сільського господарства, туризму та безпеки. Зважаючи на ці особливості, особливо важливою є зручна подача інформації, інтерфейс взаємодії користувача та можливість персоналізації даних.

1.2. Обґрунтування ідеї створення вебзастосунку для метеостанції

У сучасних умовах зміни клімату та зростаючої залежності багатьох галузей від погодних умов, ефективний доступ до актуальної метеорологічної інформації стає критично важливим. Хоча на ринку вже існує чимало метеосервісів, більшість із них мають складні або

перевантажені інтерфейси, не враховують локальні кліматичні особливості та обмежують можливості користувача у візуалізації й аналізі даних.

MeteoHub створюється як інноваційний вебзастосунок, який поєднує доступність, гнучкість та аналітичну потужність. Його головна мета – надати користувачам з різних галузей зручний інструмент для моніторингу погодних умов, аналізу кліматичних змін і своєчасного реагування на потенційні загрози.

Застосунок стане у пригоді фахівцям з таких сфер, як:

1) наукові дослідження (аналіз кліматичних трендів, вивчення змін у довгостроковій перспективі);
2) транспорт та логістика (прогнозування погодних умов для маршрутів);
3) енергетика (планування навантаження на енергосистеми залежно від погодних факторів);
4) сільське господарство (моніторинг ризиків для врожаю);
5) система безпеки та реагування на надзвичайні ситуації.

MeteoHub забезпечить:

1) інтуїтивно зрозумілий і адаптивний інтерфейс;
2) можливість вибору регіону та періоду для аналізу;
3) інтеграцію з відкритими метео- та супутниковими API;
4) побудову графіків, порівняння даних за різні роки;
5) карту погодних показників у реальному часі;
6) систему попереджень про екстремальні погодні явища.

Таким чином, MeteoHub є не просто черговим погодним застосунком, а універсальною платформою для прийняття обґрунтованих рішень у різних сферах, де клімат і погода відіграють ключову роль. Завдяки інтеграції з різними джерелами даних і гнучкому інтерфейсу, система надає користувачам актуальну інформацію у зручному форматі. Це сприяє підвищенню ефективності планування та мінімізації ризиків, пов'язаних із погодними умовами.

1.2.1. Необхідність створення системи метеомоніторингу аграрія

У цьому контексті виникає потреба у створенні сучасного цифрового інструменту – MeteoHub, який забезпечує комплексний доступ до актуальної та історичної метеорологічної інформації, особливо важливої для аграрного сектору. Агрономічний прогноз погоди є надзвичайно важливим елементом точного землеробства, адже він дозволяє не лише відстежувати погодні умови в реальному часі, а й аналізувати їхній вплив на розвиток культур, ухвалювати обґрунтовані рішення щодо агротехнічних заходів.

Застосунок MeteoHub дозволяє користувачам переглядати метеодані у вигляді інтерактивних температурних карт, супутникових знімків, вітрових мап та радарів, отримувати персоналізовані сповіщення про погіршення погодних умов, порівнювати максимальні та мінімальні показники за різні роки, а також аналізувати довгострокові кліматичні тренди. Особлива увага приділяється агрономічним показникам: температура повітря та рівень опадів, хмарність, вологість, напрям і сила вітру, що безпосередньо впливають на умови проведення польових робіт та врожайність.

Користувачі, зокрема аграрії, можуть створити персональний обліковий запис, що дозволяє отримувати локальні прогнози, адаптовані до специфіки певного регіону та типу вирощуваних культур. Завдяки цим функціям MeteoHub сприяє підвищенню ефективності сільськогосподарських операцій, захисту врожаю від екстремальних погодних умов, а також мінімізації ризиків, пов’язаних з кліматичними змінами.

Отже, створення MeteoHub як інноваційного IT-продукту є обґрунтованим та своєчасним кроком, що відповідає сучасним викликам, забезпечує гнучке управління аграрними процесами та задовольняє потреби як фахівців у сільському господарстві, так і ширшого кола користувачів, які залежать від точних погодних прогнозів.

1.2.2. Використання вебзастосунку для наукових досліджень клімату та його змін

Одна з ключових переваг вебзастосунку MeteoHub – це його потенціал для підтримки наукових досліджень у галузі кліматології, географії, екології та суміжних дисциплін. В умовах глобального потепління та зростаючої кількості кліматичних аномалій доступ до точних, структурованих і наочно представлених даних є критично важливим для аналізу довгострокових змін.

MeteoHub надає можливість дослідникам:

1) отримувати погодні дані в реальному часі;
2) аналізувати історичні кліматичні тренди на основі відкритих джерел;
3) порівнювати показники за різні періоди, регіони чи типи кліматичних умов;
4) будувати графіки та теплові карти, які спрощують виявлення аномалій.

Таким чином, MeteoHub не лише підвищує обізнаність громадськості щодо стану довкілля, а й виступає ефективним цифровим інструментом для наукової спільноти, яка досліджує кліматичні процеси та їхній вплив на екосистеми та суспільство.

1.3. Аналіз існуючих програмних рішень

Для підтвердження цінності та унікальності розробки вебзастосунку Meteohub в рамках дипломного проєкту було проведено аналіз наявних метеорологічних сервісів, які частково або повністю вирішують завдання інформування користувачів про погодні умови, кліматичні зміни та штормові попередження в реальному часі.

До досліджуваних систем були висунуті наступні ключові критерії оцінювання: зручність та інтуїтивність інтерфейсу користувача, повнота та актуальність метеоданих, наявність інтерактивної карти погодних процесів,

можливість інтеграції з іншими сервісами, наявність системи сповіщень про критичні зміни клімату.

У межах аналізу були розглянуті такі рішення, як вебзастосунок Windy, AccuWeather та вебплатформа Weather.com. Кожне з наведених рішень має як переваги, так і недоліки, а детальне вивчення їх функціональних можливостей дозволяє обґрунтувати вибрані архітектурні та дизайнерські рішення при створенні власного вебдодатку Meteohub. Особливу увагу було приділено інтерфейсу користувача, швидкості оновлення даних та інтерактивності елементів. Отримані висновки стали основою для формування унікального підходу до відображення метеорологічної інформації.

1.3.1. AccuWeather

AccuWeather – це популярний вебзастосунок та мобільний застосунок для отримання детального прогнозу погоди. Платформа пропонує метеорологічну інформацію в режимі реального часу, включаючи температуру повітря, ймовірність опадів, швидкість вітру, індекс ультрафіолетового випромінювання, відчутну температуру та інші погодні показники. Дані надаються з високою точністю завдяки використанню власних алгоритмів та співпраці з метеослужбами по всьому світу.

Головна перевага AccuWeather – інтуїтивно зрозумілий інтерфейс та широкий спектр прогнозів: від погодних умов на наступні 15 днів до погодинного прогнозу на поточний день. Крім того, користувачі можуть переглядати супутникові карти, а також отримувати сповіщення про екстремальні погодні умови, що може бути корисним у повсякденному житті або плануванні активностей на рис. 1.1. Завдяки високій точності та функціональності, застосунок широко використовується як звичайними користувачами, так і професіоналами в різних галузях. Його надійність та постійне оновлення інформації роблять AccuWeather одним із лідерів.

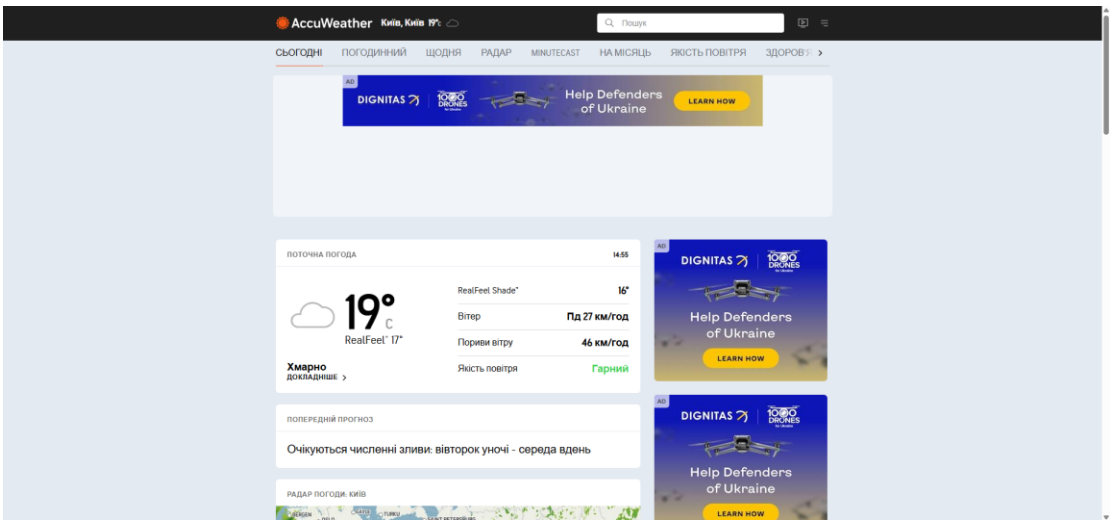


Рис. 1.1. Головна сторінка AccuWeather

Основні особливості AccuWeather:

1. Глобальне покриття. Платформа надає прогнози для більш ніж 3 мільйонів локацій по всьому світу, включаючи дрібні населені пункти.
2. Деталізований прогноз. Користувач може переглядати погодинний прогноз, щоденний прогноз на 15 днів вперед, а також поточні умови з оновленням кожні 15 хвилин.
3. Метеорологічні карти. Доступні інтерактивні карти, які демонструють опади, температуру, вітер, хмарність та інші погодні явища.
4. Сповіщення про негоду. AccuWeather надсилає попередження про наближення штормів, спеки, морозів чи інших потенційно небезпечних погодних змін.
5. "RealFeel" температура. Унікальний показник від AccuWeather, що враховує вологість, вітер та інші фактори, щоб дати реалістичну оцінку того, як погода відчувається насправді.
6. Локалізованість. Дані адаптовані до місця розташування користувача, завдяки GPS та ручному вибору локації.

Незважаючи на широкий функціонал, AccuWeather має певні обмеження. Платформа не надає аналітики змін погоди у динаміці, таких як

порівняння погодних умов з попередніми роками або глибокий аналіз кліматичних змін у конкретному регіоні. Також AccuWeather не дозволяє користувачам інтегрувати власні дані для побудови персоналізованих аналітичних звітів.

У порівнянні з іншими аналізованими програмними рішеннями, AccuWeather є інформативним погодним сервісом, однак його застосування в освітніх або дослідницьких цілях обмежене, оскільки він орієнтований на споживача, а не на інтерактивну або глибоку роботу з метеоданими.

1.3.2. Windy

Windy (також відомий як Windy.com) – це багатофункціональна вебплатформа та мобільний застосунок для візуалізації погодних умов, що надає доступ до великого обсягу метеоданих у реальному часі. Застосунок призначений як для звичайних користувачів, так і для професіоналів у сфері метеорології, авіації, мореплавства, туризму та спорту.

Головною особливістю Windy є інтерактивна мапа як на рис. 1.2, що дозволяє візуалізувати численні погодні параметри – такі як вітер, опади, хмарність, температура, тиск, вологість, а також індекс UV, глибину снігу, висоту хвиль тощо. Платформа використовує дані з провідних метеомоделей, що дозволяє користувачам обирати джерело прогнозу.

Рис. 1.2. Головна сторінка Windy.com

Ключові особливості Windy:

1. Інтерактивна карта в реальному часі. Дані оновлюються кожні 1–3 години, що забезпечує актуальність і точність відображення.
2. Багатошаровість. Користувач може одночасно переглядати кілька параметрів погоди, використовуючи шарову навігацію (наприклад, вітер + температура + опади).
3. Різноманіття метеомоделей. Можливість порівнювати прогнози з різних моделей дозволяє краще оцінити надійність передбачення.
4. Професійна інформація. Доступні такі дані, як аерологічні профілі, радіолокаційні знімки, струменеві потоки та метеозведення для пілотів (METAR, TAF).
5. Прогноз у динаміці. Користувачі можуть переглядати погодні умови в динаміці, переміщаючи часову шкалу вперед-назад.
6. Можливість створення користувацьких точок. Можна зберігати власні локації та отримувати повідомлення про зміну погодних умов.

Windy має значну перевагу у візуалізації та точності прогнозів, проте функціонал аналітики минулих погодних змін обмежений. Платформа не містить порівняння поточних умов з аналогічними даними минулих років, а також не проводить автоматизований аналіз змін клімату у довготривалій перспективі. Крім того, для глибшого аналізу або завантаження погодних даних для подальшої обробки потрібне використання сторонніх сервісів або API-доступу.

У контексті дипломного проєкту Windy є корисним прикладом професійного інструменту для відображення метеоінформації, однак його освітнє застосування потребує додаткових засобів для аналізу та порівняння даних, що обмежує його використання у завданнях кліматичного моніторингу та дослідження погодних змін у часі.

1.3.3. Weather.com

Weather.com – це офіційний вебсайт американської компанії The Weather Channel, одного з найбільших світових постачальників метеоінформації. Платформа пропонує як детальні прогнози погоди, так і супутні матеріали на тему клімату, стихійних явищ та навколишнього середовища. Вона доступна у вебверсії як на рис. 1.3 та як мобільний застосунок (The Weather Channel App).

Основні функції Weather.com:

1. Прогноз погоди на різні часові інтервали. Користувач може переглянути погодні умови на сьогодні, найближчі години або дні, включаючи денну та нічну температури, опади, вологість, силу та напрям вітру.
2. Анімовані радарні карти. Дозволяють відстежувати динаміку опадів, хмарності, температури та іншого в реальному часі.
3. Новини та відеоогляди. Платформа активно публікує матеріали про погодні події, стихійні лиха, зміни клімату, що надає інформаційну складову користувачам.
4. Попередження про небезпечні погодні явища. Автоматичні сповіщення про штормові умови, урагани, спеки чи морози.
5. Локалізація. Автоматичне визначення місця користувача або можливість додати кілька локацій для моніторингу.

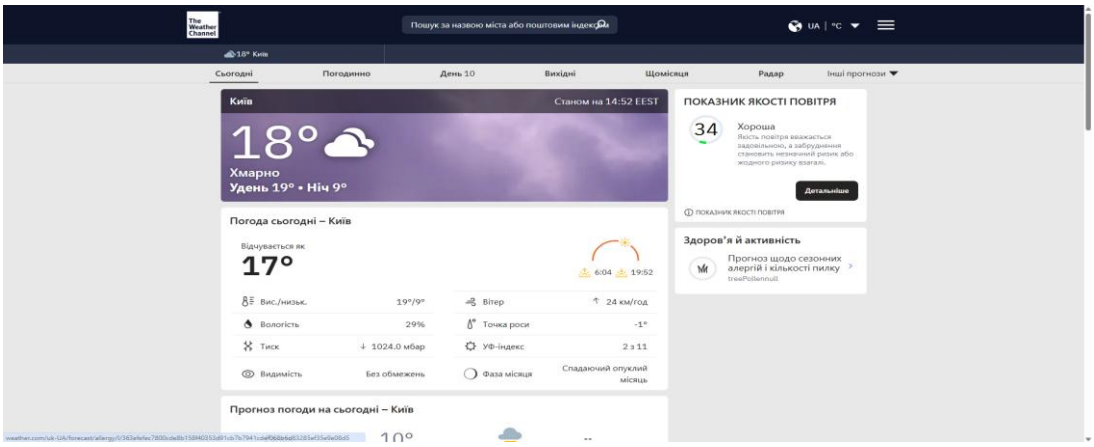


Рис. 1.3. Головна сторінка Weather.com

Попри зручність та велику кількість інформації, Weather.com не забезпечує повноцінного інструментарію для порівняння погодних умов між роками, а також не надає детального аналізу кліматичних змін у довготривалій перспективі. Аналітичні можливості обмежені – користувач не може, наприклад, автоматично зіставити температуру цього дня з аналогічною датою попереднього року або отримати графіки змін клімату за десятиріччя.

Таким чином, у контексті дипломного проєкту Weather.com виступає як наочний приклад зручного метеосервісу з широкою аудиторією, однак його функціональність не охоплює глибоку кліматичну аналітику, яка є важливою для дослідницьких або освітніх цілей, пов’язаних з вивченням змін погоди у часі.

1.4. Результати аналізу

Щоб об'єктивно оцінити переваги та недоліки існуючих метеорологічних сервісів, було сформовано перелік ключових функціональних характеристик, які мають значення для цільових користувачів. Нижче подано пояснення кожного з критеріїв, використаних у табл. 1.1:

1. Поточні погодні умови – надання актуальної інформації про температуру, вологість, вітер, опади, хмарність тощо.
2. Локалізація інформації – можливість отримання детальної погоди саме для конкретного регіону або навіть району.
3. Персоналізовані сповіщення про негоду – функція автоматичного інформування користувача про небезпечні погодні явища, з урахуванням його місцезнаходження.
4. Інтерактивна карта – наявність карти з відображенням метеоданих у динаміці (рух вітру, фронтів, опадів тощо).

5. Науково-аналітична складова – можливість використання сервісу для аналітичних досліджень, доступ до моделей, трендів, агрегованих кліматичних даних.
6. Історичні кліматичні зміни – надання даних про зміни погоди у попередні роки.
7. Порівняння з минулими роками – можливість користувача порівнювати погодні умови за аналогічні періоди різних років.
8. Спеціалізація на певних сферах – наявність функцій, орієнтованих на потреби аграріїв, туристів, енергетиків, дослідників тощо.
9. Простий та зрозумілий інтерфейс – оцінка інтуїтивності дизайну та зручності навігації для користувача.
10. Можливість реєстрації для особистого досвіду – наявність функціоналу облікового запису для збереження налаштувань, улюблених локацій, отримання рекомендацій.
11. Відкритість (відкриті API) – можливість використання зовнішніми розробниками відкритого програмного інтерфейсу для інтеграції даних у власні системи.
12. Візуалізація (графіки) – здатність сервісу наочно відображати інформацію у вигляді графіків, діаграм, анімацій тощо.

Таблиця 1.1

Порівняльна характеристика існуючих рішень

Критерій	Аналоги		
	AccuWeather	Windy	Weather.com
Поточні погодні умови	+	+	+
Локалізація інформації	–	+	+

Продовження табл. 1.1

Персоналізовані сповіщення про негоду	+	–	–
Інтерактивна карта	+	+	–
Науково-аналітична складова	–	–	–
Історичні кліматичні зміни	–	–	–
Порівняння з минулими роками	–	–	–
Спеціалізація на певних сферах	–	+	–
Простий та зрозумілий інтуїтиво інтерфейс	–	–	+
Можливість реєстрації для особистого досвіду	–	+	–
Відкритість (відкриті API)	–	–	–
Візуалізація (графіки)	–	+	+

AccuWeather є одним із найпопулярніших метеорологічних сервісів, що пропонує детальну інформацію про поточні погодні умови та персоналізовані сповіщення про негоду. Основна перевага платформи – висока точність прогнозів і швидке оновлення даних. Проте сервіс не забезпечує достатньої гнучкості у візуалізації інформації, не має аналітичної складової та не підтримує функцій для дослідницьких або професійних потреб. Інтерфейс може бути перевантаженим для звичайного користувача, а відсутність відкритих API обмежує інтеграцію з іншими системами.

Windy орієнтований на користувачів із глибшими знаннями в метеорології, зокрема пілотів, моряків та науковців. Він вирізняється високим рівнем візуалізації, детальними інтерактивними картами та можливістю відстеження численних атмосферних параметрів у режимі реального часу. Водночас відсутність простоти в інтерфейсі, бракує персоналізованих сповіщень та історичних даних, що обмежує його зручність для широкого загалу та довгострокового аналізу.

Weather.com надає користувачу доступ до базової метеоінформації у зручному форматі. Його інтерфейс є одним з найінтуїтивніших серед усіх розглянутих платформ. Проте функціональні можливості сервісу доволі обмежені: він не підтримує інтерактивні карти, аналітичні функції, персоналізацію повідомлень чи спеціалізацію для галузей. Основна мета платформи – надати швидкий доступ до стандартного прогнозу погоди, що не задовольняє вимоги професіоналів чи дослідників.

Загальний висновок: Порівняльний аналіз показує, що жоден із існуючих сервісів не охоплює одночасно всіх потреб, які сьогодні постають перед різними категоріями користувачів – від пересічних громадян до науковців, працівників агросфери, енергетики та логістики. Існує чіткий розрив між сервісами, орієнтованими на простоту використання, і тими, що пропонують багатофункціональні, але складні інтерфейси.

Цей розрив створює потребу у створенні універсальної платформи, що поєднувала б зручність користування з глибоким функціоналом. Такий підхід дозволив би задовольнити як потреби пересічних користувачів, так і вимоги професіоналів. Особливо важливою є можливість адаптації платформи до конкретних сценаріїв використання та доступ до відкритих даних для інтеграції. Саме тому розробка нових рішень у сфері метеомоніторингу є актуальним напрямом сучасних інформаційних технологій.

2. АНАЛІЗ МОВ ПРОГРАМУВАННЯ, ТЕХНОЛОГІЙ РОЗРОБЛЕННЯ ВЕБЗАСТОСУНКІВ ТА СИСТЕМ ДЛЯ ІНФОРМАЦІЙНОЇ ПІДТРИМКИ МЕТЕОСТАНЦІЇ

2.1. Переваги вебзастосунка над іншими типами програмного забезпечення у вирішенні поставленої задачі

Сучасне програмне забезпечення створюється для багатьох типів електронних пристроїв, які можуть суттєво відрізнятися за потужністю, розмірами, функціональністю тощо. Це вимагає унікального підходу до створення додатків залежно від платформи. Найбільш поширеними формами програм є: настільні (десктопні), мобільні та вебзастосунки. Розглянемо плюси та мінуси кожного з них.

2.1.1. Десктопний додаток

Десктопний додаток – це програми, які інсталюються безпосередньо на комп’ютери чи ноутбуки та орієнтовані на конкретну операційну систему.

Переваги десктопних додатків:

1. Не потребують інтернет-з’єднання для функціонування (якщо не залежать від мережевої взаємодії).
2. Вищий рівень безпеки – дані користувача зберігаються локально.
3. Можуть максимально ефективно використовувати ресурси ПК.
4. Користувач самостійно вирішує, чи оновлювати програму.

Недоліки десктопних додатків:

1. Прив’язаність до конкретного пристрою – неможливо легко перенести налаштування та дані.
2. Потреба в інсталяції, що забирає простір на диску.
3. Оновлення та підтримка можуть бути складними.
4. Залежність від конкретного обладнання та ОС.

2.1.2. Мобільний додаток

Мобільний додаток – це такі програми, які працюють на смартфонах або планшетах і мають багато спільного з десктопними, але враховують особливості портативних пристроїв.

Додаткові переваги для мобільних додатків:

1) можливість використання вбудованих функцій пристрою (GPS, камера, NFC тощо);
2) push-сповіщення для взаємодії з користувачем;
3) простота встановлення та оновлення через офіційні маркетплейси.

Додаткові недоліки для мобільних додатків:

1) обмеженість апаратних ресурсів;
2) менші екрани, що ускладнюють відображення великої кількості інформації.

2.1.3. Вебзастосунок

Вебзастосунок – це програми, які працюють через браузер, розміщені на віддалених серверах.

Переваги вебзастосунків:

1. Не потребують інсталяції, не займають місце на диску.
2. Оновлення виконуються на сервері – користувач отримує актуальну версію автоматично.
3. Платформонезалежність – працюють у будь-якому браузері.
4. Доступні з будь-якого пристрою з інтернетом.
5. Масштабованість – обробка даних здійснюється сервером, клієнту не потрібна висока продуктивність.

Недоліки вебзастосуноків:

1. Менший рівень безпеки – користувач не контролює збереження даних.
2. Робота залежить від стабільності інтернет-з’єднання.
3. Необхідність забезпечення сумісності з різними браузерами.

2.1.4. Результати аналізу різних видів додатків

Після детального аналізу переваг і недоліків різних типів програмного забезпечення, для реалізації дипломного проєкту було прийнято рішення створити саме вебзастосунок. Основним чинником такого вибору стало призначення системи – вона має виконувати роль допоміжного інструмента для користувача, що взаємодіє з вебсервісами для розміщення проєктів, які використовують системи керування версіями. Оскільки для повноцінної роботи потрібно постійне підключення до Інтернету, використання десктопного або мобільного застосунка втрачає свої ключові переваги в даному контексті.

Крім того, система аналізуватиме дані з метеостанції, тому зручне візуальне представлення результатів потребує достатньо великого екрана. Разом із тим, важливо, щоб користувач не був обмежений одним пристроєм, а міг працювати з будь-якого місця. Таким чином, вебзастосунок забезпечує необхідну портативність, гнучкість і доступність, що робить його найбільш доцільним варіантом для реалізації поставлених завдань.

2.2. Вибір мови програмування та технологій для розроблення серверної частини вебзастосунку

Серверна частина відповідає за обробку запитів від клієнтів, виконання логіки програми та взаємодію з базою даних. Для реалізації цієї частини існує велика кількість мов та архітектур. Розглянемо три основні варіанти: Java**,** Python і JavaScript. Кожна з цих мов має свої особливості, переваги та сфери найкращого застосування. Вибір конкретної мови програмування залежить від вимог проєкту, досвіду розробника та очікуваного рівня продуктивності системи.

2.2.1. Java

Java – це одна з провідних мов для створення масштабованих вебрішень. Особливо зручним для розробки є фреймворк Spring Boot.

Ключові особливості Java:

1. Об'єктно-орієнтована структура – дозволяє створювати модульні рішення.
2. Платформонезалежність – компіляція в байт-код, що виконується на будь-якому пристрої з JVM.
3. Надійність – завдяки суворій перевірці на етапах компіляції й виконання.
4. Високий рівень безпеки – контроль доступу до ресурсів.
5. Висока продуктивність – байт-код транслюється у машинний код під час виконання.
6. Підтримка багатопотоковості – ефективна робота з паралельними процесами.
7. Динамічність – базується на роботі з бібліотеками, що робить мову легко адаптованою до мінливого середовища.

Переваги:

1) висока продуктивність та стабільність;
2) потужні інструменти безпеки та роботи з базами даних;
3) підходить для великих проєктів із високими вимогами до надійності.

Недоліки:

1) більш складна крива навчання;
2) високі вимоги до ресурсів під час розгортання;
3) більший обсяг коду в порівнянні з динамічними мовами.

2.2.2. Python

Python – високорівнева мова програмування загального призначення, яка широко застосовується для веброзробки, аналізу даних, машинного навчання та автоматизації процесів. У веброзробці найчастіше використовуються фреймворки Django та Flask.

Мова програмування Python має наступні характеристики:

1. Простота та читабельність коду. Синтаксис Python нагадує псевдокод, що спрощує його вивчення й підтримку.
2. Інтерпретована мова. Python не потребує компіляції, що дозволяє швидше тестувати та змінювати код.
3. Велика стандартна бібліотека. Python включає модулі для роботи з HTTP, JSON, базами даних, потоками, файлами та ін.
4. Кросплатформеність. Python-програми можуть запускатися на Windows, Linux, Mac без модифікацій.
5. Розширюваність. Python дозволяє інтегрувати модулі, написані на C або C++ для підвищення продуктивності.
6. Спільнота та підтримка. Завдяки великій кількості розробників існує безліч відкритих рішень та документації.

Переваги:

1) швидкий старт та розробка MVP (мінімально життєздатного продукту);
2) добре підходить для задач аналізу кліматичних даних, роботи з API, побудови графіків;
3) Django забезпечує повноцінний набір інструментів для створення безпечного вебзастосунку.

Недоліки:

1) менша продуктивність у порівнянні з компільованими мовами;
2) Flask потребує додаткових налаштувань для функціональності;
3) Python менш оптимізований для високонавантажених систем у порівнянні з Java або Node.js.

2.2.3. JavaScript

JavaScript – мова програмування, яка спочатку створювалася для роботи на стороні клієнта, але з появою Node.js стала повноцінним

інструментом і для розробки серверної частини. Node.js забезпечує асинхронне неблокуюче середовище виконання JavaScript на сервері.

Мова програмування JavaScript з Node.js має наступні характеристики:

1. Асинхронність. Завдяки подієво-орієнтованій моделі Node.js дозволяє обробляти велику кількість одночасних з'єднань без блокування потоку.
2. Виконання на стороні сервера. JavaScript виконується на базі V8-движка (розробленого Google), що забезпечує високу швидкість виконання коду.
3. Легка масштабованість. Node.js підходить як для малих проєктів, так і для масштабованих мікросервісних архітектур.
4. Велика кількість пакетів. Через npm (Node Package Manager) доступні тисячі бібліотек для будь-яких задач – від роботи з БД до вебсерверів.
5. Full-stack розробка. Можливість використовувати одну мову як на фронтенді, так і на бекенді значно спрощує розробку та комунікацію між частинами застосунку.
6. Підтримка REST API та WebSockets. JavaScript легко реалізує API для клієнтів та підключення в реальному часі.

Переваги:

1) висока продуктивність у сценаріях з великою кількістю паралельних запитів;
2) єдина мова на клієнті та сервері;
3) швидка розробка та велика екосистема.

Недоліки:

1) відсутність строгих типів, що може призвести до помилок (вирішується використанням TypeScript);
2) для великих проєктів можуть бути потрібні додаткові інструменти для структурування коду.

2.2.4. Результати аналізу

Після порівняння можливих варіантів розробки серверної частини, з урахуванням вимог проєкту метеостанції (обробка даних у реальному часі, взаємодія з API, зручність розробки, інтеграція з клієнтською частиною), оптимальним вибором є JavaScript з використанням Node.js.

Це рішення дозволяє використовувати одну мову на всіх рівнях застосунку (full-stack JavaScript), забезпечує високу продуктивність при обробці великої кількості запитів, а також має потужну екосистему для розробки REST API, підключення до бази даних, та обробки метеоданих у режимі реального часу.

Таким чином, Node.js стане основною технологією для серверної частини вебзастосунку метеостанції.

2.3. Вибір технологій для розроблення клієнтської частини

Клієнтська частина вебзастосунку відповідає за взаємодію користувача із системою, зокрема за візуалізацію та поведінку інтерфейсу. Вона визначає, як виглядатиме вебсайт, чи буде він статичним або динамічним, і як елементи на сторінці реагуватимуть на дії користувача. Основу побудови клієнтської частини складають такі технології, як HTML**,** CSS та JavaScript.

HTML (HyperText Markup Language) є мовою розмітки, що використовується для структурування вебсторінок. Її основними елементами є теги, які визначають різні частини сторінки: блоки тексту, таблиці, форми, зображення тощо.

CSS (Cascading Style Sheets) дозволяє задавати зовнішній вигляд HTML-елементів. За допомогою цієї мови опису стилів можливо реалізувати адаптивність до розміру екрана, налаштувати кольори, шрифти, відступи та інші параметри дизайну. Якщо HTML задає “що” відображати, то CSS – “як” це виглядає.

Для полегшення створення інтерфейсу широко використовуються UI-фреймворки – набори готових компонентів і стилів, що базуються на HTML, CSS і JavaScript. До популярних прикладів належать Bootstrap, Bulma, Foundation, Materialize та інші. Вони допомагають створити адаптивний інтерфейс швидше, ніж писати все з нуля.

У розробці сучасних вебзастосунків ключову роль відіграють JavaScript-фреймворки, що забезпечують розширену функціональність і кращу інтерактивність. Серед найпопулярніших рішень варто розглянути Angular, React та Vue.js.

2.3.1. Angular

Angular – це JavaScript-фреймворк, що базується на концепції MVVM (Model-View-ViewModel) і створений для розробки масштабованих вебдодатків з високим рівнем інтерактивності. Він надає велику кількість інструментів для організації архітектури проєкту та підтримує розробку багатофункціональних інтерфейсів з високим рівнем реактивності.

Переваги Angular:

1) підтримка створення вебкомпонентів та npm-бібліотек через CLI;
2) одностороння прив’язка даних підвищує стабільність додатку і знижує ризик помилок;
3) вбудована система впровадження залежностей і модульна архітектура спрощують масштабування;
4) добре структурована архітектура дозволяє ефективно керувати складними проєктами.

Недоліки Angular:

1) високий поріг входження через велику кількість структурних елементів;
2) порівняно нижча продуктивність у деяких випадках.

2.3.2. React

React – це бібліотека JavaScript, що орієнтована на побудову односторінкових додатків різного рівня складності. Вона була розроблена Facebook і широко використовується у комерційних проєктах.

Переваги React:

1) простота у вивченні завдяки використанню JSX – синтаксису, подібного до HTML;
2) можна зосередитися на JavaScript, не заглиблюючись у специфіку самого фреймворку;
3) висока швидкість роботи завдяки Virtual DOM;
4) одностороння прив’язка даних забезпечує передбачувану поведінку;
5) Redux – зручна бібліотека для керування станом;
6) підтримка функціонального стилю програмування та створення повторно використовуваних компонентів;
7) зручність оновлення версій завдяки скриптам для автоматичної міграції.

Недоліки React:

Незвичний підхід до поєднання логіки з розміткою через JSX може викликати труднощі у новачків.

2.3.3. Vue.js

Vue.js – це JavaScript-фреймворк для створення інтерактивних інтерфейсів користувача. Його можна легко інтегрувати у вже існуючі проєкти або використовувати для повноцінної розробки з нуля.

Переваги Vue.js:

1) підтримка шаблонів, що розширюють можливості HTML;
2) схожість з Angular і React спрощує перехід для розробників з інших фреймворків;

3) висока інтеграційна здатність – компоненти Vue легко впроваджуються у вже готові системи;
4) добре масштабується, дозволяючи створювати багаторазово використовувані шаблони.

Недоліки Vue.js:

1) порівняно менша кількість освітніх ресурсів і прикладів;
2) надмірна гнучкість може створити труднощі при роботі над великими проєктами.

2.3.4. Результати аналізу технологій для розроблення клієнтської частини вебзастосунку

Після аналізу доступних рішень було обрано набір інструментів, який оптимально підходить для реалізації клієнтської частини розроблюваного вебзастосунку.

HTML і CSS залишаються обов'язковими технологіями для побудови структури та стилізації вебсторінки.

Серед UI-фреймворків обрано Bootstrap 4, який забезпечує зручність у створенні адаптивного дизайну завдяки готовим компонентам і підтримці мобільної верстки.

Зважаючи на необхідність створення односторінкового застосунку, було прийнято рішення використовувати React як основний JavaScript-фреймворк. Він дозволяє швидко розпочати розробку, не потребує складної конфігурації, як Angular, та забезпечує високу продуктивність.

2.4. Вибір СУБД та технологій взаємодії з обраною мовою програмування

У процесі розробки інформаційної системи важливим етапом є вибір системи управління базами даних (СУБД) та технологій взаємодії з нею. Цей вибір безпосередньо впливає на продуктивність, масштабованість, гнучкість і надійність програмного забезпечення. Крім того, правильне

поєднання СУБД з мовою програмування забезпечує ефективну обробку та зберігання даних, спрощує розробку та подальшу підтримку проєкту. У цьому підрозділі розглянуто особливості різних типів баз даних, переваги конкретних СУБД та принципи їх інтеграції з мовами програмування.

2.4.1. Відмінності між базами даних SQL та NoSQL

SQL-бази даних застосовують структуровану мову запитів для маніпуляцій з даними та їх структурування, що забезпечує високу ефективність при роботі з великими обсягами даних. Однак вони потребують заздалегідь визначеної схеми, і будь-яка її зміна може вплинути на всю систему.

У свою чергу, NoSQL-бази дозволяють працювати з даними, які не мають фіксованої структури. Вони підтримують різні моделі зберігання – документоорієнтовану, ключ-значення, графову та колонкову. Завдяки цьому можливо додавати нові поля до записів без необхідності змінювати всю схему.

2.4.2. MySQL

Основні плюси та можливості MySQL:

1. Зрілість технології – MySQL є широко вживаною базою, що пройшла випробування часом і довела свою стабільність у численних реальних застосуваннях. Повна підтримка ACID властивостей.
2. Підтримка ACID – гарантує надійність і цілісність даних.
3. Можливість реплікації – дані можуть дублюватися на кількох серверах, що покращує масштабованість, відмовостійкість і доступність.
4. Шардування – хоча не всі SQL-бази підтримують цю функцію, в MySQL можливо розподіляти дані по кількох вузлах.

2.4.3. PostgreSQL

PostgreSQL – це потужна система, яка поєднує можливості SQL та NoSQL. Серед її особливостей:

1. Потужна транзакційна система – надає гнучкі механізми контролю транзакцій.
2. Об'єктно-реляційна модель – платформа підтримує як класичні реляційні зв’язки, так і елементи об’єктного підходу.
3. Повна підтримка ACID – забезпечує надійне зберігання та обробку даних.
4. Стандартний SQL – PostgreSQL дотримується міжнародних стандартів мови SQL, що спрощує перехід між системами.

2.4.4. MongoDB

MongoDB – це документоорієнтована NoSQL база даних, яка зберігає дані у форматі BSON (розширений JSON). Вона дозволяє гнучко працювати з неструктурованими даними, підтримує масштабування, реплікацію та високу продуктивність.

Нижче наведено деякі переваги та сильні сторони MongoDB:

1. Гнучка структура даних – можна змінювати схему на льоту без необхідності змінювати вже наявні записи.
2. Автономність в адмініструванні – система потребує мінімального втручання з боку адміністратора.
3. Висока продуктивність – особливо при виконанні простих запитів та частих оновленнях.
4. Вбудоване масштабування – підтримка реплікації та шардування реалізована «з коробки».

2.4.5. Результати аналізу

За результатами аналізу систем управління базами даних MySQL, PostgreSQL та MongoDB, найбільш гнучким варіантом виявилася MongoDB.

Вона підтримує динамічну схему даних, що дає змогу змінювати структуру акаунтів без необхідності повного редизайну бази даних.

Окрім цього, MongoDB демонструє високу швидкодію при частих оновленнях, добре масштабується завдяки вбудованим механізмам реплікації та шардування, не вимагає складного адміністрування й ефективно інтегрується з JavaScript/Node.js, що робить її зручною для сучасних вебзастосунків.

2.5. Вибір системи для обробки даних з метеостанції

Для отримання, обробки та візуалізації даних з метеостанції передбачається використання готового хмарного рішення. У рамках розробки вебзастосунку особливо важливо забезпечити сумісність між апаратною частиною метеостанції та обраним сервісом, який має підтримувати телеметрію у відповідному форматі. Крім технічної інтеграції, слід враховувати наявність REST API, можливості візуалізації та масштабованість для подальшого розширення функціоналу.

Саме тому вибір відповідної платформи є критично важливим етапом у реалізації цього проєкту. У процесі дослідження було обрано три популярні сервіси, які найчастіше використовуються в подібних рішеннях: OpenWeatherApi, Weather.com та Meteostat.

2.5.1. OpenWeatherApi

OpenWeatherApi – це популярна платформа, яка надає доступ до великого масиву метеорологічних даних: поточної погоди, прогнозів, історичних даних та погодних попереджень.

Основні переваги:

1. Різноманітність джерел даних – OpenWeatherApi збирає інформацію з супутників, наземних станцій та інших відкритих джерел.

2. Зручний REST API – легко інтегрується з вебзастосунками та мобільними програмами.
3. Документація та приклади – добре описаний API із прикладами запитів на різних мовах програмування.

Однак OpenWeatherApi орієнтований на отримання даних зі сторонніх джерел, а не з власної метеостанції. Він не призначений для прийому телеметрії з IoT-пристроїв і не забезпечує обробку чи візуалізацію особистих метеоданих у режимі реального часу.

2.5.2. Weather.com

Weather.com (The Weather Channel API) також надає розширений доступ до погодних характеристик. Основні характеристики:

1. Надійність – платформа використовується провідними новинними та метеорологічними службами.
2. Висока точність прогнозів – дані обробляються з використанням моделей штучного інтелекту.
3. Додаткові сервіси – включає карти, історичну інформацію та погодні попередження.

Проте, як і OpenWeatherApi, Weather.com є зовнішнім джерелом даних. Він не підтримує прямого надсилання інформації з пристроїв користувача, що є критично важливим для реалізації власної метеостанції з обробкою телеметрії.

2.5.3. Meteostat

Meteostat – це потужна відкрита платформа для доступу до історичних і поточних метеорологічних даних. Вона надає доступ до широкого спектра погодної інформації, включаючи температуру, опади, вітер та інші параметри, зібрані з тисяч метеостанцій по всьому світу. Основні переваги:

1. Велика база даних – охоплення історичних погодних даних з 1900х років до сьогодення.

2. Зручне API та бібліотеки – підтримка мов програмування Python, R та інших для швидкої інтеграції.
3. Точність – дані надходять з офіційних джерел, таких як національні метеослужби.
4. Безкоштовний доступ – відкриті ліцензії для наукових і освітніх проєктів.
5. Масштабованість – придатний як для локальних, так і глобальних аналізів кліматичних змін.

Meteostat є ідеальним інструментом для проєктів, пов'язаних з аналізом та візуалізацією погодних даних, особливо коли не потрібне пряме підключення до фізичних сенсорів.

2.5.4. Результати аналізу

Порівняння трьох платформ – OpenWeather API, Weather.com API та Meteostat – показало, що жодна з них окремо не здатна повністю задовольнити вимоги розроблюваного вебзастосунку. Кожна система має свої сильні сторони, але й певні обмеження, що унеможливлює її використання як єдиного джерела метеоінформації.

Однак функціонал прогнозування погоди, побудови супутникових карт, відображення опадів на радарах та визначення зон посухи у Meteostat обмежений або відсутній. Для реалізації цих функцій доцільно застосовувати гібридне рішення – поєднання Meteostat з API сервісів OpenWeather та Weather.com.

Таким чином, комбіноване використання трьох платформ забезпечує гнучкість, масштабованість і високу точність системи, роблячи її ефективним інструментом як для локального моніторингу, так і для аналізу глобальних кліматичних змін. Такий підхід є оптимальним для сучасних IoT-рішень у сфері метеоспостережень, роблячи систему придатною як для локального моніторингу, так і для аналізу глобальних кліматичних тенденцій.

3. СТРУКТУРНА ОРГАНІЗАЦІЯ РОЗРОБЛЕНИХ ПРОГРАМНИХ ЗАСОБІВ

У цьому розділі розглянуто внутрішню будову програмного забезпечення, розробленого в межах дипломного проєкту. Наведено загальну архітектуру вебзастосунку, описано основні компоненти системи та їхню взаємодію. Особливу увагу приділено розмежуванню клієнтської та серверної частин. Такий підхід забезпечує масштабованість, гнучкість та спрощує подальший супровід застосунку.

3.1. Загальна структура програмного забезпечення

Програмне забезпечення дипломного проєкту реалізоване у вигляді вебзастосунку з використанням архітектурного шаблону MVC. Програмну систему можна розділити на дві частини:

1. Серверна частина, яка містить в собі всю бізнес-логіку, відповідальна за збереження та обробку даних.
2. Клієнтська частина, що представляє зовнішній вигляд вебзастосунка та дає користувачу можливість повноцінно користуватися розроблюваним програмним забезпеченням.

Основні елементи даних частин та зв’язки між ними зображено на структурній схемі вебзастосунка (Додаток 1. Архітектура вебзастосунку). Клієнтська частина взаємодіє із серверною через інтерфейс RESTful API. Такий підхід разом з обраними технологіями дозволяє розробляти елементи програмної системи незалежно один від одного. Клієнтській частині не потрібно знати особливості реалізації серверної частини, а серверній – клієнтської. Для цього попередньо формується API – адреса (URL), метод та тіло запиту, яке приймає або повертає сервер у певному форматі.

Пропонується окремо більш детально розглянути структуру серверної та клієнтської частини вебзастосунку.

3.1.1. Структура серверної частини вебзастосунку

Серверна частина реалізована з використанням Node.js та виконує наступні функції:

1. Реалізує процедуру реєстрації та авторизації користувачів.
2. Отримує та обробляє запити від клієнтської частини, повертаючи відповідь у форматі JSON.
3. Взаємодіє з базою даних, виконуючи операції створення, зчитування, оновлення та видалення (CRUD) для сутностей системи, таких як користувачі, записи про погодні умови, геолокації (детальніше в підрозділі 3.2).
4. Працює з зовнішніми API для отримання актуальних погодних даних (наприклад, Meteostat або інші подібні сервіси).
5. Реалізує кешування та регулярне оновлення погодних даних для зменшення навантаження на зовнішні сервіси та забезпечення швидкої відповіді клієнту.
6. Має можливість розсилки сповіщень (наприклад, про надзвичайні погодні явища) користувачам, які підписались на певні геолокації.

Як було описано вище, вебзастосунок реалізовано за архітектурним шаблоном MVC. Серверна частина у цій структурі виконує роль Моделі (M) та Контролера (C), реалізуючи основну логіку та обробку запитів. Для побудови серверної частини було обрано багаторівневу архітектуру, що забезпечує розділення обов’язків між компонентами та полегшує тестування й масштабування системи.

Архітектура серверної частини вебзастосунку MeteoHub реалізована відповідно до багаторівневої моделі, що дозволяє чітко розмежувати відповідальність компонентів системи. Серверна частина побудована з використанням фреймворку Node.js у поєднанні з Express.js та базою даних MongoDB через Mongoose ORM.

Рівень подання (Presentation Layer). На цьому рівні розміщені роути (маршрутизатори), які приймають HTTP-запити від клієнтської частини.

Вони обробляють запити, виконують базову валідацію вхідних даних та передають їх до відповідних сервісів. Також на цьому рівні формується структура відповіді, яку отримує клієнт.

Основні завдання: обробка запитів (реєстрація, авторизація, надсилання листів, отримання/видалення користувачів тощо); валідація вхідних даних; формування відповідей у форматі JSON. Ключові модулі цього рівня: /routes/auth.js – обробка запитів для реєстрації, входу та роботи з профілем; /routes/admin.js – запити, доступні адміністратору: видалення користувачів, зміна плану, розсилки тощо.

Рівень фасадів (Facade Layer). У проєкті цей рівень частково реалізовано шляхом: передачі даних між клієнтом і сервером у вигляді об’єктів (тіло запиту); перетворення моделей з бази даних перед відправкою (наприклад, через .select('-password') або .toJSON() у схемі). Такі дії дозволяють не розкривати чутливі дані (наприклад, паролі) при передачі на клієнт.

Рівень бізнес-логіки (Service Layer). Цей рівень реалізує логіку всього додатку: автентифікація користувачів, генерація JWT-токена; хешування паролів за допомогою bcrypt; зміна профілю користувача, включаючи завантаження аватара; розсилка email-повідомлень з використанням nodemailer; підключення до Meteostat для обробки кліматичних даних.

Рівень зберігання та доступу до даних (Persistence Layer). Цей рівень реалізований через: Mongoose-схеми (/models/User.js), що визначають структуру та правила зберігання даних у MongoDB; виклики CRUD-операцій (find, save, findByIdAndDelete, findOne) для взаємодії з базою; роботу з файловою системою сервера – збереження та видалення аватарів користувачів через fs; email-модуль (/utils/email.js) – взаємодія з зовнішнім SMTP-сервером; підключення до API Meteostat для зчитування кліматичних даних. Структура серверу представлена на рис. 3.1.

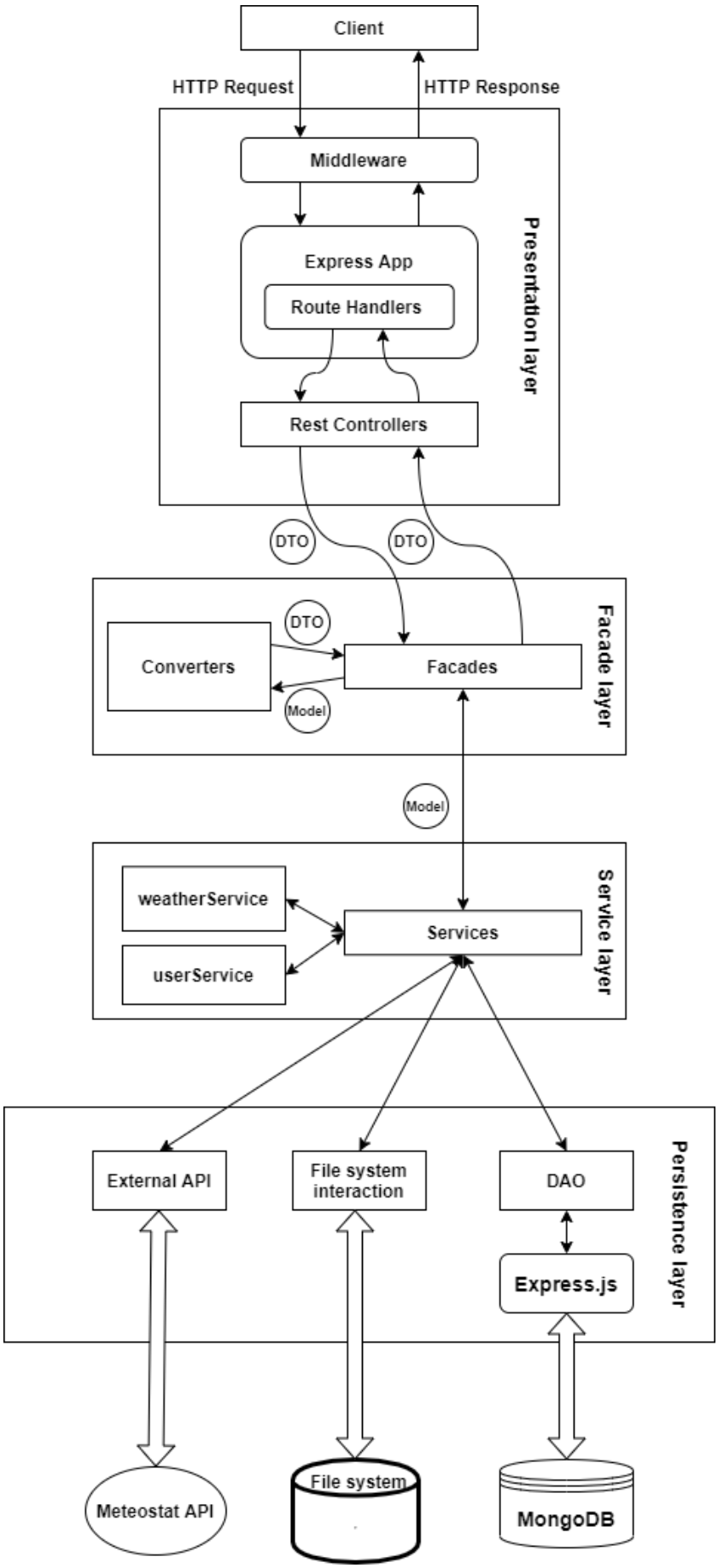


Рис. 3.1. Структура серверної частини вебзастосунку

3.1.2. Архітектура клієнтської частини вебзастосунку

Клієнтська частина відповідає за наступні функції:

1) створення інтерфейсу, який дозволяє користувачеві взаємодіяти із системою;
2) надсилання запитів до серверу з метою оновлення вмісту вебсторінки;
3) адаптивне відображення інтерфейсу на пристроях із різною роздільною здатністю.

У шаблоні MVC клієнтська частина виконує роль представлення (View). Для реалізації було обрано фреймворк React, що дає змогу створювати односторінкові вебзастосунки (SPA), де єдиний HTML-документ є основою, а взаємодія з користувачем здійснюється за рахунок динамічного завантаження HTML, CSS та JavaScript. Завдяки цьому вебсторінка не перезавантажується повністю при зміні контенту, що покращує користувацький досвід.

Клієнтська архітектура включає наступні основні елементи (рис. 3.2):

1. Actions – об'єкти, які зберігають інформацію про дії користувача (наприклад, кліки, введення тексту) з деталями про тип події, її джерело та наслідки.
2. Middlewares – проміжні програмні модулі, які обробляють Actions перед їх передачею до Reducers. Вони можуть здійснювати логування, обробку помилок, виконання асинхронних запитів до серверу або генерувати нові дії.
3. Dispatcher – центральний механізм, через який усі дії надходять до обробки в додатку.
4. State – набір змінних JavaScript, що визначає поточний стан інтерфейсу і даних, які бачить користувач.
5. Store – компонент, що зберігає та керує станом додатка.
6. Reducers – функції, які отримують Actions і відповідно до логіки змінюють State у Store.

7. View – React-компоненти, що містять структуру сторінки та її динамічне оновлення на основі даних зі State.

Структура клієнта на рис. 3.2.

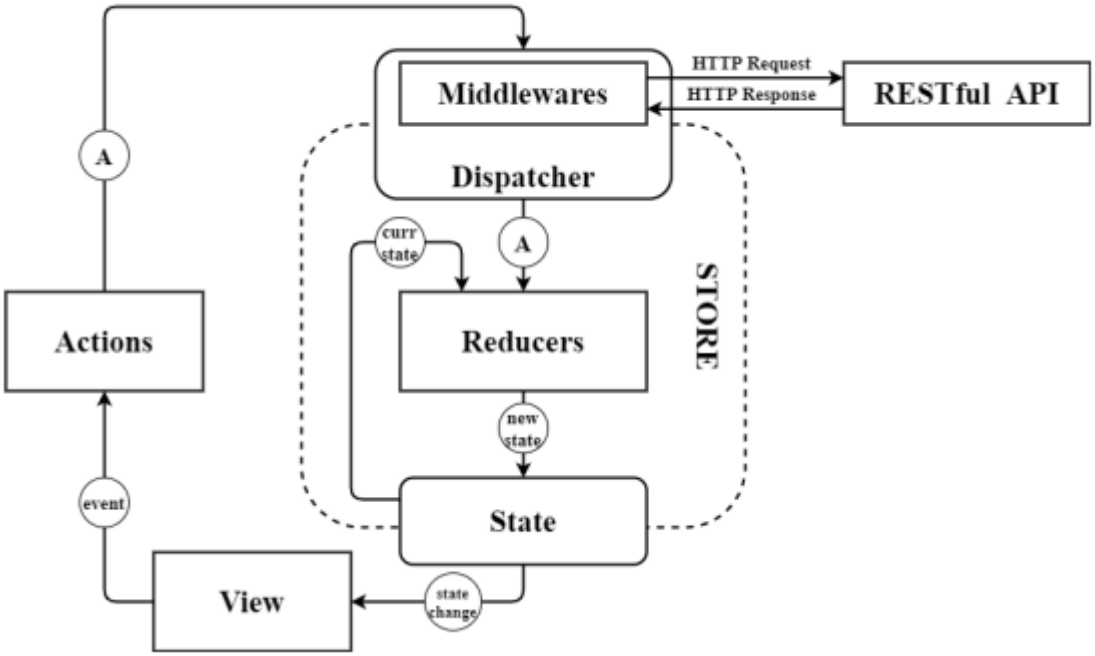


Рис. 3.2. Структура клієнтської частини вебзастосунку

Уся логіка та розмітка клієнтської частини реалізована в React-компонентах. Для їх створення використовується підхід "Container/Component", який дозволяє розділити обов’язки між компонентами. Це спрощує розробку та покращує організацію коду. Такий підхід також забезпечує кращу повторне використання компонентів та полегшує їх тестування. Структура включає:

1) Container – компонент, що відповідає за зберігання стану та методи його зміни;
2) Component – компонент, який формує інтерфейс на основі отриманих даних.

3.2. Структура бази даних

Для реалізації вебзастосунку було обрано базу даних MongoDB, яка забезпечує гнучкість зберігання даних, масштабованість, а також легку інтеграцію з JavaScript/Node.js стеком технологій. Структура бази даних була спроєктована з урахуванням особливостей MongoDB, забезпечуючи ефективний доступ до потрібної інформації. У схемі бази даних передбачено наступні основні колекції:

Користувачі (users). Структура сутності представлена у табл. 3.1.

Таблиця 3.1

Структура сутності користувач

Назва поля	Тип даних	Опис
_id	ObjectId	Унікальний ідентифікатор користувача (генерується MongoDB автоматично)
firstname	String	Ім’я користувача
lastname	String	Прізвище користувача
email	String	Електронна адреса користувача (унікальна), використовується для авторизації
password	String	Хешований пароль користувача (захищений через bcrypt)
phone	String	Контактний номер телефону
plan	String	Тип підписки: free / premium
role	String	Роль користувача в системі: user / manager
__v	Number	Версія документа, яку додає Mongoose автоматично для контролю змін

Reports (звіти). Ця колекція зберігає автоматично згенеровані звіти як у табл. 3.2.

Таблиця 3.2

Структура сутності звіти

Назва поля	Тип даних	Опис
_id	ObjectId	Унікальний ідентифікатор користувача (генерується MongoDB автоматично)
generated	Date	Дата та час створення звіту (формат: YYYY-MM-DD HH:MM:SS)
stations	String	Ідентифікатор погодної станції
data	Array[Object]	Масив об’єктів з погодними показниками по місяцях. Кожен об’єкт містить дату, температуру, опади та інші параметри
userId	ObjectId	Ідентифікатор користувача, який згенерував звіт

Структура одного об'єкта в масиві data у табл. 3.3.

Таблиця 3.3

Структура сутності data

Поле	Тип	Опис
_id	ObjectId	Унікальний ідентифікатор користувача (генерується MongoDB автоматично)
date	Date	Дата показників (YYYY-MM-DD HH:MM:SS)

Продовження табл. 3.3

tavg	Number	Середня температура (°C)
tmin	Number	Мінімальна температура (°C)
tmax	Number	Максимальна температура (°C)
prcp	Number	Кількість опадів (мм)
snow	Number	Рівень снігу (мм або см)
wdir	Number	Напрямок вітру
wspd	Number	Швидкість вітру
pres	Number	Атмосферний тиск
tsun	Number	Тривалість сонячного сяйва (хвилини)

Приклад JSON-запису одного report подано у лістингу 3.1.

Лістинг 3.1. JSON-запис report

```
{
  "generated": "2025-05-01 19:03:20",
  "stations": ["33345", "UKKM0", "33347", "33466"],
  "data": [
    {
      "date": "2010-01-01 00:00:00",
      "tavg": -8.8,
      "tmin": -11.4,
      "tmax": -6.1,
      "prcp": 54
    },
    {
      "date": "2010-02-01 00:00:00",
      "tavg": -3.3,
      "tmin": -5.1,
```

```
      "tmax": -0.9,
      "prcp": 62
    }
// ...
  ],
  "userId": "6612ea0e4f3f2d36549875aa"
}
```

Підписники розсилки новин (Newsletter Users). Ця колекція зберігає інформацію про користувачів, які оформили підписку на сповіщення про негоду, і представлена у табл. 3.4.

Таблиця 3.4

Структура сутності підписника розсилки новин

Назва поля	Тип даних	Опис
_id	ObjectId	Унікальний ідентифікатор документа, автоматично створюється MongoDB
name	String	Ім’я та прізвище користувача, який підписався на розсилку
email	String	Електронна адреса користувача
city	String	Місто проживання користувача, за якою він отримувати сповіщення про негоду

3.3. Модуль взаємодії із системою для отримання даних з метеостанції та їх обробки

Цей модуль відповідає за автоматизовану взаємодію з зовнішнім погодним API з метою отримання метеорологічних даних, їх обробку та передачу в інші частини системи.

Основні функції модуля:

1. Формування запитів до API: на основі вхідних параметрів користувача (наприклад, діапазон дат, геолокація, список станцій) модуль формує запит до відповідного погодного сервісу.
2. Отримання погодних даних: запит виконується через HTTP(S), результат надходить у форматі JSON.
3. Попередня валідація відповіді: перевірка коректності та повноти отриманих даних (наявність ключових полів, відповідність формату).
4. Обробка та нормалізація: модуль обробляє отримані дані – переводить їх у внутрішній формат системи, округлює значення, приводить дати до єдиного формату YYYY-MM-DD HH:MM:SS.
5. Передача даних до модуля генерації звітів: після обробки дані передаються у компонент, де вони можуть бути відображені, проаналізовані або збережені користувачем як звіт.
6. Обробка помилок: у разі некоректної відповіді від API або недоступності сервера модуль генерує відповідне повідомлення для користувача або логування.

Типові параметри запиту:

1) station_id – ідентифікатор метеостанції;
2) start_date, end_date – період, за який збираються дані;
3) metrics – перелік параметрів (температура, опади, тиск тощо).

Типова структура відповіді (масив об'єктів) подана у лістингу 3.2.

Лістинг 3.2. JSON-запис структури відповіді з API

```
{
  "date": "2010-05-01 00:00:00",
  "tavg": 17.3,
  "tmin": 12.4,
  "tmax": 22.8,
  "prcp": 55
}
```

Технічна реалізація:

1. Модуль реалізований на Node.js з використанням бібліотеки axios (або fetch) для здійснення HTTP-запитів.
2. Дані обробляються та перетворюються за допомогою вбудованих JavaScript-функцій та бібліотек типу moment.js для форматування дат.
3. Усі критичні помилки логуються на сервері, а у разі потреби – повідомляються користувачеві через UI.

3.4. Модуль для організації робочого процесу користувача

Цей модуль відповідає за забезпечення інтуїтивної та зручної взаємодії користувача з системою під час роботи з погодними звітами. Він дозволяє ефективно планувати, переглядати та зберігати інформацію, отриману в результаті запитів до API.

Основні функції модуля:

1. Авторизація та аутентифікація користувача: користувач входить у систему за допомогою email і паролю.
2. Формування запитів до погодного API: інтерфейс дозволяє користувачеві вказати параметри, після чого здійснюється запит до стороннього сервісу.
3. Відображення погодних даних у зручному форматі: результати запиту виводяться у вигляді таблиці або графіка.
4. Збереження звіту: користувач може зберегти результати запиту у вигляді звіту. Звіт включає мета-дані та погодні показники.
5. Рольова модель доступу: функціонал може обмежуватись в залежності від ролі користувача.

Компоненти інтерфейсу:

1. Форма створення запиту: вибір метеостанцій, діапазону дат, параметрів.
2. Графічна аналітика: побудова графіків температури, опадів тощо.

3. Сторінка профілю: перегляд та редагування особистих даних.

Технічні особливості:

1. Дані зберігаються в базі даних MongoDB, у колекції reports, зв'язані з користувачем через поле userId.
2. Запити до погодного API виконуються асинхронно та обробляються на сервері Node.js.
3. Інтерфейс побудований з використанням React.
4. Паролі користувачів зберігаються у хешованому вигляді з використанням bcrypt.

Модуль для організації робочого процесу користувача є ключовим елементом системи, що забезпечує зручну взаємодію з погодними даними. Завдяки інтуїтивному інтерфейсу, підтримці рольового доступу та можливості формування і збереження звітів, модуль сприяє ефективному плануванню й аналізу метеоінформації. Використання сучасних технологій, дозволяє досягти високої продуктивності та безпеки роботи з даними.

Крім основних функціональних можливостей, модуль має розширення для забезпечення персоналізації досвіду користувача. Завдяки інтеграції з графічними бібліотеками, такими як Chart.js або Recharts, користувачі отримують візуально привабливе та інформативне представлення інформації, що полегшує прийняття рішень у сферах, де метеоаналіз має критичне значення.

Особливу увагу приділено зручності використання на різних пристроях. Інтерфейс модуля адаптований для мобільних платформ, що дозволяє користувачам отримувати доступ до функціоналу з будь-якого місця та в будь-який час. Крім того, реалізовано систему сповіщень про завершення обробки запиту або виникнення помилок, що підвищує загальну інформативність та надійність системи.

4. АНАЛІЗ РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У цьому розділі наведено детальний опис технічної реалізації вебзастосунку для інформаційної підтримки метеостанції, розробленого в рамках дипломного проєкту. Основна увага приділяється аналізу ключових етапів розробки, способам інтеграції з зовнішніми API та особливостям обробки й візуалізації метеорологічної інформації. Також розглядаються програмні компоненти, що забезпечують безпечну та ефективну взаємодію користувача із системою, включаючи автентифікацію, збереження звітів та рольовий доступ. У подальших підрозділах буде проаналізовано функціонування кожного окремого сервісу, способи обробки даних, а також архітектурні та інтерфейсні рішення.

4.1. Особливості реалізації програмного забезпечення

Під час розробки програмного забезпечення дипломного проєкту було реалізовано стандартну для сучасного вебзастосунку функціональність, яка забезпечує повний цикл взаємодії користувача із системою. Зокрема, було впроваджено реєстрацію користувачів, механізми автентифікації (перевірка особи користувача) та авторизації (надання відповідного рівня доступу), а також реалізовано набір CRUD-операцій для роботи з основними сутностями системи – створення, читання, оновлення та видалення записів. Крім того, було реалізовано інтерфейс акаунт-менеджера, який дозволяє адміністратору керувати профілями користувачів, зокрема редагувати їх підписки або видаляти облікові записи. Для забезпечення ефективної комунікації із користувачами була впроваджена система електронної розсилки, яка дозволяє надсилати важливі оновлення, повідомлення та рекомендації. Однак, ключовою особливістю вебзастосунку Meteohub є складна інтеграція з низкою зовнішніх метеорологічних вебсервісів, що дозволяє формувати цілісну інформаційну картину погодних умов для конкретної локації користувача. Зокрема, сервіс Meteostat забезпечує доступ

до історичних метеоданих на основі даних фізичних метеостанцій, що дає змогу аналізувати погодні тренди за попередні роки. Сервіс OpenWeatherAPI використовується для отримання актуальних погодних умов у реальному часі, включно з температурою, вологістю, атмосферним тиском тощо. За допомогою WeatherAPI реалізовано прогнозування погоди на найближчі дні та отримання сповіщень про потенційні небезпечні погодні явища, як-от бурі, зливи чи сильний вітер. Нарешті, інтеграція з OpenWeatherMap дозволяє візуалізувати погодні карти, знімки супутників, радарні дані та інші графічні елементи, що доповнюють текстову інформацію. Усі ці сервіси взаємодіють у межах єдиної системи, де для кожної введеної користувачем локації визначаються координати, після чого відповідні API-запити повертають комплексну інформацію, яка обробляється, аналізується та відображається у вигляді графіків, таблиць і інтерактивних елементів. Таким чином, Meteohub забезпечує користувача повною, актуальною та наочно поданою метеоінформацією, сприяючи кращому розумінню кліматичних умов у заданому регіоні.

4.1.1. Взаємодія з Meteostat API – історичні спостереження

Мета інтеграції. Meteostat надає ретроспективні кліматичні ряди з реальних фізичних метеостанцій, що дозволяє користувачу бачити, як температура та опади змінювалися протягом багатьох років.

Сценарій роботи:

1. Геокодування координат міста (одноразовий запит до api.openweathermap.org/geo), після чого отримані lat/lon зберігаються у state для подальших викликів.
2. Пошук найближчої метеостанції (/stations/nearby) через RapidAPI-шлюз. Сервіс повертає масив станцій, з якого береться перший елемент як «референсна» станція.
3. Отримання щомісячних агрегатів (/stations/monthly) в діапазоні 2010-01-01 … сьогодні. Запит виконується асинхронно; відповідь

перетворюється на масив об’єктів {date, tavg, prcp} із додатковим форматуванням дат.

4. Візуалізація: у залежності від режиму перегляду (середньомісячна динаміка, порівняння одного місяця різних років, детальний розріз року) формується вибірка для Recharts-графіків. Функція getTemperatureChartData() будує масив точок, а renderChart() проявляє його на фронтенді й дозволяє експортувати PNG-знімок через html2canvas.
5. Окрім цього, реалізовано окрему функціональність для визначення рекордних значень температур (tmin і tmax) у вибраний день. Для цього, при перегляді користувачем прогнозу, програма виконує серію запитів до ендпоінту /stations/daily, підставляючи поточну дату в історичному зрізі. Отримані дані зберігаються у форматі {min, max, year} для подальшого аналізу. За допомогою методу reduce знаходяться абсолютні мінімальні та максимальні значення температури за всі роки, і результат виводиться в інтерфейсі користувача.

Особливості реалізації:

1. Кешування. Щомісячні дані не змінюються ретроспективно, тому результат зберігається в IndexedDB та оновлюється лише при зміні вибору станції.
2. Обмеження API. Meteostat безкоштовно віддає 500 запитів/день, тож передбачено локальний rate-limiter, що проставляє затримку при швидкому перемиканні локацій.
3. Надійність. Усі запити обгорнуті в try/catch; при помилці користувач бачить toast-сповіщення з можливістю повторити запит.

4.1.2. Взаємодія з OpenWeather API – отримання актуальної погоди в реальному часі

OpenWeather API забезпечує оперативний доступ до поточних погодних умов у будь-якому місті світу. Його використання дозволяє відображати температуру, вологість, швидкість вітру, хмарність, атмосферний тиск та інші параметри в реальному часі. Це створює інтерактивний досвід для користувача та доповнює історичну інформацію з Meteostat.

Після завдання назви міста користувачем, на фронтенді виконується запит до ендпоінту api.openweathermap.org/data/2.5/weather, у якому параметри q, units та appid формуються динамічно. У відповідь сервіс повертає об’єкт із поточними даними про погоду: температурою повітря (temp), відчуттям (feels_like), координатами міста (coord.lat і coord.lon), станом неба (weather[0].description), вологістю (humidity) тощо. Отримані координати далі використовуються для запиту до іншого сервісу – Open-Meteo API – для визначення висоти місцевості, а також передаються у функцію, яка ініціює запит на історичні погодні дані (див. розділ 4.1.1.).

Дані оновлюються автоматично при зміні міста, що відслідковується через React-хук useEffect. Кожен запит обгорнутий у try/catch для обробки помилок, а всі асинхронні дії виконуються послідовно. Таке каскадне завантаження забезпечує коректне формування інформації без втрати синхронізації.

OpenWeather дозволяє до 60 викликів на хвилину у безкоштовному тарифі, тому не впроваджувався додатковий rate-limiter, проте передбачена обробка помилок і fallback-механізми при недоступності сервісу.

Поточні погодні умови виводяться поруч із графіками історичних значень, дозволяючи користувачу швидко порівняти актуальну температуру з середніми показниками за попередні роки, а також оцінити контекст. Це посилює аналітичну цінність застосунку.

4.1.3. Взаємодія з WeatherAPI – прогноз погоди на 7 днів

WeatherAPI дозволяє отримувати детальний погодний прогноз на кілька днів уперед. У рамках застосунку сервіс використовується для побудови короткострокових прогнозів тривалістю до тижня, що дає змогу користувачеві заздалегідь планувати свою діяльність та порівнювати майбутню температуру з історичними й поточними значеннями.

Після введення або зміни міста формується запит до ендпоінту https://api.weatherapi.com/v1/forecast.json, з параметрами key, q (назва міста) та days=7. API повертає об'єкт, у якому ключ forecast.forecastday містить масив прогнозів по днях. Кожен елемент включає середню, мінімальну й максимальну температуру, опади, рівень ультрафіолету, напрямок та силу вітру, рівень хмарності, а також текстовий опис очікуваних умов. Ці дані зберігаються у локальному стані через setForecast() і використовуються для виводу на інтерфейс.

Процес виклику API ініціюється в useEffect() після зміни значення міста (CITY). Запит обгорнутий у блок try/catch/finally, що забезпечує надійність роботи застосунку: при помилці вона логуються, а інтерфейс повідомляє користувача про завершення завантаження, навіть якщо запит не вдався. Змінна loading використовується для індикації процесу запиту.

Прогноз на 7 днів відображається у вигляді погодного слайдера або табличного представлення на інтерфейсі. Користувач може візуально оцінити коливання температур, а також подивитись опис умов у кожен день тижня. Прогноз по WeatherAPI доповнює поточні дані з OpenWeather і історичні значення з Meteostat, створюючи цілісну погодну аналітику з трьох часових площин: минуле, сьогодення і майбутнє.

4.1.4. Взаємодія з OpenWeatherMap – геокодування міст і відображення погодних шарів

OpenWeatherMap використовується для перетворення текстової назви міста на географічні координати (широту та довготу). Ця операція є

ключовою для інтеграції з іншими API, зокрема для побудови мап із погодними шарами (Windy), а також для виконання запитів, які потребують координатного формату.

При введенні або зміні значення міста (cityName) у полі пошуку, ініціюється запит до ендпоінту https://api.openweathermap.org/geo/1.0/direct, де передається назва міста та API-ключ. У відповідь OpenWeatherMap повертає масив об'єктів із координатами відповідних населених пунктів. З першого елемента береться lat і lon, які зберігаються у стані (setCoords) для подальшого використання. Запит до геокодування виконується лише при наявності cityName, що захищає від зайвих звернень. Він обгорнутий у try/catch, що дозволяє безпечно обробляти помилки мережі чи некоректний формат відповіді. Використано encodeURIComponent для правильного кодування назв міст із пробілами та спецсимволами.

Після отримання координат формується інтерактивна карта з ресурсу Windy.com. Шари карти перемикаються залежно від вибору користувача: температура, вітер, радар, супутник, індекс УФ або посуха. Для кожного режиму формується відповідне посилання з координатами, що відображається у вбудованому фреймі (iframe).

OpenWeatherMap виконує роль геокодувального модуля, який з'єднує текстові запити користувача з координатно-залежними модулями системи. Таким чином, навіть без введення точних координат, користувач може отримати доступ до шарів погодної інформації.

4.2. Тестування вебзастосунку

Для перевірки працездатності розробленої системи інформаційної підтримки метеостанції було проведено комплексне тестування, що охоплює як клієнтську, так і серверну частину додатка.

Основним методом тестування було обрано ручне тестування, оскільки воно дозволяє змоделювати дії реального користувача та

переконатися у коректності роботи ключових функціональних модулів через інтерфейс.

Крім того, окремі частини додатка було протестовано за методикою "сірого ящику" – з частковим урахуванням внутрішньої логіки системи, що дозволило не лише візуально перевірити правильність результатів, а й оцінити змінені дані у базі даних та на файловій системі.

У процесі тестування були охоплені такі основні аспекти функціональності:

1) авторизація користувачів та обмеження доступу;
2) перегляд даних з метеостанції (температура, вологість, тиск, вітер тощо);
3) візуалізація даних у вигляді графіків і таблиць;
4) оновлення та збереження записів;
5) робота з повідомленнями та сповіщеннями;
6) інтеграція з зовнішніми джерелами (наприклад, API для синхронізації даних з відкритих метеосервісів).

4.2.1. Тестування авторизації користувача

Передумови: у базі даних вже існує запис користувача з логіном та паролем.

Опис тестового випадку №1:

1) користувач відкриває головну сторінку додатка;
2) вводить логін та пароль у відповідні поля;
3) натискає кнопку "Увійти".

Результат тестового випадку №1: система успішно авторизує користувача, відбувається перехід на головну сторінку з доступом до функціональних блоків. У разі неправильних даних – з’являється відповідне повідомлення про помилку.

4.2.2. Тестування відображення та оновлення метеоданих

Передумови: система підключена до джерела даних або емулює його.

Опис тестового випадку №2:

1) користувач відкриває вкладку "Поточні показники";
2) обирає конкретну локацію;
3) оновлює дані вручну або чекає на автоматичне оновлення.

Результат тестового випадку №2: користувач бачить актуальні значення. Значення змінюються відповідно до показників, що надходять з пристрою. У випадку втрати зв’язку з джерелом – система інформує користувача відповідним повідомленням.

4.2.3. Тестування графічного представлення історичних даних

Опис тестового випадку №4:

1) користувач переходить до вкладки "Графіки";
2) обирає параметр (наприклад, температура);
3) задає фільтри;
4) натискає кнопку "Побудувати графік".

Результат тестового випадку №4: на екрані з’являється лінійний графік із точками значень. Графік будується коректно, без помилок, за заданим проміжком часу. Якщо в цей період дані відсутні – система виводить повідомлення "Дані відсутні".

4.2.4. Тестування валідації введених користувачем даних

Опис тестового випадку №5:

1) користувач намагається ввести дані користувача, який вже існує;
2) отримує повідомлення “користувач вже існує”.

Результат тестового випадку №5: система не дозволяє зберегти дані, поки не буде виправлено помилки. Користувачу виводиться повідомлення про некоректний формат або обов’язковість поля.

4.2.5. Тестування стійкості системи до збоїв (edge cases)

Опис тестового випадку №6:

1) вимкнення джерела даних;
2) перевантаження великою кількістю запитів (емуляція 100 запитів за секунду);
3) видалення станції, яка наразі збирає дані.

Результат тестового випадку №6: система повідомляє про відсутність зв'язку або перевантаження. Всі дії логуються. Критичних збоїв або падінь не зафіксовано. При видаленні активної станції – запит підтвердження з попередженням.

Таким чином, у результаті тестування було підтверджено працездатність вебзастосунку в умовах стандартного використання та під час моделювання потенційних помилок. Виявлені незначні недоліки були виправлені. Система продемонструвала стабільну роботу, адекватну реакцію на помилки та зручність для кінцевого користувача.

4.3. Рекомендації щодо використання розробленого програмного забезпечення

Для правильного та ефективного використання розробленого вебзастосунку, користувачам слід дотримуватися кількох важливих рекомендацій:

1. Правильне використання фільтрів та пошукових функцій. При використанні пошукових функцій для фільтрації даних за датами або метеостанціями, варто уважно вказувати точні діапазони значень. Це допоможе уникнути помилок при пошуку або отриманні некоректних результатів.
2. Обмеження на розмір файлів при експорті даних. При експорті даних в форматах CSV або PDF варто пам'ятати, що великі обсяги інформації можуть впливати на швидкість обробки та

завантаження. Для зручності рекомендується розділяти великі масиви даних на декілька менших частин перед експортом.

3. Використання актуальних версій браузерів. Для оптимальної роботи вебзастосунку рекомендується використовувати останні версії браузерів, таких як Google Chrome, Mozilla Firefox або Safari. Це забезпечить сумісність з усіма функціями вебзастосунку та дозволить уникнути можливих проблем із завантаженням або відображенням даних.
4. Робота з графіками та діаграмами. При перегляді графіків та діаграм змін погодних умов варто звертати увагу на точність вибраних діапазонів дат. Рекомендується регулярно оновлювати сторінку для отримання найсвіжіших даних та коректного відображення змін погодних показників.
5. Безпека та конфіденційність. Для забезпечення безпеки та захисту персональних даних користувачам слід регулярно змінювати паролі та не передавати їх стороннім особам. Також не варто зберігати пароль у браузері, щоб уникнути несанкціонованого доступу.
6. Оновлення програмного забезпечення. Для оптимальної роботи вебзастосунку важливо періодично перевіряти наявність оновлень, оскільки нові версії можуть містити виправлення помилок та покращення функціональності.
7. Використання допоміжних інструментів. Якщо виникають складнощі з налаштуванням або використанням деяких функцій додатку, рекомендується звертатися до інструкцій користувача або використовувати функцію онлайн-підтримки для швидкого вирішення проблем.

4.4. Рекомендації щодо подальшого вдосконалення вебзастосунку

Архітектура розробленого вебзастосунку спроєктована з урахуванням можливості розширення та вдосконалення його функціональності в

майбутньому. На даний момент додаток вже виконує основні функції, однак є кілька напрямків для його подальшого розвитку, які дозволять поліпшити ефективність і зручність використання.

4.4.1. Розширення інтеграції з метеорологічними вебсервісами

На даний момент вебзастосунок отримує дані лише з безкоштовних метеорологічних сервісів. Для підвищення точності прогнозів і збільшення доступності даних пропонується додати підтримку додаткових метеорологічних API, таких як WeatherStack або AccuWeather. Це дозволить користувачам вибирати найбільш підходящий сервіс для отримання прогнозів, а також забезпечить резервування у разі недоступності основного сервісу.

Для кожного сервісу слід реалізувати можливість відображення різних параметрів, таких як температура, вологість, швидкість вітру, що дасть користувачам більш широкий вибір для аналізу погодних умов.

4.4.2. Покращення функціональності графічних інтерфейсів

Один із важливих напрямків для вдосконалення – це покращення відображення даних у вигляді графіків і діаграм. Зараз вебзастосунок дозволяє відображати основні показники погоди у вигляді простих графіків, але для більш детального аналізу рекомендується додати інтерактивні можливості, зокрема:

1) можливість масштабування та зміни діапазону часу для більш точного аналізу даних;
2) інтерактивні легенди та анотації на графіках, щоб користувач міг отримати додаткову інформацію по кожному елементу;
3) підтримка різних типів графіків (лінійні, стовпчикові, гістограми).

Це значно покращить користувацький досвід при взаємодії з погодними даними.

4.4.3. Покращення можливостей пошуку та фільтрації даних

Зважаючи на те, що вебзастосунок обробляє велику кількість метеорологічних даних, важливо забезпечити користувачів зручними інструментами для пошуку і фільтрації. Пропонується реалізувати:

1) можливість фільтрувати дані по конкретним регіонам, що дозволить користувачам швидше знаходити потрібну інформацію;
2) вдосконалити пошукову систему за допомогою розширених фільтрів, таких як температура, швидкість вітру, вологість, що дасть можливість відображати лише необхідні дані;
3) додати сортування результатів за різними критеріями (наприклад, за часом, температурою або вітром).

4.4.4. Додавання можливості персоналізації налаштувань

Щоб покращити користувацький досвід, варто додати можливість персоналізації налаштувань для кожного користувача. Це може включати:

1) можливість збереження улюблених місць для швидкого доступу до прогнозу погоди;
2) користувацькі налаштування для відображення даних: вибір одиниць вимірювання (Цельсій/Фаренгейт, м/с/км/год);
3) сповіщення про зміни погодних умов на основі заданих критеріїв (наприклад, досягнення певної температури або зміни напрямку вітру).

Ці покращення дозволять зробити вебзастосунок більш персоналізованим і зручним для різних типів користувачів.

4.4.5. Підтримка мобільних версій додатку

Для забезпечення доступу до вебзастосунку на мобільних пристроях необхідно розробити адаптивний інтерфейс або окремі мобільні додатки для платформ iOS і Android. Це дозволить користувачам зручно переглядати

прогнози погоди без залежності від пристрою, а також забезпечить додаткові функції, такі як сповіщення про зміни погоди в реальному часі.

4.4.6. Покращення ефективності роботи з великими даними

У майбутньому планується розширення можливостей додатка для обробки великих обсягів метеорологічних даних. Для цього можна:

1) впровадити технології кешування та асинхронної обробки даних для покращення часу відгуку;
2) використовувати розподілену обробку даних, щоб забезпечити швидке отримання результатів навіть при великій кількості запитів.

Ці заходи дозволять зберегти ефективність роботи додатку навіть при збільшенні обсягу інформації.

4.4.7. Результати аналізу щодо подальшого вдосконалення

У ході тестування вебзастосунку було виявлено низку аспектів, які можуть бути суттєво покращені для підвищення загальної продуктивності системи, зручності використання та задоволеності користувачів. Аналіз показав, що запропоновані напрямки розвитку відповідають сучасним вимогам.

Розширення інтеграції з метеорологічними сервісами забезпечить точніші та стабільніші джерела даних, що позитивно вплине на достовірність прогнозів. Покращення візуалізації дозволить користувачам краще аналізувати погодні тенденції, а розширені фільтри і пошук зроблять взаємодію з даними більш інтуїтивно зрозумілою. Персоналізація функцій дозволить адаптувати додаток під конкретні потреби кожного користувача, що є важливим фактором у підвищенні лояльності аудиторії.

Таким чином, реалізація зазначених рекомендацій дозволить перетворити вебзастосунок на потужний інструмент для моніторингу та аналізу погодних умов, забезпечуючи високий рівень продуктивності, гнучкості та зручності у використанні.

ВИСНОВКИ

У межах виконання дипломної роботи було успішно розроблено функціональний вебзастосунок Meteohub, який надає користувачам зручний доступ до комплексної метеорологічної інформації в інтерактивному форматі. Основною метою проєкту було створення інформативного, наочного та технічно надійного інструменту, що поєднує дані з кількох зовнішніх API для отримання актуальної, історичної та прогнозної інформації про погоду.

У процесі розробки:

1) було реалізовано повноцінну архітектуру вебзастосунку, що включає автентифікацію, авторизацію, панель адміністратора, базові CRUD-операції та систему управління користувачами;
2) здійснено глибоку інтеграцію з чотирма метеосервісами (Meteostat, OpenWeather, WeatherAPI, Windy.com), що забезпечило багатовекторне охоплення погодних даних;
3) реалізовано візуалізацію кліматичної інформації за допомогою графіків, таблиць та мап, що значно підвищує інформативність системи;
4) впроваджено кешування даних, обробку помилок, обмеження на кількість запитів (rate-limiting) та інші засоби підвищення надійності системи;
5) проведено модульне тестування основних функцій взаємодії з API та візуального інтерфейсу.

Практичне значення роботи полягає у створенні зручного та ефективного засобу аналізу погодних умов, що може бути використаний як у повсякденному житті (планування активностей, подорожей), так і в спеціалізованих галузях (аграрна сфера, логістика, туризм тощо). Застосунок також має потенціал для подальшого розширення – зокрема, інтеграції з мобільними платформами, розширення аналітичного

функціоналу та впровадження машинного навчання для прогнозування.

Таким чином, поставлені в роботі цілі було досягнуто, а реалізоване програмне забезпечення повністю відповідає технічним та функціональним вимогам, демонструючи високий рівень якості реалізації та актуальність обраної теми.

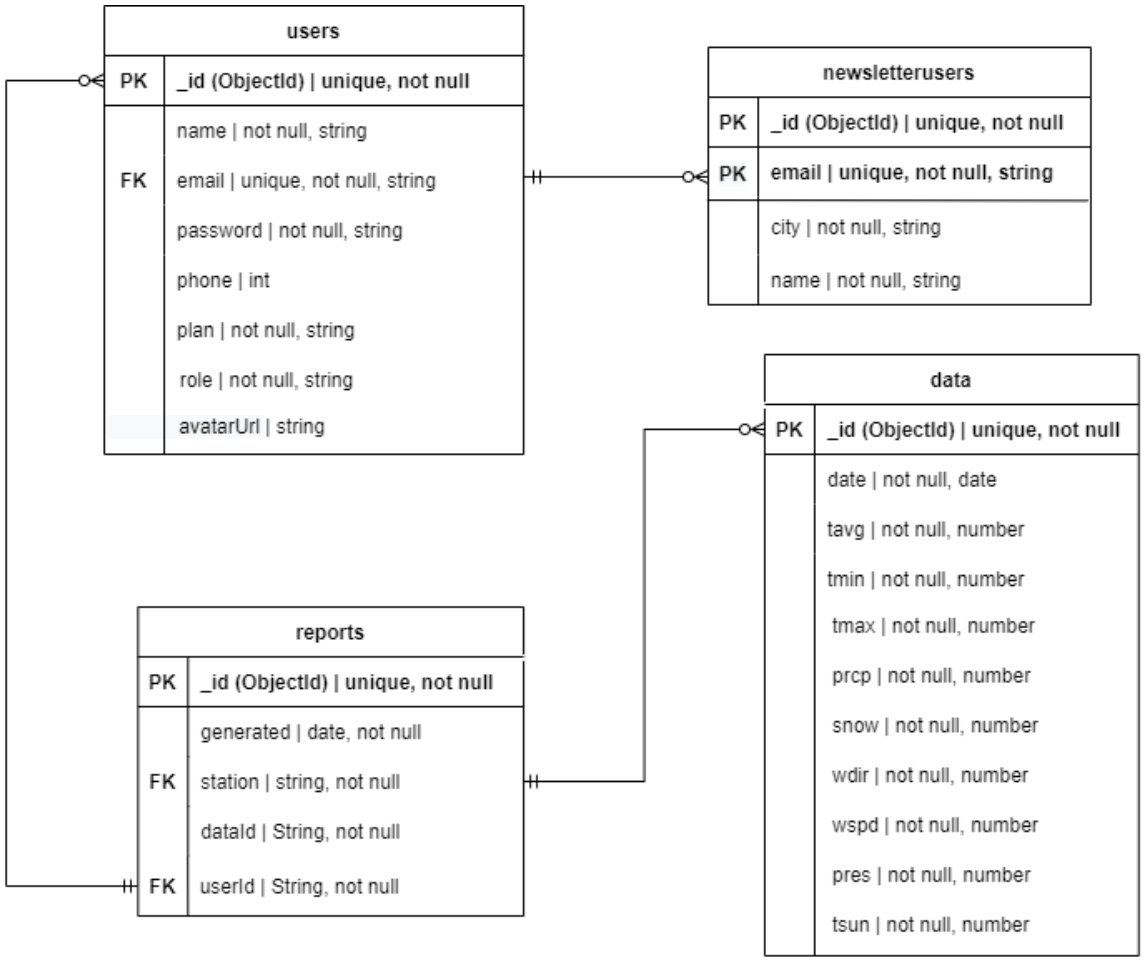
СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. React documentation [Електронний ресурс]. – Режим доступу: https://react.dev.– Дата доступу: 28.03.2025 р.
2. JavaScript documentation [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/uk/docs/Web/JavaScript. – Дата доступу: 30.03.2025 р.
3. Node.js documentation [Електронний ресурс]. – Режим доступу: https://nodejs.org/en/docs. – Дата доступу: 30.03.2025 р.
4. HTML & CSS documentation [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/uk/docs/Web. – Дата доступу: 30.03.2025 р.
5. OpenWeather API documentation [Електронний ресурс]. – Режим доступу: https://openweathermap.org. – Дата доступу: 30.03.2025 р.
6. Meteostat API documentation [Електронний ресурс]. – Режим доступу: https://dev.meteostat.net. – Дата доступу: 30.03.2025 р.
7. WeatherAPI documentation [Електронний ресурс]. – Режим доступу: https://www.weatherapi.com. – Дата доступу: 30.03.2025 р.
8. Windy API documentation [Електронний ресурс]. – Режим доступу: https://api.windy.com. – Дата доступу: 30.03.2025 р.
9. Open-Meteo API documentation [Електронний ресурс]. – Режим доступу: https://open-meteo.com. – Дата доступу: 30.03.2025 р.
10. Node-cron documentation [Електронний ресурс]. – Режим доступу: https://www.npmjs.com/package/node-cron. – Дата доступу: 20.04.2025 р.
11. Jsonwebtoken documentation [Електронний ресурс]. – Режим доступу: https://www.npmjs.com/package/jsonwebtoken. – Дата доступу: 20.04.2025 р.
12. Mongoose documentation [Електронний ресурс]. – Режим доступу: https://mongoosejs.com. – Дата доступу: 20.04.2025 р.

13. Bcrypt documentation [Електронний ресурс]. – Режим доступу: https://www.npmjs.com/bcrypt. – Дата доступу: 20.04.2025 р.
14. Express.js documentation [Електронний ресурс]. – Режим доступу: https://expressjs.com. – Дата доступу: 20.04.2025 р.
15. Multer documentation [Електронний ресурс]. – Режим доступу: https://www.npmjs.com/multer. – Дата доступу: 23.04.2025 р.
16. Node.js File System (fs) documentation [Електронний ресурс]. – Режим доступу: https://nodejs.org/api/fs.html. – Дата доступу: 23.04.2025 р.
17. Nodemailer documentation [Електронний ресурс]. – Режим доступу: https://nodemailer.com. – Дата доступу: 23.04.2025 р.

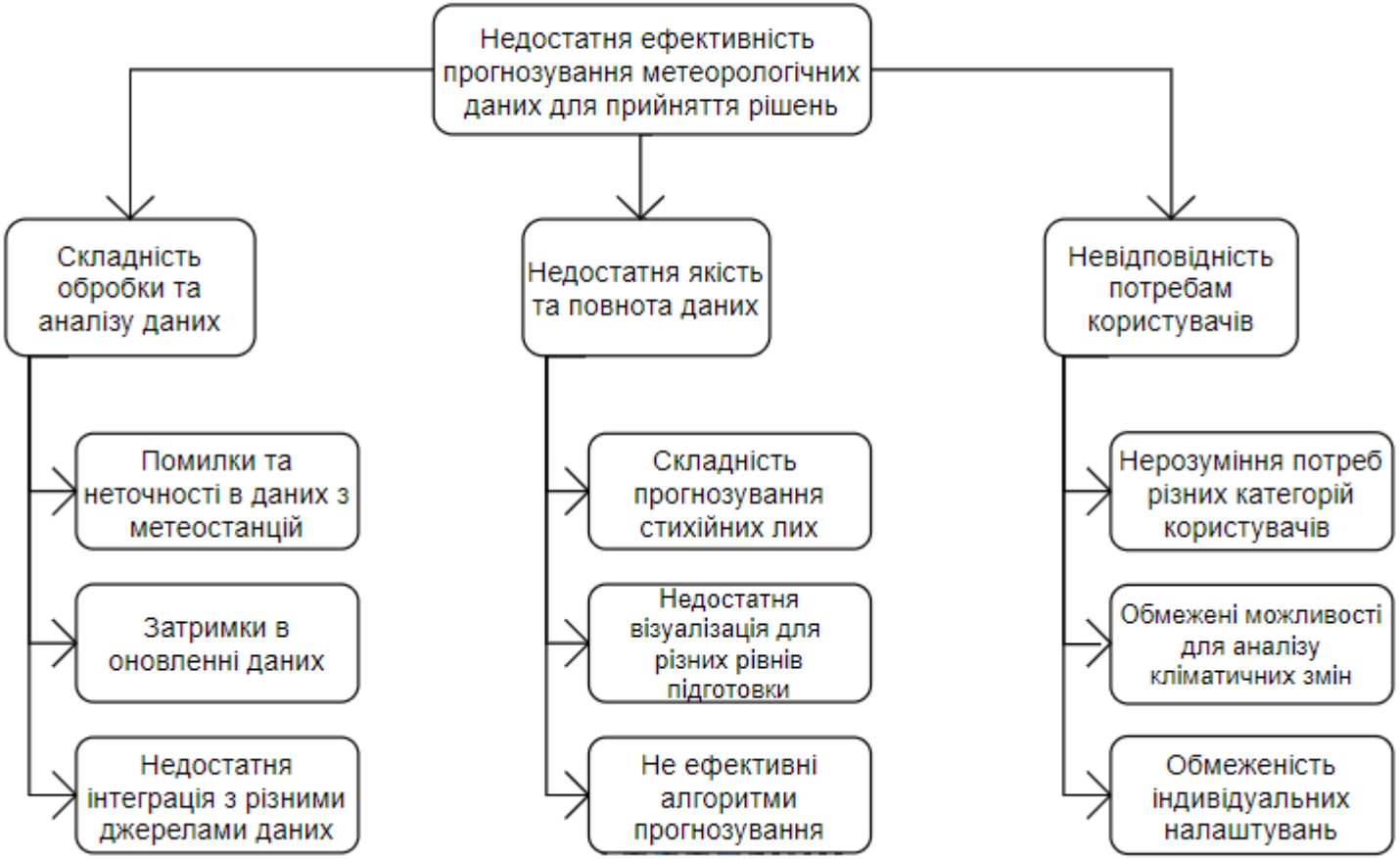
ДОДАТКИ

Додаток 1
Копії графічних матеріалів

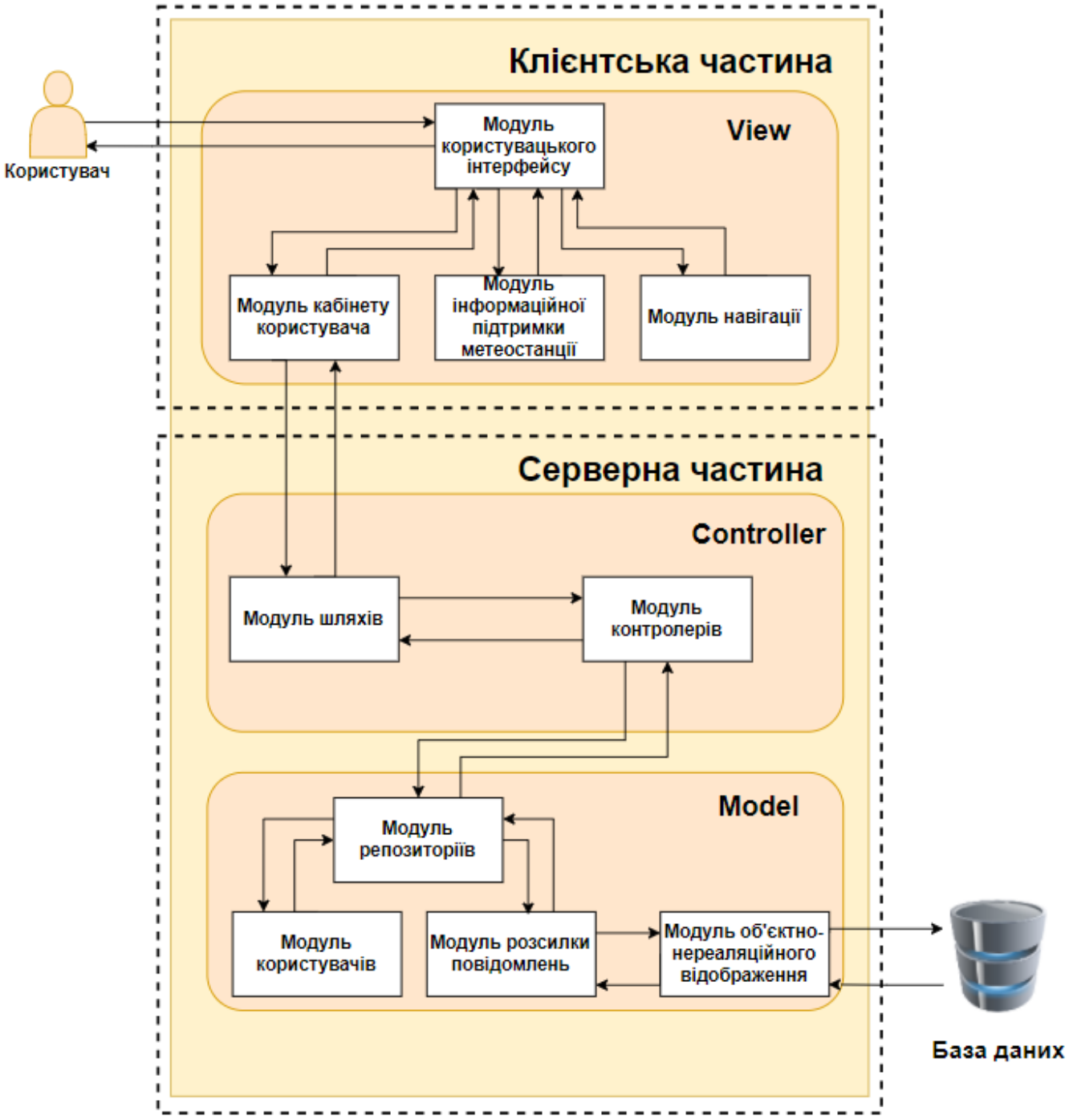


ДП.045440-06-99
Вебзастосунок для інформаційної підтримки метеостанції. Структура бази даних. ER-діаграма

ДП.045440-07-99
Вебзастосунок для інформаційної підтримки метеостанції. Функціональність вебзастосунку. UML-діаграма прецедентів



Дерево проблем потенційних користувачів системи
Пісоцька С.О., група КП-11



Архітектура вебзастосунку
Пісоцька С.О., група КП-11

Додаток 2
Лістинг програми

Category.jsx

```
  const fetchHistoricalTemperatures = async (lat, lon) => {
    const today = new Date();
    const mmdd = `${(today.getMonth() + 1).toString().padStart(2, '0')}-
${today.getDate().toString().padStart(2, '0')}`;
    const years = Array.from({ length: 10 }, (_, i) =>
today.getFullYear() - i); // Last 10 years

    const fetchStation = async () => {
      try {
        // Step 1: Get nearest station
        const stationRes = await
axios.get('https://meteostat.p.rapidapi.com/stations/nearby', {
          params: { lat, lon },
          headers: {
            'X-RapidAPI-Key': '',
            'X-RapidAPI-Host': 'meteostat.p.rapidapi.com',
          },
        });
        return stationRes.data?.data?.[0];
      } catch (error) {
        if (error.response && error.response.status === 429) {
          console.error('Rate limit exceeded, retrying...');
          const retryAfter = error.response.headers['retry-after'] || 60;
// Default retry after 60 seconds
          await new Promise(resolve => setTimeout(resolve, retryAfter *
1000)); // Wait before retry
          return fetchStation(); // Retry the request
        }
        console.error('Error fetching station:', error);
        throw error; // Rethrow error if not rate limit error
      }
    };

    const fetchTemperaturesForYear = async (stationId, year) => {
      try {
        const date = `${year}-${mmdd}`;
        const dailyRes = await
axios.get('https://meteostat.p.rapidapi.com/stations/daily', {
          params: {
            station: stationId,
            start: date,
            end: date,
          },
          headers: {
            'X-RapidAPI-Key':,
            'X-RapidAPI-Host': 'meteostat.p.rapidapi.com',
          },
        });

        const tempData = dailyRes.data?.data?.[0];
        if (tempData) {
          return { max: tempData.tmax, min: tempData.tmin, year: year };
// Include year in the result
        }
        return null; // Return null if no data found
      } catch (error) {
        if (error.response && error.response.status === 429) {
          console.error('Rate limit exceeded, retrying...');
          const retryAfter = error.response.headers['retry-after'] || 60;
// Default retry after 60 seconds
          await new Promise(resolve => setTimeout(resolve, retryAfter *
1000)); // Wait before retry
```

```
          return fetchTemperaturesForYear(stationId, year); // Retry the
request
        }
        console.error('Error fetching temperatures for year:', error);
        throw error; // Rethrow error if not rate limit error
      }
    };

    try {
      const station = await fetchStation();
      if (!station) return;

      const stationId = station.id;

      const temps = [];

      for (const year of years) {
        const tempData = await fetchTemperaturesForYear(stationId, year);
        if (tempData) {
          temps.push(tempData);
        }
      }

      if (temps.length > 0) {
        const maxTempData = temps.reduce((prev, current) => (prev.max >
current.max ? prev : current));
        const minTempData = temps.reduce((prev, current) => (prev.min <
current.min ? prev : current));

        setRecordTemps({
          max: maxTempData.max,
          min: minTempData.min,
          maxYear: maxTempData.year,
          minYear: minTempData.year
        });
      }
    } catch (err) {
      console.error('Meteostat error:', err);
    }
  };

  useEffect(() => {
  const fetchWeatherData = async () => {
    try {
      const lang = i18n.language || 'en';
      const weatherRes = await axios.get(
        `https://api.openweathermap.org/data/2.5/weather?q=${CITY}&units=
metric&lang=${lang}&appid=${API_KEY}`
      );
      const weatherData = weatherRes.data;
      setData(weatherData);

      const { lat, lon } = weatherData.coord;

      const elevationRes = await axios.get(
        `https://api.open-
meteo.com/v1/elevation?latitude=${lat}&longitude=${lon}`
      );
      setElevation(elevationRes.data.elevation);

      fetchHistoricalTemperatures(lat, lon);

    } catch (err) {
```

```
      console.error('Error fetching data:', err);
    }
  };

  fetchWeatherData();
}, [CITY, languageKey]); // <-- FIXED: add languageKey here


  const formatTime = (timestamp) => {
    const date = new Date(timestamp * 1000);
    return date.toLocaleTimeString([], { hour: '2-digit', minute: '2-
digit' });
  };

  const categoryValues = data ? [
    { label: `${t('categoryLabels.location')}`, value: `${data.name},
${data.sys.country}` },
    { label: t('categoryLabels.clouds'), value:
`${data.weather[0].description}` },
    { label: t('categoryLabels.temperature'), value:
`${data.main.temp.toFixed(1)}°C` },
    { label: t('categoryLabels.feelsLike'), value:
`${data.main.feels_like.toFixed(1)}°C` },
    { label: t('categoryLabels.pressure'), value: `${data.main.pressure}
hPa` },
    { label: t('categoryLabels.humidity'), value:
`${data.main.humidity}%` },
    {
      label: t('categoryLabels.wind'),
      value: (
        <span style={{ display: 'flex',
justifyContent:"center",textAlign:"center", alignItems: 'center', gap:
'8px', fontSize: '20px', marginBottom:"-30px" }}>
          {t('speed')}: {data.wind.speed} {t('speedms')}, {t('gust')}:
{Math.round(data.wind.gust) || 'N/A'} {t('speedms')},
          {t('direction')}: <span
            style={{
              textAlign:"flexstart",
              display: 'inline-block',
              transform: `rotate(${data.wind.deg}deg)`,
              fontSize: '1.2em',
            }}
          >
            ↓
          </span>
        </span>
      )
    },
    {
      label: t('categoryLabels.rain'),
      value: data.rain?.['3h'] ? `${data.rain['3h']} ${t('mm')} (last
3h)` : `0 ${t('mm')}`
    },
    {
      label: t('categoryLabels.sunriseSunset'),
      value: (
        <span style={{ fontSize: '30px' }}>
          {t('sunrise')}: {formatTime(data.sys.sunrise)}, {t('sunset')}:
{formatTime(data.sys.sunset)}
        </span>
      )
    }
  ] : [];
```

```
  const visibleItems = showAll ? categoryValues : categoryValues.slice(0,
6);

  const downloadCategoryDataAsTxt = () => {
    const now = new Date();
    const timestamp = now.toLocaleString('en-US', {
      weekday: 'long',
      year: 'numeric',
      month: 'short',
      day: 'numeric',
      hour: '2-digit',
      minute: '2-digit',
      second: '2-digit',
      hour12: true,
    });

    let content = `Downloaded on: ${timestamp}\n\n`;
    content += `Weather Categories for ${CITY}\n\n`;

    if (data) {
      content += `Location: ${data.name}, ${data.sys.country}\n`;
      content += `Clouds: ${data.weather[0].description}\n`;
      content += `Temperature: ${data.main.temp.toFixed(1)}°C\n`;
      content += `Feels like: ${data.main.feels_like.toFixed(1)}°C\n`;
      content += `Pressure: ${data.main.pressure} hPa\n`;
      content += `Humidity: ${data.main.humidity}%\n`;

      const windSpeed = data.wind?.speed ?? 'N/A';
      const windGust = data.wind?.gust ?? 'N/A';
      const windDeg = data.wind?.deg ?? 'N/A';
      content += `Wind: Speed ${windSpeed} m/s, Gust
${Math.round(windGust)} m/s, Direction ${windDeg}°\n`;

      const rainAmount = data.rain?.['3h'] ?? 0;
      content += `Rain (last 3h): ${rainAmount} mm\n`;

      const sunrise = new Date(data.sys.sunrise *
1000).toLocaleTimeString([], { hour: '2-digit', minute: '2-digit' });
      const sunset = new Date(data.sys.sunset *
1000).toLocaleTimeString([], { hour: '2-digit', minute: '2-digit' });
      content += `Sunrise/Sunset: Sunrise at ${sunrise}, Sunset at
${sunset}\n`;
    } else {
      content += 'No data available.\n';
    }
    const blob = new Blob([content], { type: 'text/plain' });
    const url = URL.createObjectURL(blob);
    const link = document.createElement('a');
    link.href = url;
    link.download = `weather_categories_${CITY}.txt`;
    document.body.appendChild(link);
    link.click();
    document.body.removeChild(link);
    URL.revokeObjectURL(url);
  };
  const categoryLabels = [
    t('categoryLabels.clouds'), t('categoryLabels.temperature'),
t('categoryLabels.feelsLike'), t('categoryLabels.pressure'),
t('categoryLabels.wind'), t('categoryLabels.humidity'),
t('categoryLabels.rain'), t('categoryLabels.sunriseSunset'),
  ];
```

ClimateAnalytics.jsx

```
const ClimateAnalytics = ({ userLocation }) => {
  const { user } = useUser();
  const [monthlyData, setMonthlyData] = useState([]);
  const [coords, setCoords] = useState({ lat: 50.4501, lon: 30.5234 });
  const [activeChart, setActiveChart] = useState('temperature');
  const [chartType, setChartType] = useState('line'); // 'line' or 'bar'

  const [stations, setStations] = useState([]);
  const [selectedStationId, setSelectedStationId] = useState(null);

  const [temperatureViewMode, setTemperatureViewMode] =
useState('monthlyAvg');
  const [selectedMonth, setSelectedMonth] = useState('07');
  const [selectedYear, setSelectedYear] = useState('2013');

  const chartRef = useRef();
  const { t } = useTranslation();

  const OPENWEATHER_API_KEY = ‘’;
  const RAPIDAPI_KEY =;

  useEffect(() => {
    const fetchCoordinates = async () => {
      if (!userLocation) return;
      try {
        const res = await axios.get(
          `https://api.openweathermap.org/geo/1.0/direct?q=${userLocation
}&limit=1&appid=${OPENWEATHER_API_KEY}`
        );
        if (res.data && res.data.length > 0) {
          const { lat, lon } = res.data[0];
          setCoords({ lat, lon });
        }
      } catch (err) {
        console.error('Error fetching coordinates:', err);
      }
    };
    fetchCoordinates();
  }, [userLocation]);

  useEffect(() => {
    const fetchStations = async () => {
      try {
        const res = await axios.get(
          `https://meteostat.p.rapidapi.com/stations/nearby?lat=${coords.
lat}&lon=${coords.lon}`,
          {
            headers: {
              'X-RapidAPI-Key': RAPIDAPI_KEY,
              'X-RapidAPI-Host': 'meteostat.p.rapidapi.com'
            }
          }
        );
        if (res.data?.data?.length) {
          setStations(res.data.data);
          setSelectedStationId(res.data.data[0].id);
        }
      } catch (err) {
        console.error('Error fetching stations:', err);
      }
```

```
    };
    fetchStations();
  }, [coords]);

  useEffect(() => {
    const fetchClimateData = async () => {
      if (!selectedStationId) return;
      try {
        const today = new Date();
        const endDate = today.toISOString().split('T')[0];
        const startDate = '2010-01-01';
        const response = await axios.get(
          `https://meteostat.p.rapidapi.com/stations/monthly?station=${se
lectedStationId}&start=${startDate}&end=${endDate}`,
          {
            headers: {
              'X-RapidAPI-Key': RAPIDAPI_KEY,
              'X-RapidAPI-Host': 'meteostat.p.rapidapi.com'
            }
          }
        );
        const rawData = response.data?.data;
        if (!rawData || !Array.isArray(rawData)) {
          console.error("No valid data returned from Meteostat API");
          return;
        }
        const formatted = rawData.map((item) => ({
          date: item.date.slice(0, 7),
          temperature: item.tavg,
          precipitation: item.prcp
        }));
        setMonthlyData(formatted);
      } catch (err) {
        console.error('Error fetching climate data:', err);
      }
    };
    fetchClimateData();
  }, [selectedStationId]);

  const getTemperatureChartData = () => {
    if (temperatureViewMode === 'monthlyAvg') {
      return monthlyData
        .filter(d => d.temperature !== null)
        .map(d => ({
          label: d.date,
          temperature: parseFloat(d.temperature.toFixed(2))
        }));
    } else if (temperatureViewMode === 'monthPerYear') {
      return monthlyData
        .filter(d => d.date.endsWith(`-${selectedMonth}`) &&
d.temperature !== null)
        .map(d => ({
          label: d.date.slice(0, 4),
          temperature: parseFloat(d.temperature.toFixed(2))
        }));
    } else if (temperatureViewMode === 'yearBreakdown') {
      return monthlyData
        .filter(d => d.date.startsWith(`${selectedYear}-`) &&
d.temperature !== null)
        .map(d => ({
          label: d.date.slice(5, 7),
          temperature: parseFloat(d.temperature.toFixed(2))
```

```
        }));
    }
    return [];
  };
```

Forecast.jsx

```
import { useEffect, useState } from 'react';
import { useTranslation } from 'react-i18next';
import i18n from './i18n';

const Forecast = ({ userLocation }) => {
  const [forecast, setForecast] = useState([]);
  const [loading, setLoading] = useState(true);
  const [selectedDay, setSelectedDay] = useState(null);
  const { t } = useTranslation();
  const [languageKey, setLanguageKey] = useState(0);

  const CITY = userLocation || 'Kyiv';
  const API_KEY =’’;
  const LANG = i18n.language === 'uk' ? 'uk' : 'en'; // Ensure it's
either 'uk' or 'en'

  useEffect(() => {
    const fetchForecast = async () => {
      try {
        const response = await fetch(
          `https://api.weatherapi.com/v1/forecast.json?key=${API_KEY}&q=$
{CITY}&days=7&lang=${LANG}`
        );
        const data = await response.json();
        if (data.forecast?.forecastday) {
          setForecast(data.forecast.forecastday);
        }
      } catch (error) {
        console.error('Failed to fetch forecast:', error);
      } finally {
        setLoading(false);
      }
    };
    fetchForecast();
  }, [CITY, LANG, languageKey]); // refetch when language changes

  useEffect(() => {
    const onLanguageChanged = () => {
      setLanguageKey(prev => prev + 1); // trigger re-fetch on lang
change
    };

    i18n.on('languageChanged', onLanguageChanged);
    return () => {
      i18n.off('languageChanged', onLanguageChanged);
    };
  }, []);

  const closeModal = () => setSelectedDay(null);

  return (
    <div className="forecast">
      <h1 className="mainHeader"
id="centermainHeader">{t('7daysforecast')}</h1>
      {loading ? (
        <p>{t('loadingforecast')}</p>
```

```
      ) : (
        <div className="forecast-cards">
          {forecast.map((day, index) => (
            <div className="forecast-card" key={index}>
              <h3>
                {new Date(day.date).toLocaleDateString(LANG === 'uk' ?
'uk-UA' : 'en-US', {
                  weekday: 'long',
                  month: 'short',
                  day: 'numeric',
                })}
              </h3>
              <img src={day.day.condition.icon}
alt={day.day.condition.text} />
              <p>{day.day.condition.text}</p>
              <p>△ {day.day.maxtemp_c}°C</p>
              <p>▽ {day.day.mintemp_c}°C</p>
              <p>💧 {day.day.avghumidity}% {t('humidity')}</p>
              <p>💨 {day.day.maxwind_kph} {t('kphwind')}</p>
              <button className="ExploreHours" onClick={() =>
setSelectedDay(day)}>
                {t('exploreforecast')}
              </button>
            </div>
          ))}
        </div>
      )}

      {selectedDay && (
        <div className="modal-overlay" onClick={closeModal}>
          <div className="modal-content" onClick={(e) =>
e.stopPropagation()}>
            <h2>
              {t('hourlyforecastfor')}{" "}
              {new Date(selectedDay.date).toLocaleDateString(LANG ===
'uk' ? 'uk-UA' : 'en-US', {
                weekday: 'long',
                month: 'short',
                day: 'numeric',
              })}
            </h2>
            <button className="close-button"
onClick={closeModal}>X</button>
            <div className="hourly-scroll">
              {selectedDay.hour.map((hourData, index) => (
                <div key={index} className="hour-box">
                  <p><strong>{hourData.time.split(' ')[1]}</strong></p>
                  <img src={hourData.condition.icon}
alt={hourData.condition.text} />
                  <p>{hourData.temp_c}°C</p>
                  <p>{hourData.condition.text}</p>
                  <p>💧 {hourData.humidity}%</p>
                  <p>💨 {hourData.wind_kph} {t('kph')}</p>
                </div>
              ))}
            </div>
          </div>
        </div>
      )}
    </div>
  );
```

Header.jsx

```
function Header({ onScrollTo, setUserLocation }) {
  const [isSignUpOpen, setIsSignUpOpen] = useState(false);
  const navigate = useNavigate();
  const [showLocationModal, setShowLocationModal] = useState(false);
  const [showSignUpModal, setShowSignUpModal] = useState(false);

  const { t, i18n } = useTranslation();
  const [languageIcon, setLanguageIcon] = useState('images/ukrflag.png');

  useEffect(() => {
    setLanguageIcon(i18n.language === 'en' ? 'images/ukrflag.png' :
'images/ukflag.png');
  }, [i18n.language]);

  const toggleLanguage = () => {
    const newLang = i18n.language === 'en' ? 'uk' : 'en';
    i18n.changeLanguage(newLang);
  };

  const handleProfileClick = () => {
    const user = JSON.parse(localStorage.getItem("user"));
    if (user) {
      navigate('/profile');
    } else {
      setIsSignUpOpen(true);
    }
  };

  const handleLocationClick = () => {
    setShowLocationModal(true);
  };

  return (
    <div className="header">
      <div className="top-panel">
        <img src="images/menu.png" alt="Menu Icon"
className="hamburgermenu" />
        <img src="images/Logo1.png" alt="logo" className="logo" />
        <div className="textButton">
          <button onClick={() =>
onScrollTo('dashboard')}>{t('dashboard')}</button>
          <button onClick={() =>
onScrollTo('forecast')}>{t('forecast')}</button>
          <button onClick={() =>
onScrollTo('weathermap')}>{t('weatherMap')}</button>
          <button onClick={() =>
onScrollTo('reports')}>{t('analytics')}</button>
          <button onClick={() =>
onScrollTo('contact')}>{t('contactUs')}</button>
        </div>
        <div className="iconButton">
          <button onClick={toggleLanguage}>
            <img src={languageIcon} alt="Lang Flag" style={{ width:
"36px", height: "28px" }} />
          </button>
          <button onClick={handleLocationClick}>
            <img src="/images/location1.png" alt="Location Icon" style={{
width: "28px", height: "38px" }} className="locationIcon"/>
          </button>
          <button onClick={() => onScrollTo('alerts')}>
```

```
            <img src="/images/bell.png" alt="Bell Icon"
className="bellIcon" />
          </button>
          <button onClick={handleProfileClick}>
            <img src="/images/Profile.png" alt="Profile Icon" />
          </button>
        </div>
      </div>

      {(showLocationModal || showSignUpModal) && (
        <ModalManager
          setUserLocation={setUserLocation}
          showLocationModal={showLocationModal}
          setShowLocationModal={setShowLocationModal}
          showSignUpModal={showSignUpModal}
          setShowSignUpModal={setShowSignUpModal}
        />
      )}

      <div className="mainBox">
        <h1 className="mainHeader">{t('mainHeader')}</h1>
        <h3 className="headerComment">{t('headerComment')}</h3>
        <button className="arrowbutton" id="mainShop" onClick={() =>
onScrollTo('dashboard')}>
          {t('exploreNow')}
        </button>
      </div>

      {isSignUpOpen && <SignUpModal onClose={() =>
setIsSignUpOpen(false)} />}
    </div>
  );
}

export default Header;
```

i18n.js

```
import i18n from 'i18next';
import { initReactI18next } from 'react-i18next';
import LanguageDetector from 'i18next-browser-languagedetector';
import HttpBackend from 'i18next-http-backend';

i18n
  .use(HttpBackend)
  .use(LanguageDetector)
  .use(initReactI18next)
  .init({
    fallbackLng: 'en',
    debug: true,
    interpolation: {
      escapeValue: false
    },
    backend: {
      loadPath: '/locales/{{lng}}/translation.json'
    }
  });

export default i18n;
```

Locationmodal.jsx

```
import { useUser } from './UserContext';
import { useState } from "react";
import { useTranslation } from 'react-i18next';
```

```
function LocationModal({ onSubmit, onClose, onSignInClick }) {
  const [location, setLocation] = useState("");
  const { isSignedIn } = useUser();
  const { t } = useTranslation();

  const handleLocationSubmit = () => {
    if (!isSignedIn()) {
      alert(t('locationModal.signInAlert'));
      onSignInClick();
      return;
    }

    if (location.trim()) {
      onSubmit(location);
      onClose();
    }
  };

  return (
    <div className="modal-overlay" onClick={onClose} style={{ position:
'fixed', display: 'flex', justifyContent: 'center', zIndex: 50 }}>
      <div className="modal-content" onClick={(e) => e.stopPropagation()}
style={{ width: "300px" }}>
        <h2 className="weathertitle" style={{ textAlign: "center",
marginLeft: "-20px", color: '#133154' }}>
          {t('locationModal.welcome')}
        </h2>
        <p style={{ textAlign: 'center' }}>
          {isSignedIn()
            ? t('locationModal.enterCitySignedIn')
            : t('locationModal.enterCityUnsigned')}
        </p>

        <form
          onSubmit={(e) => {
            e.preventDefault();
            handleLocationSubmit();
          }}
        >
          <input
            type="text"
            placeholder={t('locationModal.placeholder')}
            value={location}
            onChange={(e) => setLocation(e.target.value)}
            style={{ width: '95%', borderRadius: '10px', padding: '5px',
paddingLeft: '10px', fontSize: "16px" }}
          />

          <button
            type="submit"
            style={{ background: '#64A1E2', color: 'white', width:
'100%', border: 'none', borderRadius: '10px', padding: '10px', marginTop:
"20px" }}
          >
            <strong>{t('locationModal.showForecast')}</strong>
          </button>
        </form>

        {!isSignedIn() && (
          <button
```

```
            onClick={onSignInClick}
            style={{ background: '#64A1E2', color: 'white', width:
'100%', border: 'none', borderRadius: '10px', padding: '10px', marginTop:
"10px" }}
          >
            <strong>{t('locationModal.signInButton')}</strong>
          </button>
        )}
      </div>
    </div>
  );
}

export default LocationModal;
```

Manager.jsx

```
import { useEffect, useState } from 'react';

const ManageUsers = () => {
  const [users, setUsers] = useState([]);

  useEffect(() => {
    fetch('http://localhost:5000/api/admin/users', {
      headers: {
        Authorization: `Bearer ${localStorage.getItem('token')}`,
      }
    })
      .then((res) => res.json())
      .then(setUsers)
      .catch(console.error);
  }, []);

  const sendMessage = async (userId) => {
    const message = prompt('Enter message to send via email:');
    if (!message) return;

    try {
      const res = await
fetch(`http://localhost:5000/api/admin/message/${userId}`, {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
          Authorization: `Bearer ${localStorage.getItem('token')}`,
        },
        body: JSON.stringify({ message }),
      });

      const data = await res.json();
      alert(data.message || 'Message sent');
    } catch (err) {
      console.error(err);
      alert('Failed to send message');
    }
  };

  return (
    <div>
      <h2>Manage Users</h2>
      {users.map(user => (
        <div key={user._id}>
          <p>{user.firstName} {user.lastName} - {user.email}</p>
```

```
          <button onClick={() => sendMessage(user._id)}>Send
Email</button>
        </div>
      ))}
    </div>
  );
};
```

ManagerProfile.jsx

```
const ManagerProfile = ( {onClose}) => {
  const [user,setUser]=useState([]);
  const [users, setUsers] = useState([]);
  const [subject, setSubject] = useState("");
  const [message, setMessage] = useState("");
  const token = localStorage.getItem('token');
  const navigate=useNavigate();
  const [showEmailModal, setShowEmailModal] = useState(false);
  const [selectedUserEmail, setSelectedUserEmail] = useState("");
  const [newsletterSubscribers, setNewsletterSubscribers] = useState([]);
  const [showNewsletterModal, setShowNewsletterModal] = useState(false);

  useEffect(() => {
    fetchUsers();
  }, []);

  const handleClose = () => {
    localStorage.removeItem("user");
    setUser({ firstname: "", lastname: "", email: "", phone: "", plan: ""
});
    navigate("/");
  };

  const fetchUsers = async () => {
    try {
      const res = await fetch('http://localhost:5000/api/admin/users', {
        headers: {
          Authorization: `Bearer ${token}`
        }
      });
      const data = await res.json();
      if (Array.isArray(data)) {
        setUsers(data);
      } else {
        setUsers([]); // or handle the error differently if you want
        console.error('Expected array, got:', data);
      }
    } catch (error) {
      console.error(error);
    }
  };

  const openEmailModal = (email) => {
    setSelectedUserEmail(email);
    setShowEmailModal(true);
  };

  const closeEmailModal = () => {
    setSelectedUserEmail("");
    setSubject("");
    setMessage("");
    setShowEmailModal(false);
  };
```

```
  const deleteUser = async (id) => {
    try {
      await fetch(`http://localhost:5000/api/admin/users/${id}`, {
        method: 'DELETE',
        headers: {
          Authorization: `Bearer ${token}`
        }
      });
      fetchUsers(); // Refresh users after delete
    } catch (error) {
      console.error(error);
    }
  };

  const changePlan = async (id, newPlan) => {
    try {
      await fetch(`/api/users/${id}/plan`, {
        method: 'PUT',
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ plan: newPlan }),
      });
      fetchUsers();
    } catch (error) {
      console.error(error);
    }
  };

  const sendEmail = async () => {
    try {
      const response = await
fetch(`http://localhost:5000/api/admin/email`, {
        method: 'POST',
        headers: {
          "Content-Type": "application/json",
          Authorization: `Bearer ${token}`,
        },
        body: JSON.stringify({ toEmail: selectedUserEmail, subject,
message }),
      });

      const data = await response.json();

      if (!response.ok) {
        console.error("Server responded with error:", data.error);
        alert("Failed to send email: " + data.error);
        return;
      }

      alert("Email sent!");
      closeEmailModal();
    } catch (error) {
      console.error("Request failed:", error);
      alert("Error sending email");
    }
  };

  const sendNewsletter = async () => {
    try {
        await fetch('http://localhost:5000/api/admin/newsletter', {
            method: 'POST',
            headers: {
                "Content-Type": "application/json",
```

```
                Authorization: `Bearer ${token}`
              },
              body: JSON.stringify({ subject, message }),
              });
      alert("Newsletter sent!");
    } catch (error) {
      console.error(error);
    }
  };

  const fetchNewsletterSubscribers = async () => {
    try {
      const res = await
fetch('http://localhost:5000/api/admin/newsletter/subscribers', {
        headers: {
          Authorization: `Bearer ${token}`,
        },
      });
      const data = await res.json();
      if (Array.isArray(data)) {
        setNewsletterSubscribers(data);
        setShowNewsletterModal(true);
      } else {
        console.error("Expected array, got:", data);
      }
    } catch (error) {
      console.error("Error fetching subscribers:", error);
    }
  };
```

ModalManager.jsx

```
const ModalManager = ({
  setUserLocation,
  showLocationModal,
  setShowLocationModal,
  showSignUpModal,
  setShowSignUpModal,
}) => {
  const handleLocationSubmit = (location) => {
    setUserLocation(location);
    setShowLocationModal(false);
  };

  return (
    <>
      {showLocationModal && (
        <LocationModal
          onSubmit={handleLocationSubmit}
          onClose={() => setShowLocationModal(false)}
          onSignInClick={() => {
            setShowLocationModal(false);
            setShowSignUpModal(true);
          }}
        />
      )}
      {showSignUpModal && (
        <SignUpModal onClose={() => setShowSignUpModal(false)} />
      )}
    </>
  );
};

export default ModalManager;
```

NewsLetter.jsx

```
const Newsletter = () => {
  const [email, setEmail] = useState("");
  const [name, setName] = useState("");
  const [city, setCity] = useState("");
  const { user } = useUser();
  const { t } = useTranslation();

  const handleSubmit = async () => {
    if (!email || !name || !city) {
      alert(t('newsletter.fillAllFields'));
      return;
    }

    try {
      const response = await
fetch('http://localhost:5000/api/newsletter/subscribe', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ email, name, city })
      });
      const data = await response.json();
      if (response.ok) {
        alert(t('newsletter.success'));
      } else {
        alert(data.error || t('newsletter.failure'));
      }
    } catch (error) {
      console.error(error);
      alert(t('newsletter.error'));
    }
  };

  return (
    <div className="newsletter">
      <div className="newsletterContainer">
        <img className="imgfornews" src="images/sunnyweather.png"
alt="newsletter" />
        <div className="newsrightContainer">
          <h1 className="firstlinenews">{t('newsletter.join')}</h1>
          <h1 className="secondlinenews">{t('newsletter.title')}</h1>
          <p className="newsparagraph">{t('newsletter.description')}</p>
          <div style={{ display: "flex" }}>
            <input
              className="emailnewsinput"
              placeholder={t('newsletter.emailPlaceholder')}
              value={email}
              onChange={(e) => setEmail(e.target.value)}
            />
            <input
              className="emailnewsinput"
              placeholder={t('newsletter.namePlaceholder')}
              value={name}
              onChange={(e) => setName(e.target.value)}
            />
            <input
              className="emailnewsinput"
              placeholder={t('newsletter.cityPlaceholder')}
              value={city}
              onChange={(e) => setCity(e.target.value)}
            />
          </div>
```

```
          <button
            className="subscribeNews"
            onClick={handleSubmit}
            disabled={user?.plan !== 'premium'}
            title={user?.plan !== 'premium' ? t('onlyForPremiumUsers') :
""}
          >
            {user?.plan === "premium" ? t('newsletter.subscribe') :
t('onlyForPremium')}
          </button>
        </div>
      </div>
    </div>
  );
};
```

Profile.jsx

```
const Profile = ({ onClose }) => {
  // Provide a fallback for context if undefined
  const context = useUser() || {};
  const contextUser = context.user || null;
  const setContextUser = context.setUser || (() => {});

  const [user, setUser] = useState({
    firstname: "",
    lastname: "",
    email: "",
    phone: "",
    plan: "",
    avatarUrl: ""
  });

  const navigate = useNavigate();

  useEffect(() => {
    const storedUser = localStorage.getItem("user");
    if (storedUser) {
      setUser(JSON.parse(storedUser));
    } else if (contextUser) {
      setUser(contextUser || {});
    }
  }, [contextUser]);

  useEffect(() => {
    // Ensure the avatarUrl is updated correctly
    if (user.avatarUrl) {
      setUser(prevUser => ({ ...prevUser, avatarUrl: user.avatarUrl }));
    }
  }, [user.avatarUrl]);

  const handleClose = () => {
    onClose?.();
  };

  const handleLogout = () => {
    localStorage.removeItem("user");
    localStorage.removeItem("token");
    setContextUser(null);
    navigate("/", { replace: true });
    window.location.reload();
  };
```

```
  const handleChange = (e) => {
    const { name, value } = e.target;
    setUser(prev => ({ ...prev, [name]: value }));
  };

  const handleSaveChanges = async () => {
    try {
      const token = localStorage.getItem('token');
      const response = await fetch('http://localhost:5000/api/auth/me', {
        method: 'PUT',
        headers: {
          'Content-Type': 'application/json',
          'Authorization': `Bearer ${token}`
        },
        body: JSON.stringify(user),
      });
      if (response.ok) {
        const updatedUser = await response.json();
        setUser(updatedUser);
        setContextUser(updatedUser);
        localStorage.setItem("user", JSON.stringify(updatedUser));
        alert('Profile updated successfully!');
      } else {
        alert('Failed to update profile.');
      }
    } catch (error) {
      console.error('Error updating profile:', error);
      alert('An error occurred.');
    }
  };

  const handleAvatarChange = async (e) => {
    const file = e.target.files[0];
    if (!file) return;

    const formData = new FormData();
    formData.append('avatar', file);

    try {
      const token = localStorage.getItem('token');
      if (!token) {
        alert('No token found! Please log in again.');
        return;
      }

      // Delete old avatar first if exists
      if (user.avatarUrl) {
        await fetch('http://localhost:5000/api/auth/delete-avatar', {
          method: 'DELETE',
          headers: {
            'Content-Type': 'application/json',
            'Authorization': `Bearer ${token}`
          },
          body: JSON.stringify({ avatarUrl: user.avatarUrl })
        });
      }

      const response = await
fetch('http://localhost:5000/api/auth/upload-avatar', {
        method: 'POST',
```

```
        headers: {
          'Authorization': `Bearer ${token}`
        },
        body: formData,
      });

      const data = await response.json();

      if (response.ok) {
        const updatedUser = { ...user, avatarUrl: data.avatarUrl };
        console.log(data.avatarUrl);
        setUser(updatedUser);
        setContextUser(updatedUser);
        localStorage.setItem("user", JSON.stringify(updatedUser));
        alert('Avatar updated!');
      } else {
        alert(data.message || 'Failed to upload avatar.');
      }
    } catch (error) {
      console.error('Avatar upload error:', error);
      alert('An error occurred while uploading the avatar.');
    }
  };

  return (
    <div style={{
      backgroundImage:
"url('https://images.pexels.com/photos/1118873/pexels-photo-
1118873.jpeg')",
      backgroundSize: "cover",
      backgroundPosition: "center",
      display: "flex",
      justifyContent: "center",
      alignItems: "center",
      height: "100vh",
      width: "100%"
    }}>
      <div style={{
        zIndex: 99,
        padding: "30px",
        background: "white",
        borderRadius: "10px",
        boxShadow: "0 4px 6px rgba(0, 0, 0, 0.1)",
        width: "80%",
        maxWidth: "600px"
      }}>
        <button onClick={handleClose} style={{ float: 'right',
background: 'none', border: 'none', color: 'black' }}>❌</button>
        <h2 style={{ fontFamily: "Open Sans", fontWeight: 100 }}>Account
Settings</h2>
        <div className="profilecontainer">
          <div className="profileleftside">
            <img
              src={user.avatarUrl && user.avatarUrl !== '' ?
user.avatarUrl : "https://cdn.pixabay.com/photo/2015/10/05/22/37/blank-
profile-picture-973460_1280.png"}
              style={{ borderRadius: "50%", width: "150px", height:
"150px" }}
              alt="Profile"
              id="preview"
            />
            <br />
```

```
            <input
              type="file"
              accept="image/*"
              style={{ display: 'none' }}
              id="avatarInput"
              onChange={handleAvatarChange}
            />
            <button className="changeavatarbtn" onClick={() =>
document.getElementById('avatarInput').click()}>
              Change Avatar
            </button>
          </div>
          <div className="profilerightside">
            <div className="profileRow">
              <strong>Name:</strong>
              <input
                className="profileinput"
                placeholder={user.firstname }
                onChange={handleChange}
                name="firstname"
                style={{width:"50px"}}
              />
              <input
                className="profileinput"
                placeholder={user.lastname}
                onChange={handleChange}
                name="lastname"
              />
            </div>
            <div className="profileRow">
              <strong>Email:</strong> <input className="profileinput"
placeholder={user.email || "N/A"}  onChange={(e) => setUser({...user,
email: e.target.value})} name="email"/>
            </div>
            <div className="profileRow">
              <strong>Phone:</strong> <input className="profileinput"
placeholder={user.phone || "N/A"}  onChange={(e) => setUser({...user,
phone: e.target.value})}  name="phone"/>
            </div>
            <div className="profileRow">
              <strong>Password:</strong> <input className="profileinput"
placeholder="****"  onChange={(e) => setUser({...user, password:
e.target.value})}  name="password" />
            </div>
            <div className="profileRow">
              <strong>Plan:</strong> <input className="profileinput"
placeholder={user.plan || "Free"}  onChange={(e) => setUser({...user,
plan: e.target.value})} name="plan" />
            </div>
            <div className="profileRow">
              <button className="profilebottombtn"
onClick={handleSaveChanges}>Save Changes</button>
              <button className="profilebottombtn"
onClick={handleLogout}>Log Out</button>
            </div>
          </div>
        </div>
      </div>
    </div>
  );
};

export default Profile;
```

RecentAlerts.jsx

```
import { useEffect, useState } from 'react';
import { useUser } from './UserContext';
import { useTranslation } from 'react-i18next';
import i18n from './i18n';

const RecentAlerts = ({userLocation}) => {
  const [alerts, setAlerts] = useState([]);
  const [loading, setLoading] = useState(true);
  const [precipDays, setPrecipDays] = useState([]);
  const { user } = useUser();
  const city = userLocation || 'Kyiv';
  const { t } = useTranslation();
  const currentLanguage = i18n.language || 'en-US';

  useEffect(() => {
    const city = userLocation || 'Kyiv';
    const fetchAlerts = async () => {
      try {
        const response = await fetch(
          `https://api.weatherapi.com/v1/forecast.json?key=&q=${city}&day
s=3&alerts=yes`
        );
        const data = await response.json();

        // Set official alerts
        if (data.alerts && Array.isArray(data.alerts.alert)) {
          const uniqueAlerts = [];
          const seenHeadlines = new Set();

          for (const alert of data.alerts.alert) {
            if (!seenHeadlines.has(alert.headline)) {
              seenHeadlines.add(alert.headline);
              uniqueAlerts.push(alert);
            }
          }
          setAlerts(uniqueAlerts);
        } else {
          setAlerts([]);
        }

        // Extract days with precipitation
        const forecastDays = data.forecast?.forecastday || [];
        const rainyDays = forecastDays
          .filter(day => day.day.totalprecip_mm > 0)
          .map(day => ({
            date: new Date(day.date).toLocaleDateString(currentLanguage,
{
              weekday: 'short',
              month: 'short',
              day: 'numeric',
            }),
            amount: day.day.totalprecip_mm.toFixed(1),
          }));

        setPrecipDays(rainyDays);
      } catch (error) {
        console.error('Failed to fetch alerts:', error);
      } finally {
        setLoading(false);
      }
    };
```

```
    fetchAlerts();
  }, [userLocation]);

  const downloadAlertsAsTxt = () => {
    const now = new Date();
    const timestamp = now.toLocaleString(currentLanguage, {
      weekday: 'long',
      year: 'numeric',
      month: 'short',
      day: 'numeric',
      hour: '2-digit',
      minute: '2-digit',
      second: '2-digit',
      hour12: true,
    });

    let content = `Downloaded on: ${timestamp}\n\n`;
    content += `Weather Alerts for ${city}\n\n`;

    if (alerts.length > 0) {
      alerts.forEach((alert, index) => {
        content += `Alert ${index + 1}:\n`;
        content += `Headline: ${alert.headline}\n`;
        content += `Description: ${alert.desc}\n`;
        content += `Severity: ${alert.severity}\n`;
        content += `Source: ${alert.msgtype} | ${alert.areas}\n`;
        content += `Expires: ${new
Date(alert.expires).toLocaleString()}\n\n`;
      });
    } else {
      content += `No active alerts for your location.\n`;
      if (precipDays.length > 0) {
        content += `\n🌧 Precipitation expected on:\n`;
        precipDays.forEach(day => {
          content += `- ${day.date}: ${day.amount} mm\n`;
        });
      } else {
        content += `⬡ Expect sunny weather throughout the next 3
days.\n`;
      }
    }

    const blob = new Blob([content], { type: 'text/plain' });
    const url = URL.createObjectURL(blob);
    const link = document.createElement('a');
    link.href = url;
    link.download = `weather_alerts_${city}.txt`;
    document.body.appendChild(link);
    link.click();
    document.body.removeChild(link);
    URL.revokeObjectURL(url);
  };

  const getAlertStyleFromTitle = (title) => {
    const firstWord = title.split(' ')[0].toLowerCase();
    let backgroundColor = '#f0f0f0';

    switch (firstWord) {
      case 'red':
        backgroundColor = '#ffcccc';
        break;
```

```
      case 'yellow':
        backgroundColor = '#fff4cc';
        break;
      case 'orange':
        backgroundColor = '#FFBE79';
        break;
      case 'winter':
        backgroundColor = '#cce0ff';
        break;
      default:
        break;
    }

    return {
      backgroundColor,
      borderLeft: `6px solid ${firstWord === 'red' ? '#cc0000' :
        firstWord === 'yellow' ? '#ffcc00' :
        firstWord === 'orange' ? '#ff7f00' :
        firstWord === 'winter' ? '#3399cc' : '#cccccc'}`,
      padding: '16px',
      marginBottom: '20px',
      borderRadius: '12px',
      boxShadow: '0 2px 6px rgba(0, 0, 0, 0.1)',
      display: 'flex',
      alignItems: 'flex-start',
    };
  };

  return (
    <div className="recentalerts">
      <div style={{display:'flex', justifyContent:"space-between"}}>
      <h1 className="alertname">{t('recentAlerts.title')}</h1>
      <button
        onClick={() => user?.plan === 'premium' && downloadAlertsAsTxt()}
        className="downloadbar"
        disabled={user?.plan !== 'premium'}
        title={user?.plan === 'premium' ? t('recentAlerts.importData') :
t('recentAlerts.premiumFeature')}
        style={{
          marginTop: '30px',
          marginRight: '40px',
          marginBottom: '-40px',
          fontSize: '16px',
          border: 'none',
          borderRadius: '8px',
          cursor: user?.plan === 'premium' ? 'pointer' : 'not-allowed',
          height: "60px",
          width: "225px",
          opacity: user?.plan === 'premium' ? 1 : 0.6
        }}
      >
        {user?.plan === "premium" ? t('recentAlerts.importData') :
t('recentAlerts.premiumFeature')}
      </button>

    </div>
      {loading ? (
  <p style={{ fontSize: '20px'
}}><pre>    {t('recentAlerts.loading')}</pre></p>
) : alerts.length === 0 ? (
  <div>
    <p style={{ fontSize: '20px' }}>
```

```
      <pre>{t('recentAlerts.noAlerts')}</pre>
    </p>
    {precipDays.length === 0 && (
      <div style={{ fontSize: '20px', marginTop: '10px' }}>
        <pre>{t('recentAlerts.sunnyForecast')}</pre>
      </div>
    )}
    {precipDays.length > 0 && (
      <div style={{ fontSize: '20px', marginTop: '10px' }}>
        <pre>{t('recentAlerts.precipExpected')}</pre>
        <ul style={{ marginLeft: '40px' }}>
          {precipDays.map((day, index) => (
            <li key={index}>
                {day.date}: {day.amount} {t('mm')}
            </li>
          ))}
        </ul>
      </div>
    )}
  </div>
) : (
  alerts.map((alert, index) => (
    <div
      className="alert"
      key={index}
      style={getAlertStyleFromTitle(alert.headline)}
    >
      <div className="alert-icon-container">
        <img src="images/alert.png" alt="alert" className="alerticon" />
      </div>
      <div className="alertDesc">
        <h3 className="alertTitle">{alert.headline}</h3>
        <p className="alertparagraph">{alert.desc}</p>
        <p className="alertparagraph">
          <strong>{t('recentAlerts.severity')}:</strong> {alert.severity}
        </p>
        <p className="alertparagraph">
          <em>{t('recentAlerts.source')}: {alert.msgtype} |
{alert.areas}</em>
        </p>
        <p className="alertparagraph">
        <em>
          {t('recentAlerts.expires')}:{" "}
          {new Date(alert.expires).toLocaleString(currentLanguage, {
            weekday: "short",
            month: "short",
            day: "numeric",
            hour: "2-digit",
            minute: "2-digit",
            hour12: true,
            timeZoneName: "short",
          })}
        </em>
        </p>
      </div>
    </div>
  ))
)}
    </div>
  );
};
export default RecentAlerts;
```

Додаток 3
Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
"КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО"

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП'ЮТЕРНИХ СИСТЕМ

ВЕБЗАСТОСУНОК ДЛЯ ІНФОРМАЦІЙНОЇ ПІДТРИМКИ РОБОТИ МЕТЕОСТАНЦІЇ

Виконала: Пісоцька Софія Олександрівна
Керівник: к.т.н., доцент, Новак Дмитро Сергійович

Київ - 2025

ПОСТАНОВКА ЗАДАЧІ

Мета проєкту: розроблення вебзастосунку для інформаційної підтримки метеостанції, який забезпечує доступ до актуальної, структурованої та персоналізованої метеорологічної інформації для аналізу кліматичних змін та моніторингу погодних умов.

Завдання:

1. Проаналізувати ринок ПЗ для інформаційної підтримки метеостанції
2. Порівняти конкурентів, виявити їхні сильні та слабкі сторони
3. Визначити функціональні та нефункціональні вимоги до ПЗ
4. Підготувати дизайн застосунку
5. Обґрунтувати вибір стеку технологій для розроблення ПЗ
6. Програмно реалізувати вебзастосунок для інформаційної підтримки метеостанції
7. Протестувати систему на відповідність вимогам
8. Навести шляхи подальшого покращення застосунку

АКТУАЛЬНІСТЬ

1. Зростання частоти аномальних погодних явищ та кліматичних змін.

2. Недостатня гнучкість і складний інтерфейс традиційних метеосервісів.

3. Відсутність можливості аналізу кліматичних даних і персоналізації під потреби користувача.

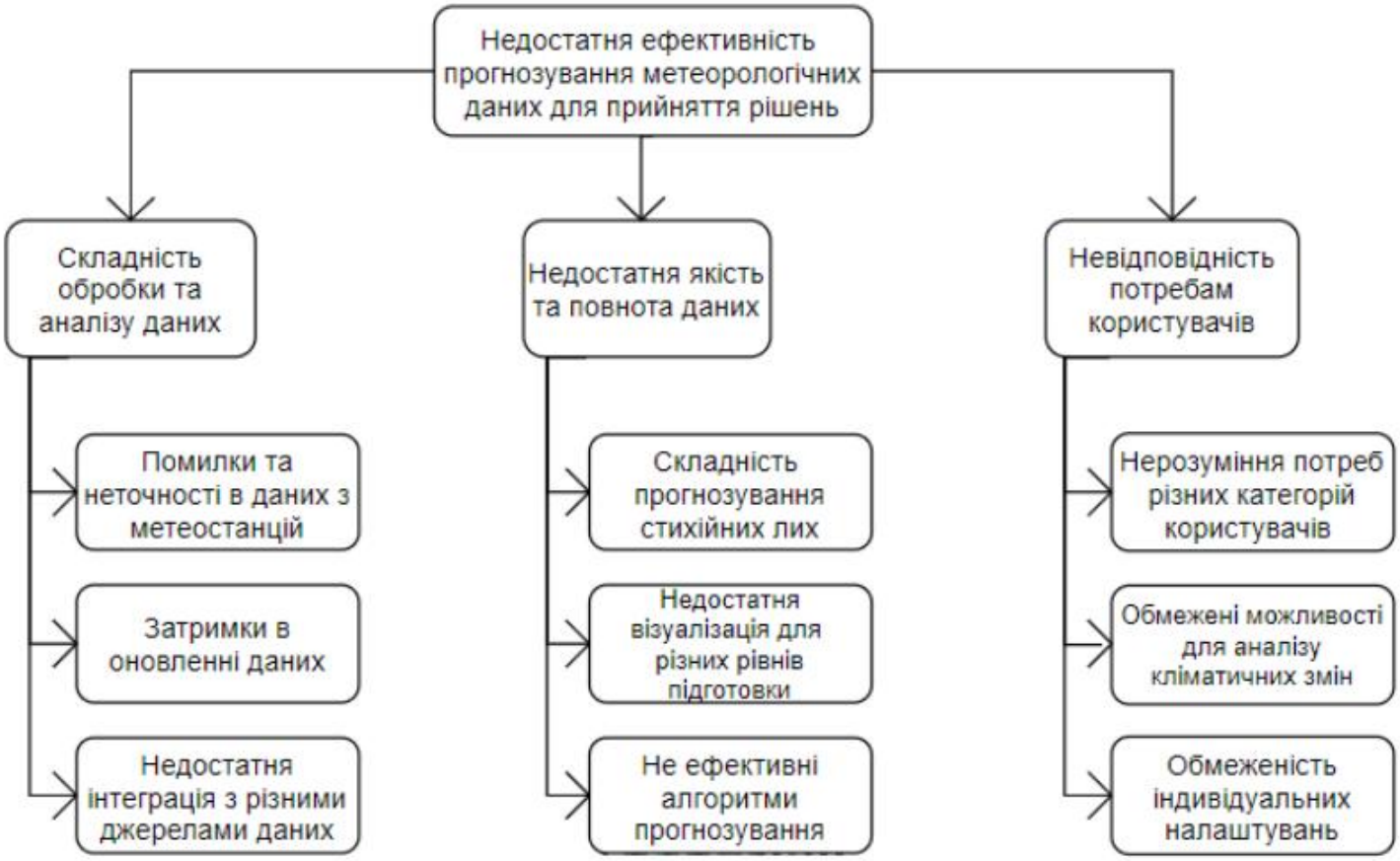
ІСНУЮЧІ РІШЕННЯ

AccuWeather

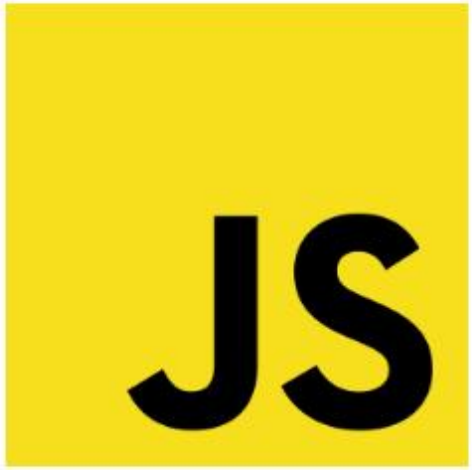
Windy.com

Weather.com

ДЕРЕВО ПРОБЛЕМ

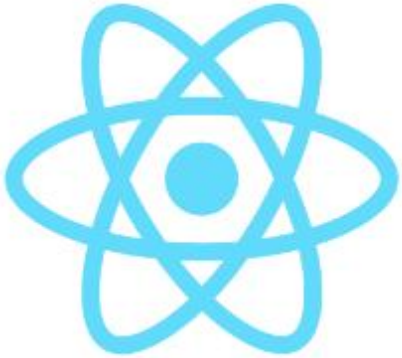


ВИБІР ТЕХНОЛОГІЙ РОЗРОБЛЕННЯ



JavaScript

Express.js



React.js

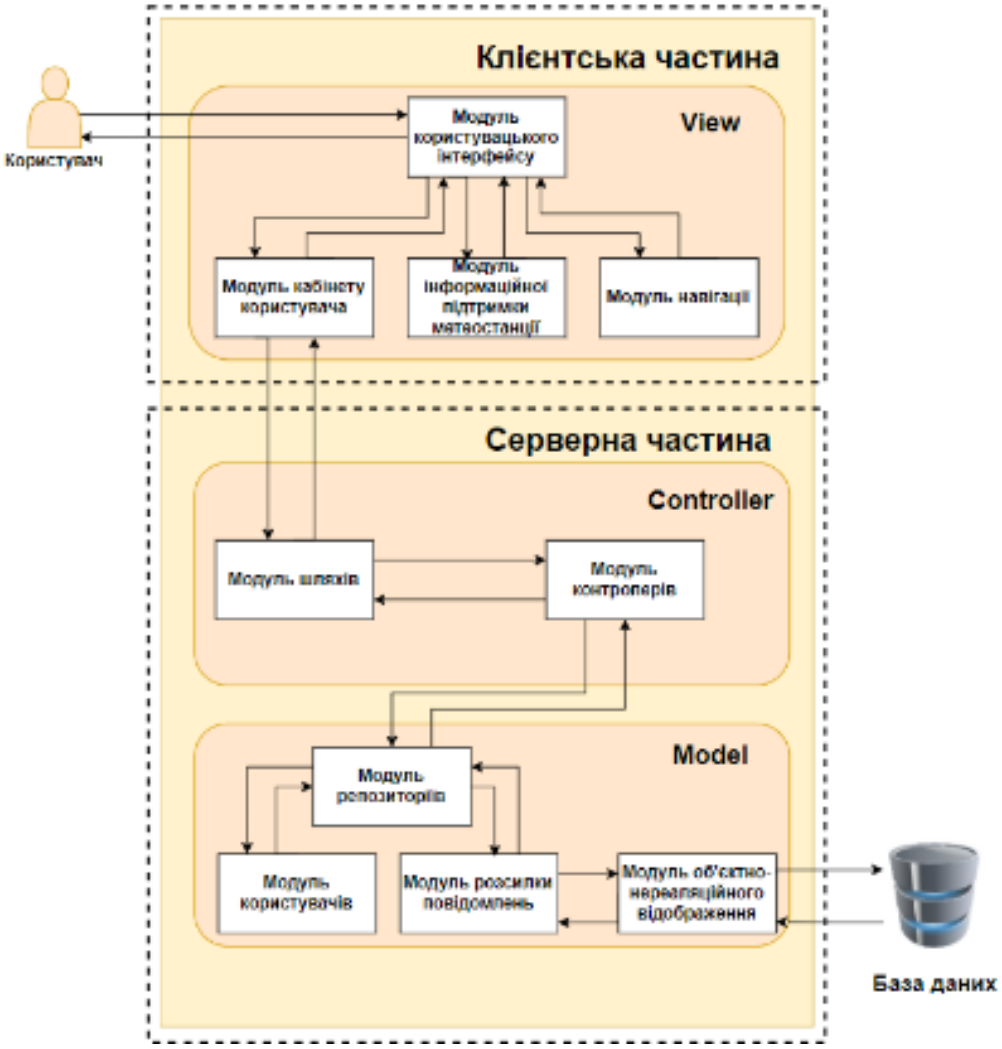
MongoDB

Nodemailer

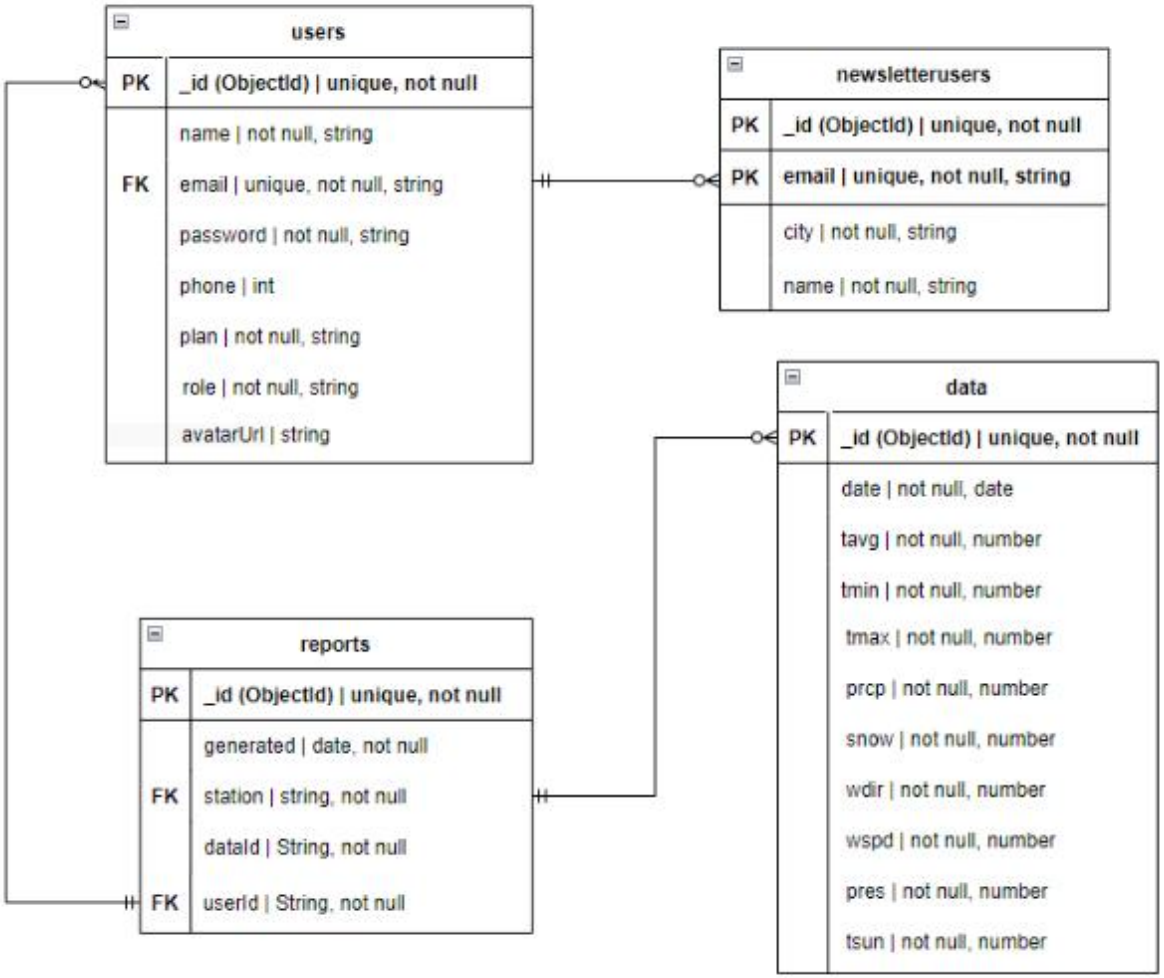
node-cron

i18next

АРХІТЕКТУРА СИСТЕМИ

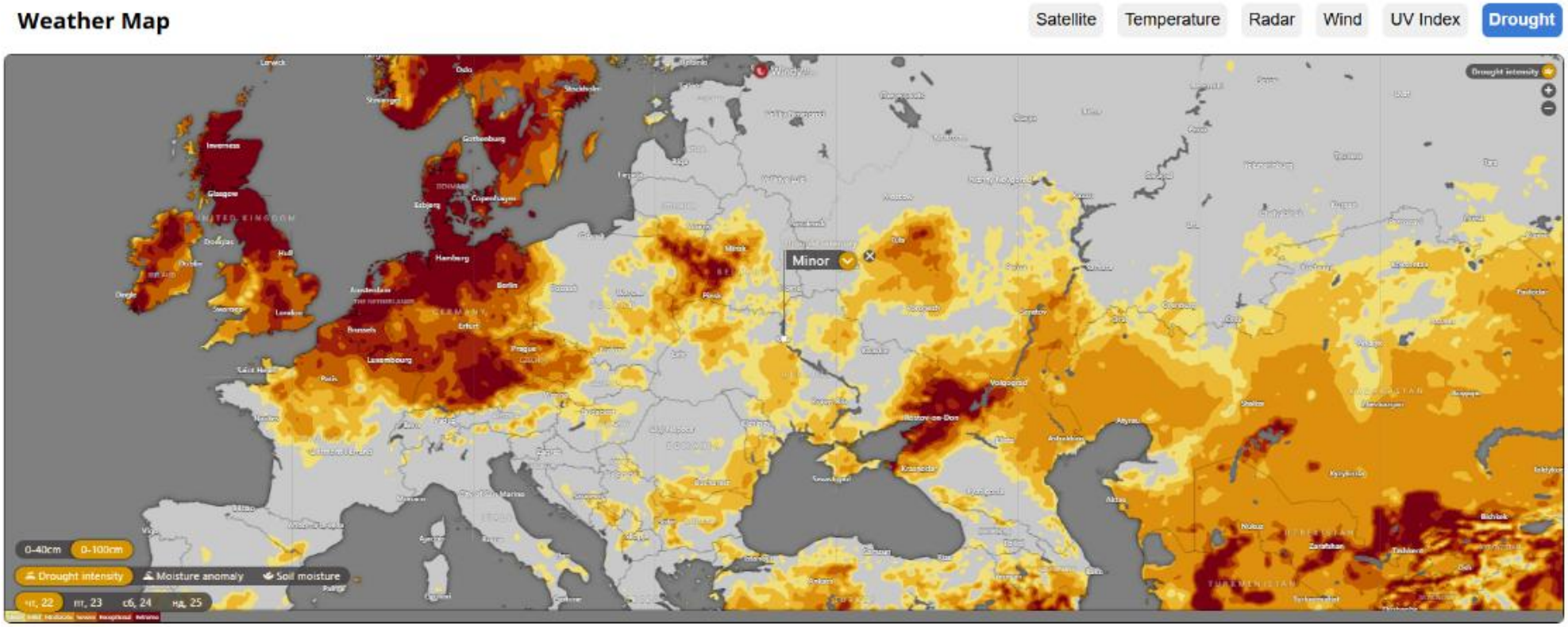


СТРУКТУРА БАЗИ ДАНИХ



ФУНКЦІОНАЛЬНІСТЬ СИСТЕМИ

ПРИКЛАД РОБОТИ ПРОГРАМИ. КАРТИ ПОГОДНИХ УМОВ



Drought Intensity: Shows general dryness based on long-term conditions. The marker reflects **drought severity**.

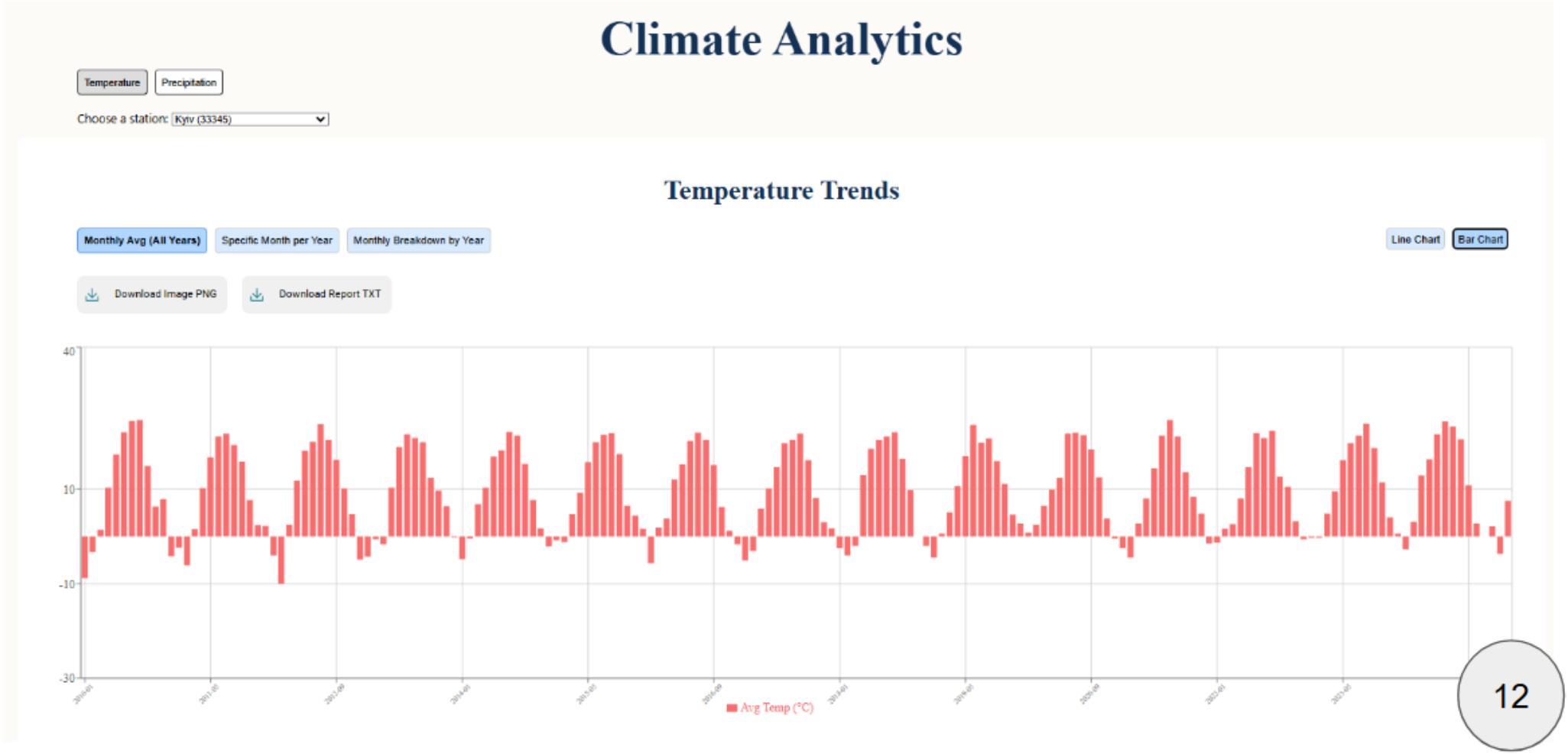
Moisture Anomaly: Displays how current humidity differs from the average. The marker value is in **millimeters** (e.g., **-15.6 mm** means significantly drier than usual).

Soil Moisture: Indicates the percentage of **moisture in the soil**. The marker value (e.g., **60%**) shows how saturated the soil is at the selected location.

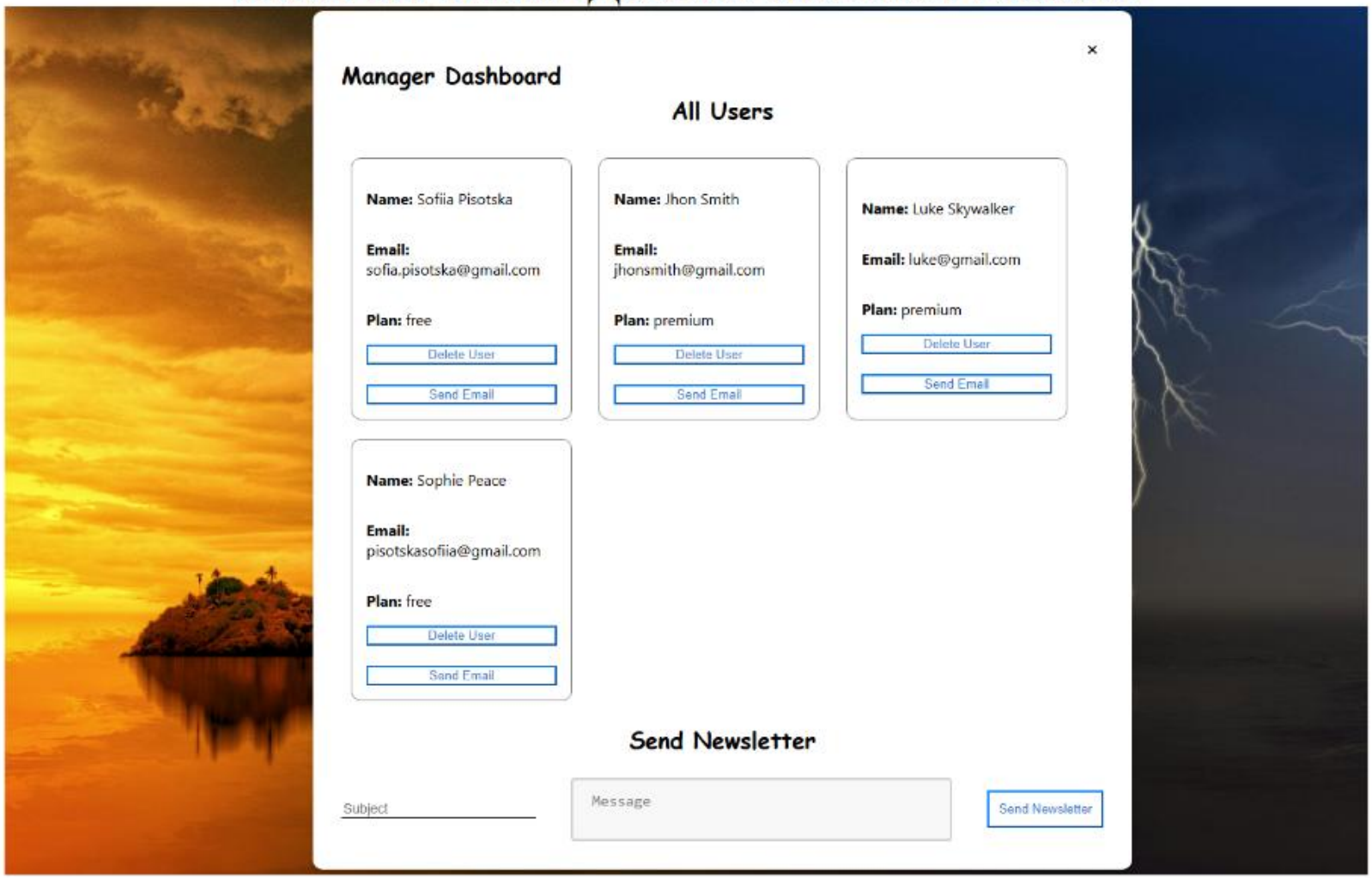
The drought map shows **soil moisture levels** at different depths. Switching between **0–40 cm** and **0–100 cm** lets you view **short-term surface conditions** or **deeper, long-term moisture trends** important for agriculture.

ПРИКЛАД РОБОТИ ПРОГРАМИ. СПОВІЩЕННЯ ПРО НЕБЕЗПЕЧНІ ПОГОДНІ УМОВИ

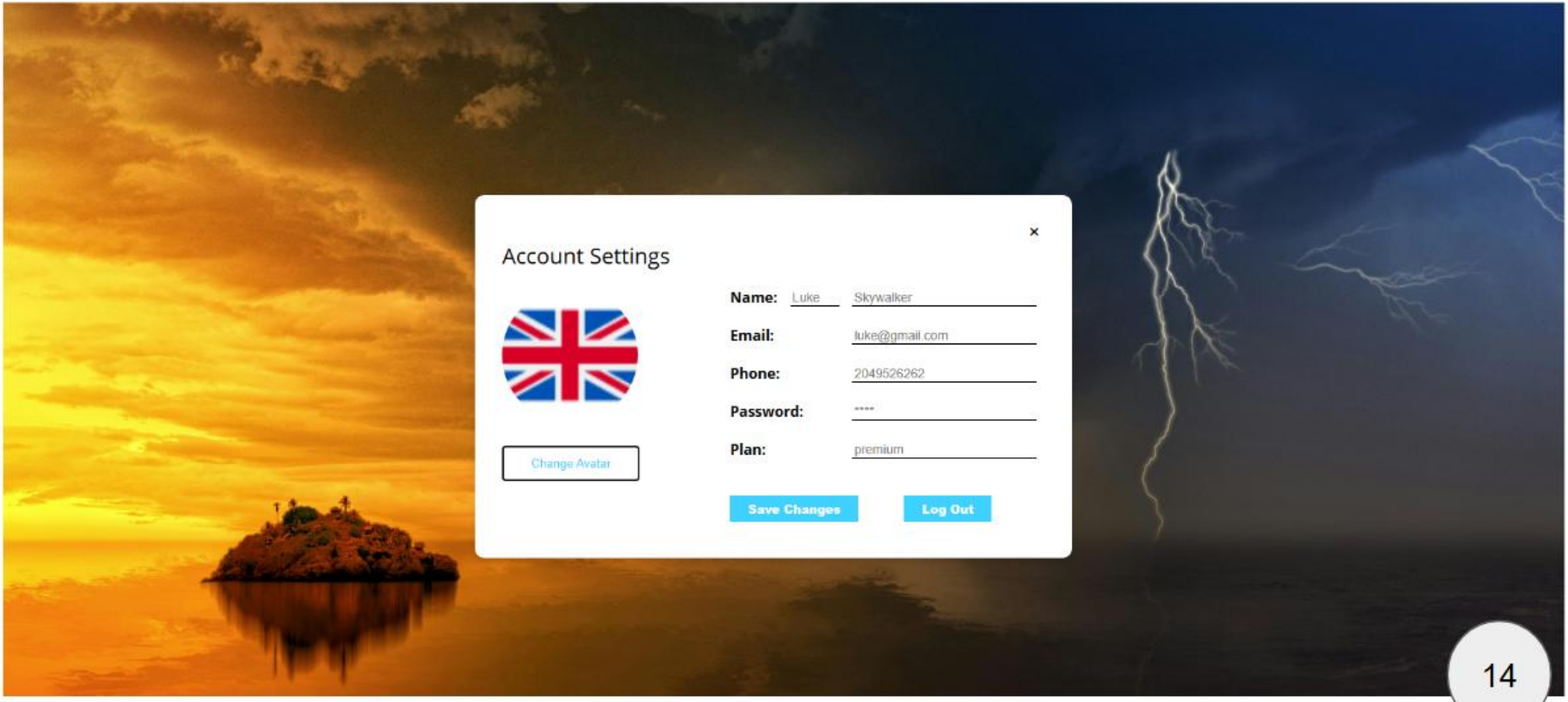
ПРИКЛАД РОБОТИ ПРОГРАМИ. АНАЛІТИКА КЛІМАТУ



ПРИКЛАД РОБОТИ ПРОГРАМИ. АКАУНТ МЕНЕДЖЕРА КОРИСТУВАЧІВ



ПРИКЛАД РОБОТИ ПРОГРАМИ. АКАУНТ КОРИСТУВАЧА



ПОРІВНЯННЯ З АНАЛОГАМИ

Критерій	Аналоги			
	AccuWeather	Windy	Weather.com	**Розроблене ПЗ**
Локалізація інформації	-	+	+	+
Персоналізовані сповіщення про негоду	+	-	-	+
Інтерактивна карта	+	+	-	+
Історичні кліматичні зміни	-	-	-	+
Порівняння з минулими роками	-	-	-	+
Спеціалізація на певних сферах	-	+	-	+
Простий та зрозумілий інтуїтивно інтерфейс	-	-	+	+
Можливість реєстрації для особистого досвіду	-	+	-	+
Відкритість (відкриті API)	-	-	-	+
Візуалізація (графіки)	-	+	+	+

НАПРЯМКИ РОЗВИТКУ

- **Розширення API-інтеграцій** — додавання підтримки нових метеосервісів для точніших прогнозів.

- **Персоналізація** — улюблені місця, одиниці виміру, сповіщення за умовами.

- **Покращення графіків** — додати інтерактивність, масштабування, різні типи візуалізацій.

- **Оптимізація продуктивності** — кешування, асинхронна та розподілена обробка даних.

ВИСНОВКИ

1. Проаналізовано існуючі рішення.

2. Висунуто ідею розробки.

3. Спроєктовано архітектуру системи та структуру бази даних.

4. Налаштовано взаємодію з сторонніми API.

5. Реалізовано функціональність для організації робочого процесу.

6. Представлені напрямки подальшого вдосконалення системи.

Розробка виконана у повному обсязі згідно з положенням Технічного завдання, тестування продукту проведено відповідно до затвердженої програми та методики тестування.

Дякую за увагу!

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

_____ Євгенія СУЛЕМА

"___" _____ 2024 р.

ВЕБЗАСТОСУНОК ДЛЯ ІНФОРМАЦІЙНОЇ ПІДТРИМКИ РОБОТИ МЕТЕОСТАНЦІЇ

Програма та методика тестування

ДП.045440-04-51

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Дмитро НОВАК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Софія ПІСОЦЬКА

2024

ЗМІСТ

1. ОБ’ЄКТ ВИПРОБУВАНЬ……..3
2. МЕТА ТЕСТУВАННЯ……3
3. МЕТОДИ ТЕСТУВАННЯ……..3
4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ…….4

1. ОБ’ЄКТ ВИПРОБУВАНЬ

Вебзастосунок для інформаційної підтримки роботи метеостанції, серверна частина якого написана мовою програмування JavaScript з використанням веб-фреймворком Node.js, а клієнтська частина – мовою програмування JavaScript із використанням бібліотеки для розробки користувацького інтерфейсу React.

2. МЕТА ТЕСТУВАННЯ

Метою тестування є перевірка наступних елементів:

1. Працездатність функцій реєстрації та автентифікації.
2. Попередження неавторизованого доступу.
3. Функціональна працездатність елементів сторінок веб-додатка.
4. Доступ до бази даних через основну серверну частину.
5. Коректна робота функціоналу створення отримання погодних даних, зміна локації, зміна мови.
6. Робота сервера з модулем, що відповідає за інтеграцію інструмента для інформаційної підтримки метеостанції в вебзастосунку.
7. Забезпечення коректної обробки REST запитів та відповідей.
8. Відповідність дизайну вимогам програмного забезпечення.

3. МЕТОДИ ТЕСТУВАННЯ

Тестування виконується відповідно до техніки сірого ящику або Gray Box Testing. Існують ще два схожі підходи до тестування:

1. Black Box Testing – тестування відбувається без будь-яких знань внутрішньої структури системи. Тестувальник має доступ до програми тільки через ті ж інтерфейси, що і користувач.

2. White Box Testing – тестування здійснюється на основі внутрішнього функціонування і логіки роботи коду. Тестувальнику відомо все про систему, щоб точно розпізнати частину коду, яка має помилку.

Gray Box Testing є середнім між двома вищеописаними підходами тестування. Тестувальник в даному разі має обмежені знання про внутрішні деталі системи й перевіряє як вихідний код, так і сам програмний продукт на відповідність існуючим програмним функціональним та нефункціональним вимогам.

Використовуються такі методи тестування:

1. Функціональне тестування.
2. Ручне тестування інтерфейсу.
3. Тестування продуктивності веб-додатка.

4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Повний процес тестування проходить у такому порядку:

1. Статичне тестування коду.
2. Динамічне ручне тестування на відповідність функціональним вимогам технічного завдання.
3. Тестування інтерфейсу при різній роздільній здатності екрану.
4. Тестування інтерфейсу в різних браузерах: Chrome, Safari, Firefox.
5. Тестування при конкурентному навантажені на систему з боку декількох користувачів.
6. Тестування стабільності роботи при різних зовнішніх умовах (максимальному навантаженні на оперативну пам’ять та процесор, низька швидкість мережі Інтернет).

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

_____ Євгенія СУЛЕМА

"___" _____ 2025 р.

ВЕБЗАСТОСУНОК ДЛЯ ІНФОРМАЦІЙНОЇ ПІДТРИМКИ РОБОТИ МЕТЕОСТАНЦІЇ

Керівництво користувача

ДП.045440-05-34

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Дмитро НОВАК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Софія ПІСОЦЬКА

2025

ЗМІСТ

1. Структура вебзастосунка.....3
2. Процедура реєстрації та авторизації 4
3. Види підписок та налаштування їх опцій.....6
4. Налаштування робочого процесу для використання автоматизованої перевірки.....10
5. Перегляд результатів виявлення плагіату.....18

1. СТРУКТУРА ВЕБЗАСТОСУНКУ

Вебзастосунок для інформаційної підтримки роботи метеостанції складається лише з динамічних вебсторінок. Головна мова інтерфейсу – українська та англійська. Англійська мова використовується для показу результатів з сповіщень про негоду.

Вебсторінки системи: головна сторінка (рис. 1.1), сторінка реєстрації, сторінка профілю, сторінка всіх менеджера користувачів, сторінка поточної погоди, сторінка прогнозу погоди, сторінка карт з радарів та супутника, сторінка сповіщень про небезпечну погоду, сторінка аналітики клімату, сторінка підписки на розсилку сповіщень та промоакцій від менеджера.

Для авторизованих користувачів з’являється можливість зміни локації для керування системою. Для користувачів з преміум акаунтом є можливість завантаження звітів та підписку на розсилку сповіщень та повідомлень. Вміст даних елементів змінюється в залежності від того, авторизований користувач чи ні та на якій вебсторінці він знаходиться.

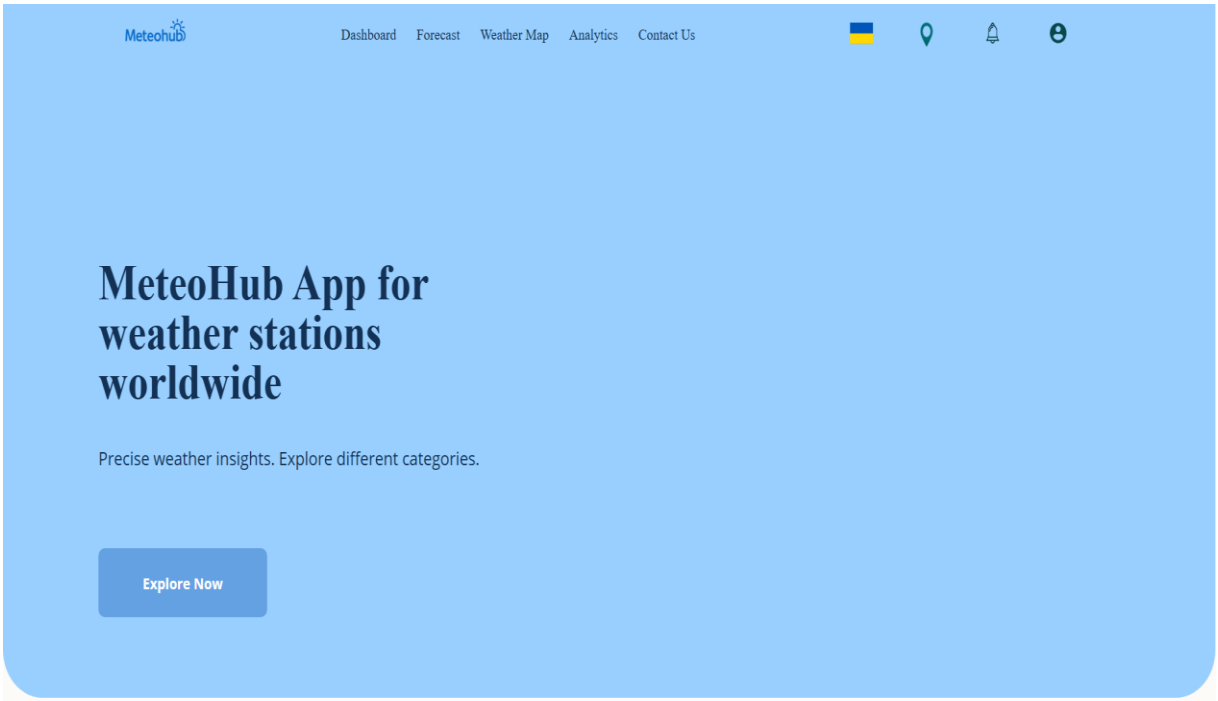


Рис. 1.1. Головна сторінка вебзастосунку

2. ПРОЦЕДУРА РЕЄСТРАЦІЇ ТА АВТОРИЗАЦІЇ

Для повноцінного використання можливостей вебзастосунку користувач повинен пройти процедуру реєстрації та авторизації. Форма реєстрації має наступні поля для вводу: електронна пошта користувача, ім’я та прізвища (дозволяються тільки букви), пароль (не менше 4 символів). Електронна пошта унікальна для кожного користувача. У випадку співпадіння з’явиться помилка реєстрації (рис. 2.1).

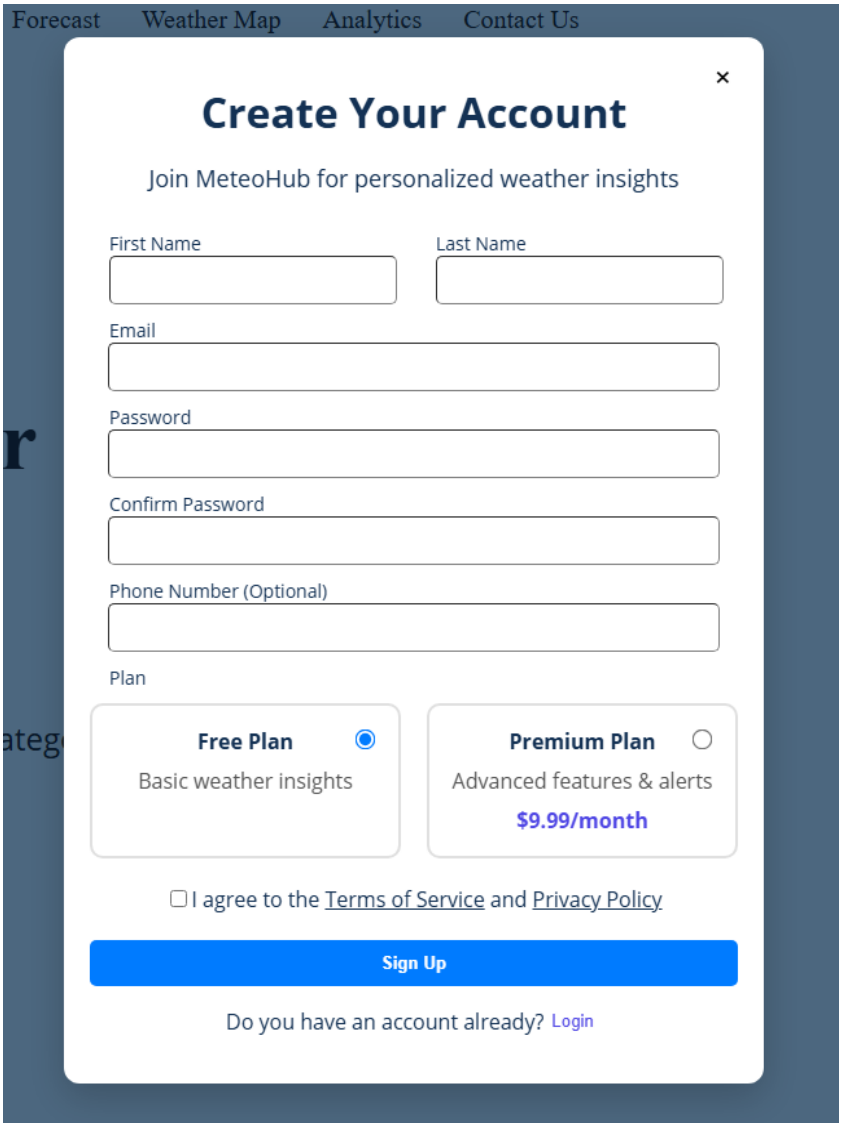


Рис. 2.1. Модальне вікно реєстрації

Для входу в систему користувач заповнює поля електронної пошти та пароля у формі авторизації. Якщо дані не співпадають, з’явиться помилка авторизації (рис. 2.2).

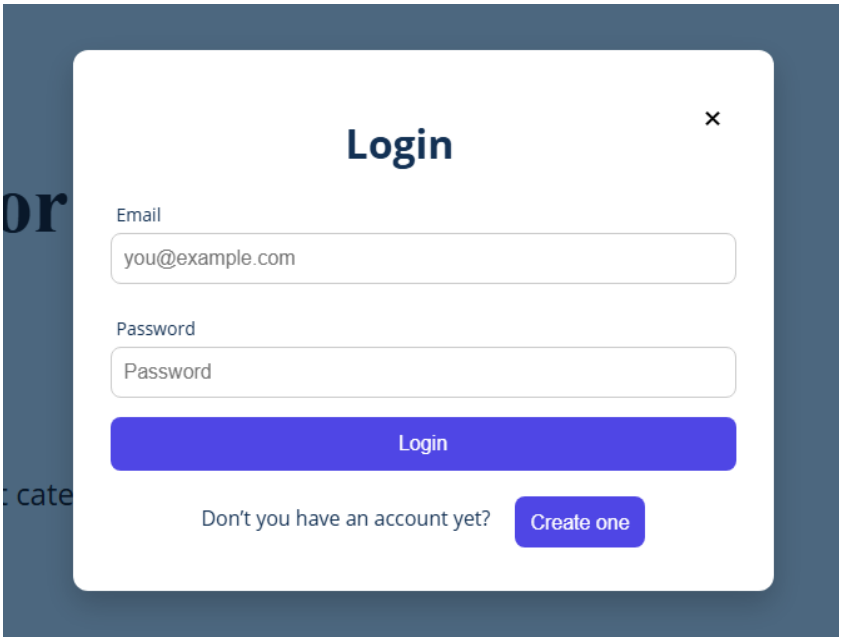


Рис. 2.2. Модальне вікно авторизації

Користувач має свій кабінет, де може редагувати свої дані (рис. 2.3).

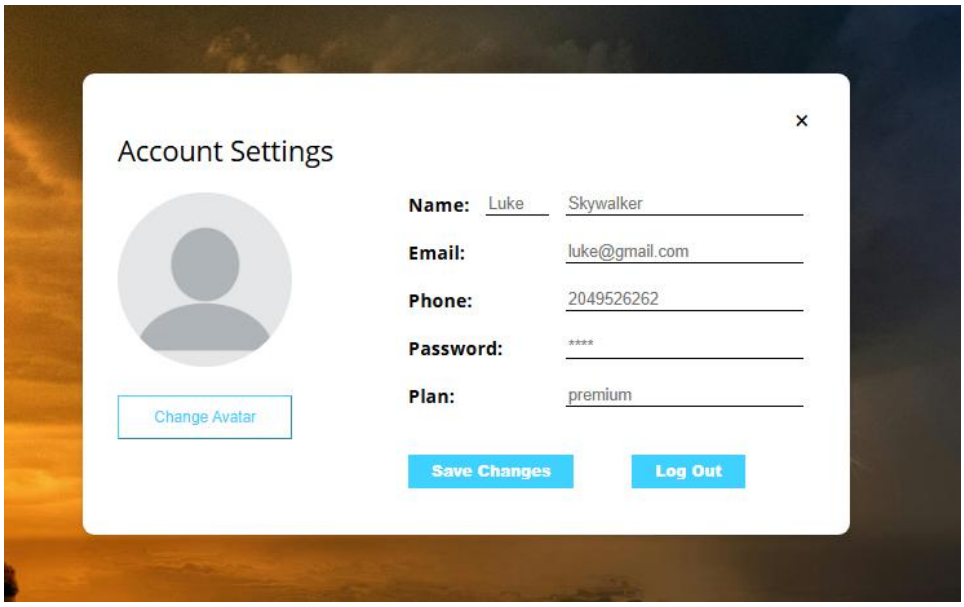


Рис. 2.3. Сторінка користувача

3. ВИДИ ПІДПИСОК ТА НАЛАШТУВАННЯ ЇХ ОПЦІЙ

Для користування вебзастосунок надає два види підписки.

3.1. Безкоштовна підписка

Дозволяє користувачу переглядати інформацію для конкретної локації. Авторизований користувач вводить назву міста у модальному вікні (рис. 3.1).

Рис. 3.1. Модальне вікно локації

Користувач з безкоштовною підпискою має обмежений функціонал та не може переглядати детальну інформацію по поточній погоді (рис. 3.2), завантажувати звіти (рис. 3.3) та оформити підписку на розсилку (рис. 3.4).

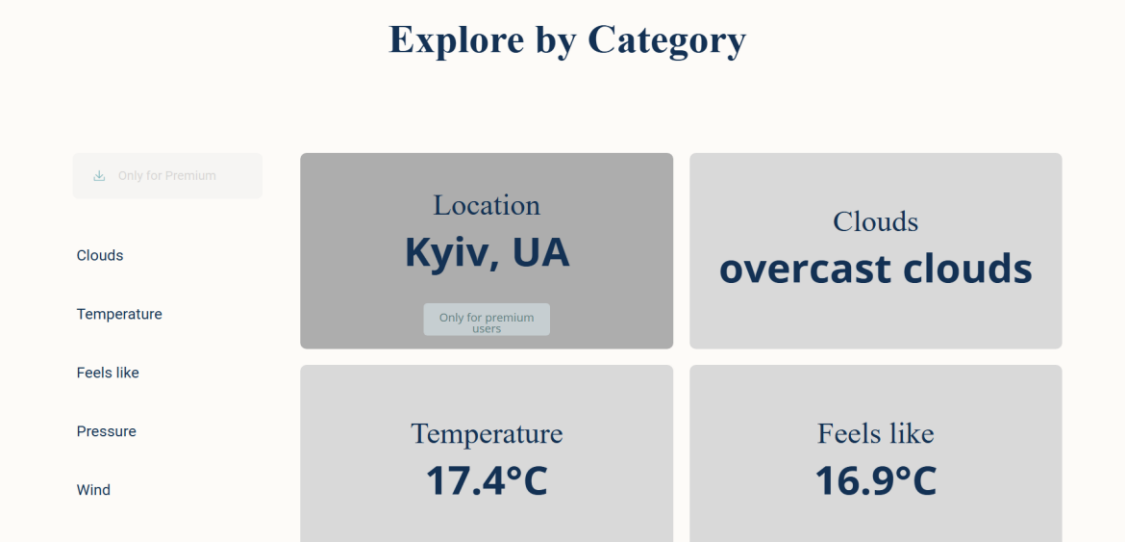


Рис. 3.2. Авторизований користувач у секції поточної погоди

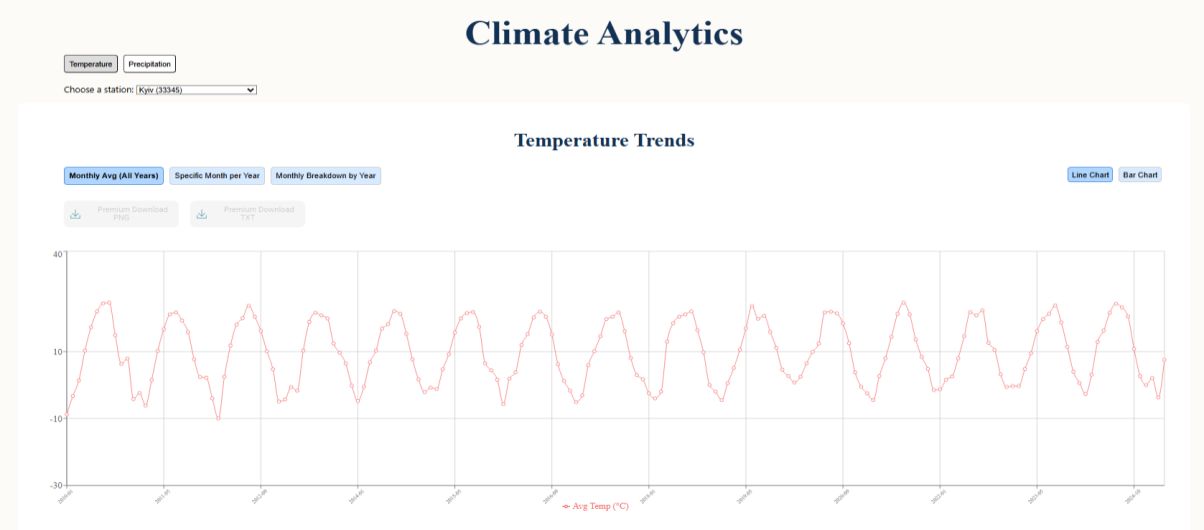


Рис. 3.3. Авторизований користувач у секції аналізу клімату

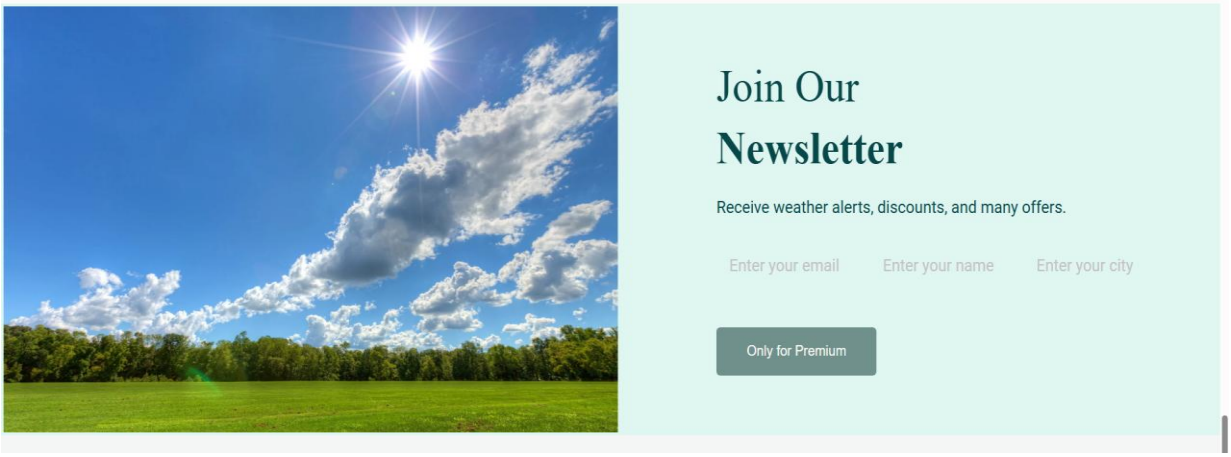


Рис. 3.4. Авторизований користувач у секції підписки на розсилку

3.2. Преміум підписка

Користувач з преміум підпискою має повний функціонал та може переглядати детальну інформацію по поточній погоді (рис. 3.5), завантажувати звіти (рис. 3.6) та оформити підписку на розсилку (рис. 3.7).

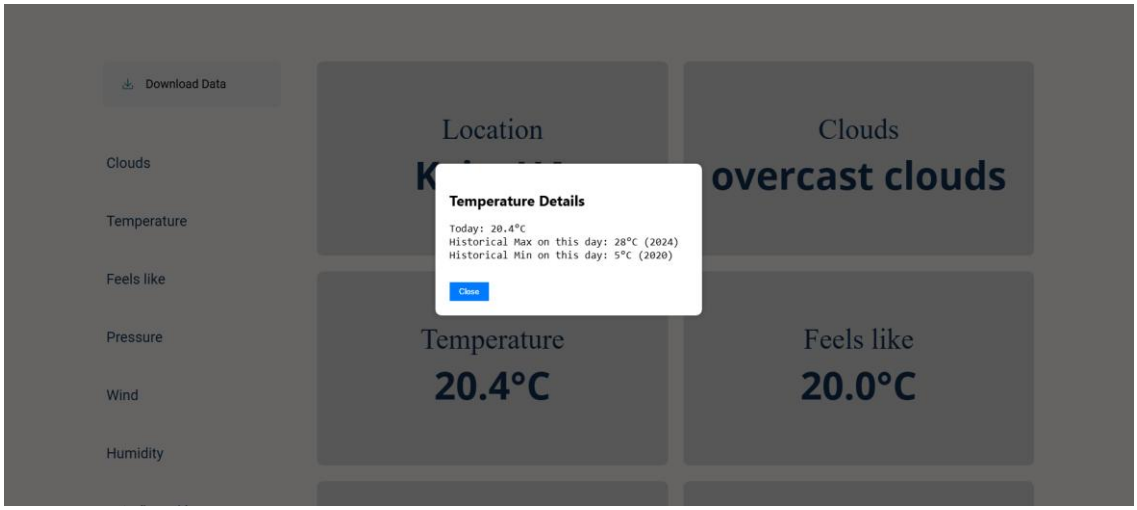


Рис. 3.5. Преміум користувач у секції поточної погоди

```
Downloaded on: Thursday, May 22, 2025, 09:28:07 PM

Weather Categories for Kyiv

Location: Kyiv, UA
Clouds: overcast clouds
Temperature: 20.4°C
Feels like: 20.0°C
Pressure: 1013 hPa
Humidity: 58%
Wind: Speed 0.45 m/s, Gust 1 m/s, Direction 141°
Rain (last 3h): 0 mm
Sunrise/Sunset: Sunrise at 05:01, Sunset at 20:48
```

Рис. 3.6. Звіт завантажений преміум користувачем

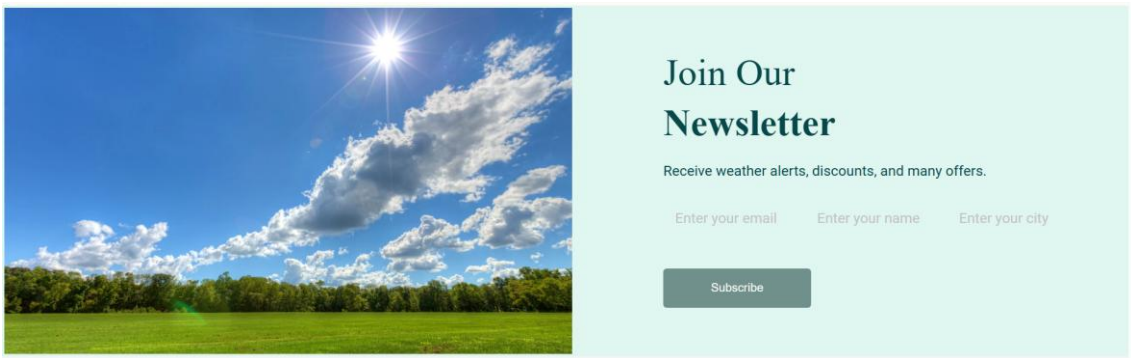


Рис. 3.7. Преміум користувач у секції підписки на розсилку

3.3. Гість

Дозволяє користувачу переглядати інформацію для локації Київ. Користувач вводить назву міста у модальному вікні, а потім переноситься на вікно реєстрації (рис. 3.8).

Гість має обмежений функціонал та не може переглядати детальну інформацію по поточній погоді (рис. 3.2), завантажувати звіти (рис. 3.9) та оформити підписку на розсилку (рис. 3.4).

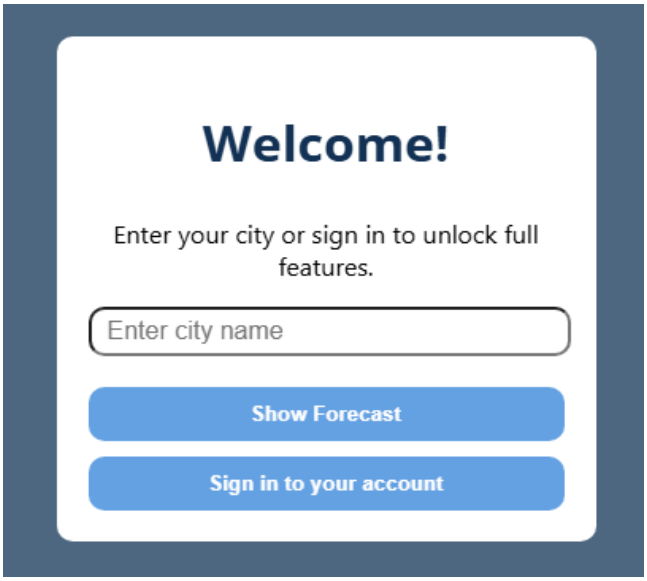


Рис. 3.8. Модальне вікно локації для гостя

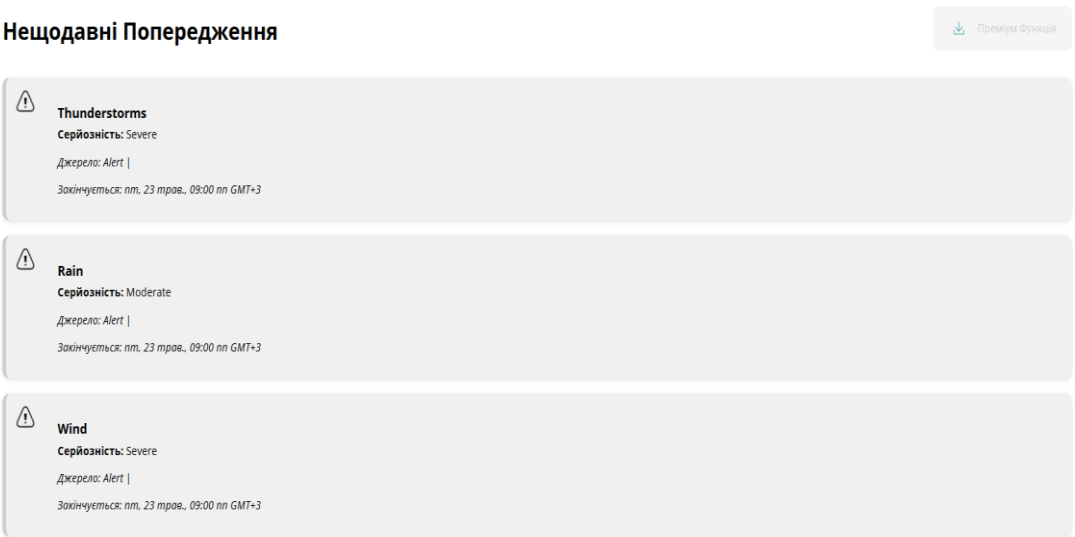


Рис. 3.9. Панель сповіщень для локації Київ для гостя

4. НАЛАШТУВАННЯ РОБОЧОГО ПРОЦЕСУ ДЛЯ ПЕРСОНАЛІЗОВАНОГО ВИКОРИСТАННЯ

4.1. Налаштування мови

Вебзастосунок Meteohub дозволяє користувачу змінювати мову додатку на українську, хоча основною мовою додатку є англійська. Переклад був здійснений для усієї сторінки.

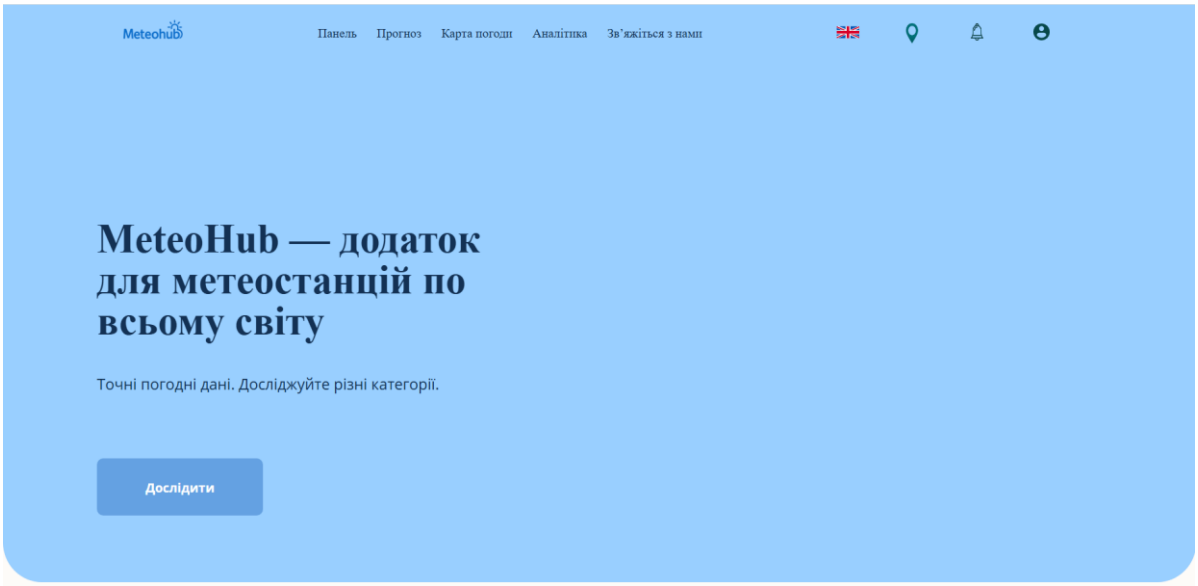


Рис. 4.1. Перекладена на українську головна сторінка

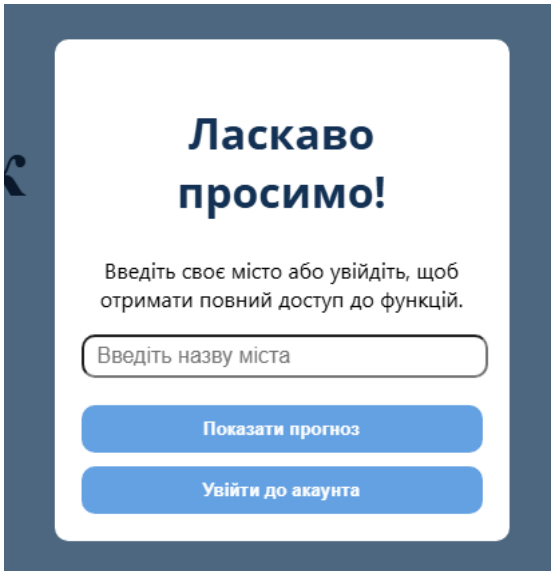


Рис. 4.2. Перекладене на українську модальне вікно зміни локації

4.2. Налаштування локації

Налаштування локації для користувача (рис. 4.3). Користувач може змінювати локацію отриманих погодних результатів (рис. 4.4).

Рис. 4.3. Секція поточної погоди для локації Вінніпег, Канада

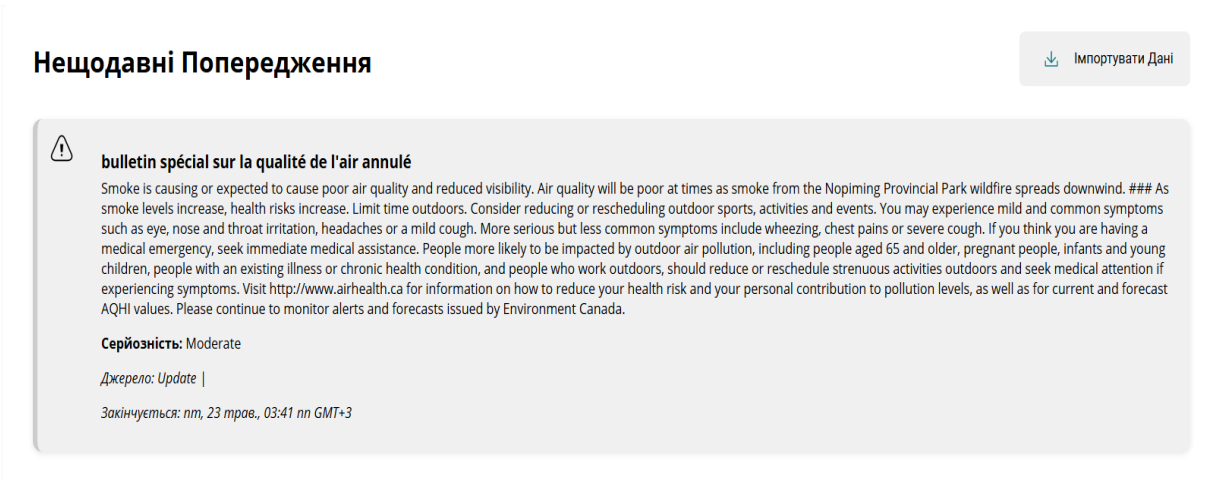


Рис. 4.4. Секція сповіщення про негоду для локації Вінніпег, Канада

4.3. Налаштування аналітики клімату

Користувач може змінювати метеостанцію (рис. 4.5), з якої будуть отримуватись дані про аналітику клімату (рис. 4.6).

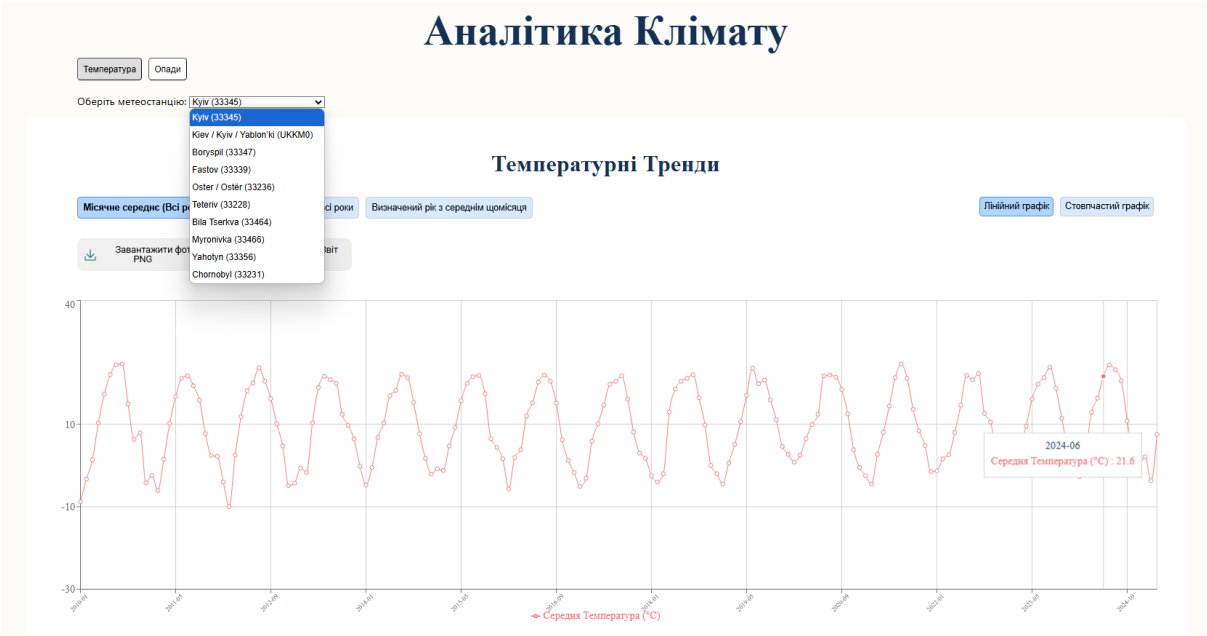


Рис. 4.5. Секція аналітики клімату для метеостанції Kyiv (33345)

Рис. 4.6. Секція аналітики клімату для метеостанції Kiev/Kyiv/Yablon’ki (UKKM0)

Користувач також може змінювати вигляд графіку на лінійчастий та стовпчастий (рис. 4.7). Користування фільтрами (рис. 4.8): місячне середнє (всі роки), порівняння визначеного місяця за всі роки, визначений рік з

середнім його місячним значенням. Також можна переглянути сумарне значення опадів кожного місяця за всі роки (рис. 4.9).

Рис. 4.7. Секція аналітики клімату з фільтром Порівняння визначеного місяця за всі роки

Рис. 4.8. Секція аналітики клімату з фільтром Визначений рік з середнім його місячним значенням

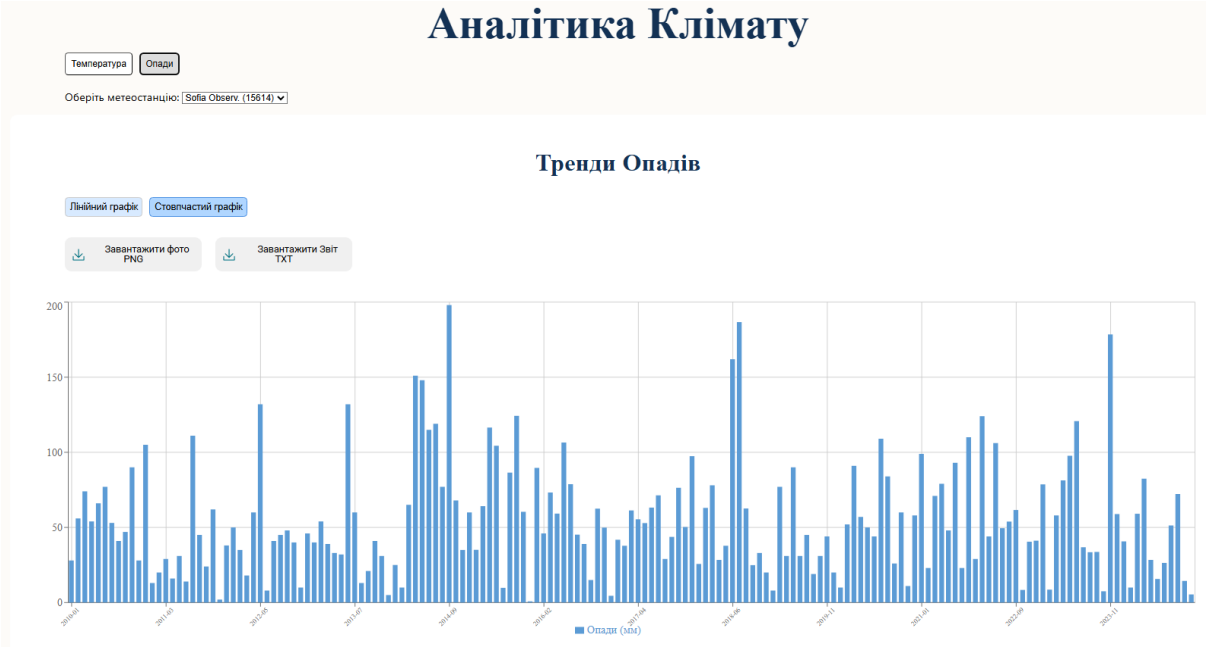


Рис. 4.9. Секція аналітики клімату з сумарним значенням опадів кожного місяця за всі роки

4.4. Перегляд різних погодних мап

Користувач може перемикати мапи погоди (рис. 4.10) та отримати маркер зі своєю вказаною локацією та значенням параметру відповідної мапи.

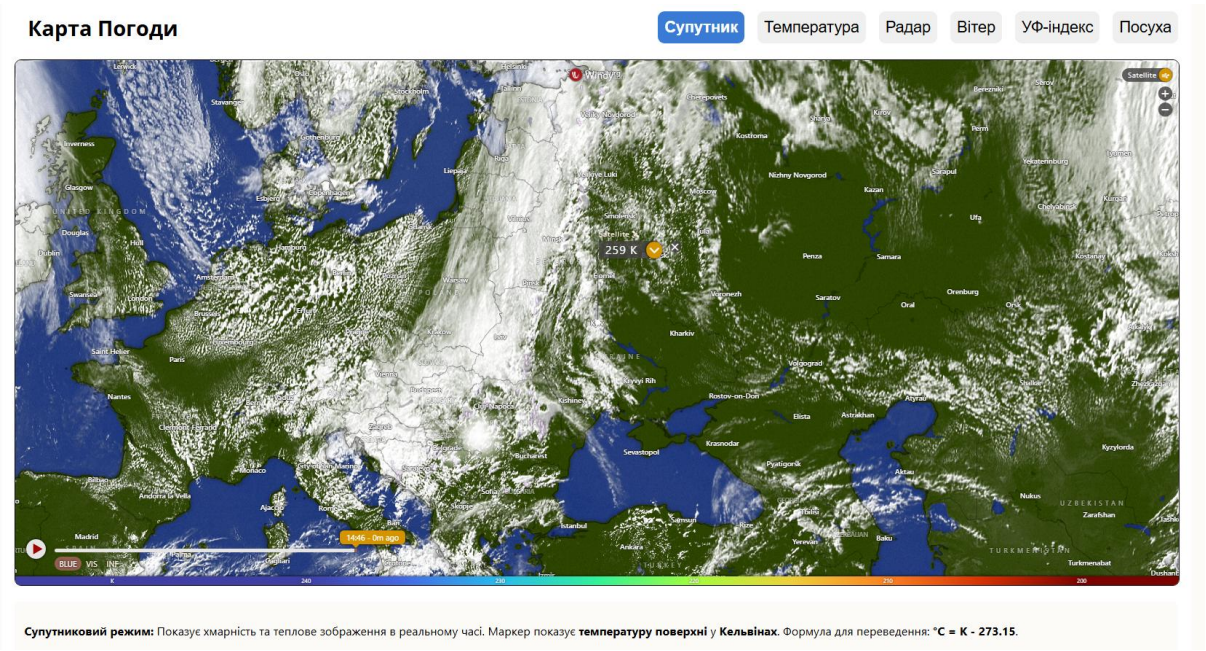


Рис. 4.10. Карта знімку з супутника

Супутниковий режим (рис. 4.11): показує хмарність та теплове зображення в реальному часі. Маркер показує температуру поверхні у Кельвінах. Формула для переведення: °C = K - 273.15.

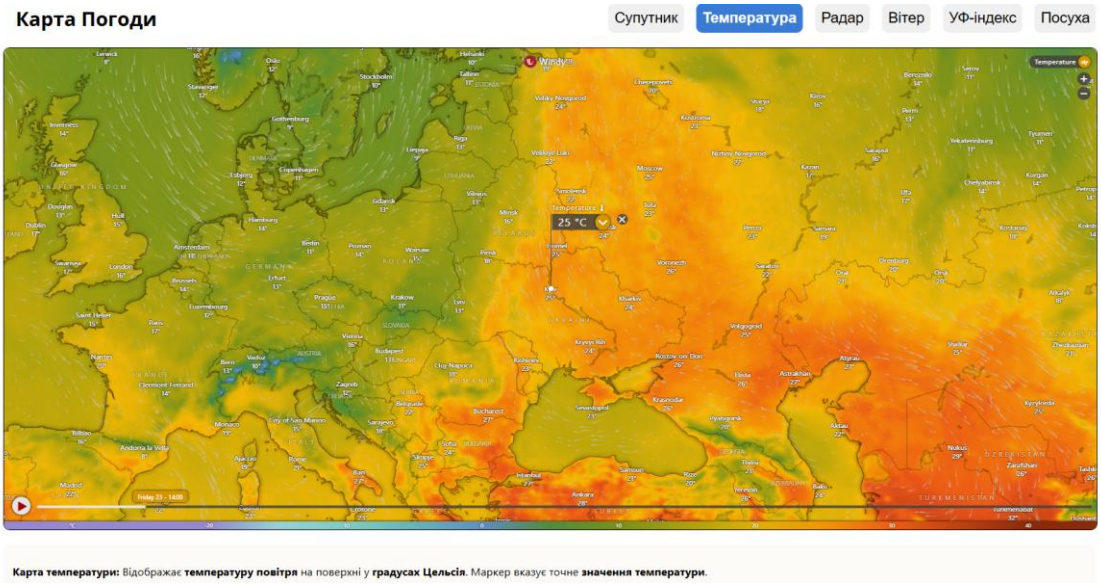


Рис. 4.11. Карта температури

Карта температури (рис. 4.12): відображає температуру повітря на поверхні у градусах Цельсія. Маркер вказує точне значення температури.

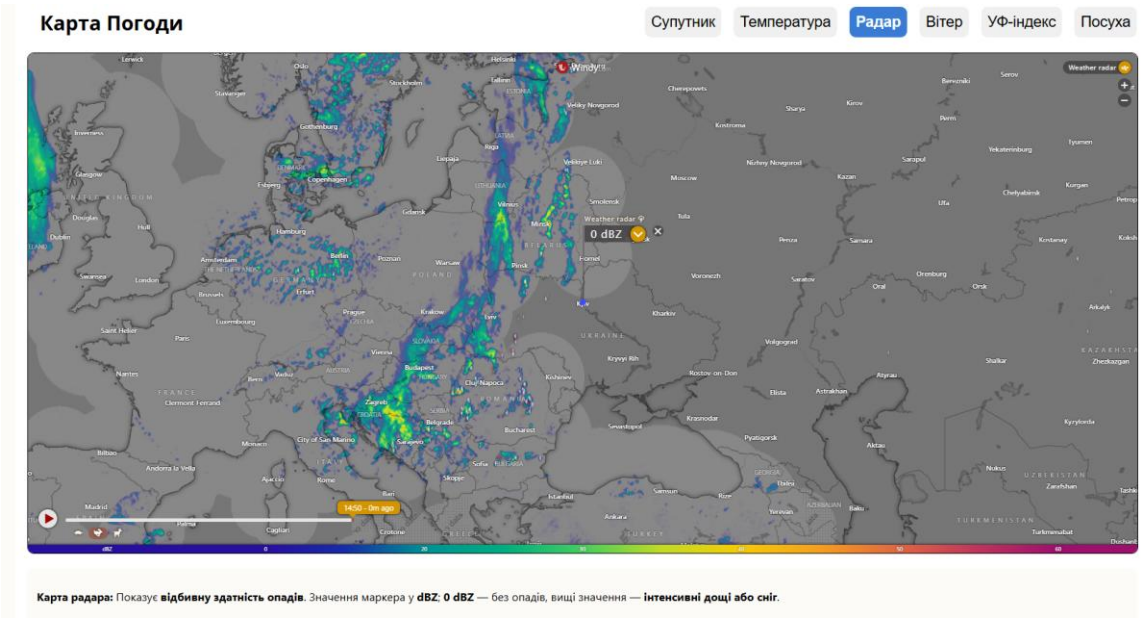


Рис. 4.12. Карта радара

Карта радара (рис. 4.13): показує відбивну здатність опадів. Значення маркера у dBZ; 0 dBZ – без опадів, вищі значення – інтенсивні дощі або сніг.

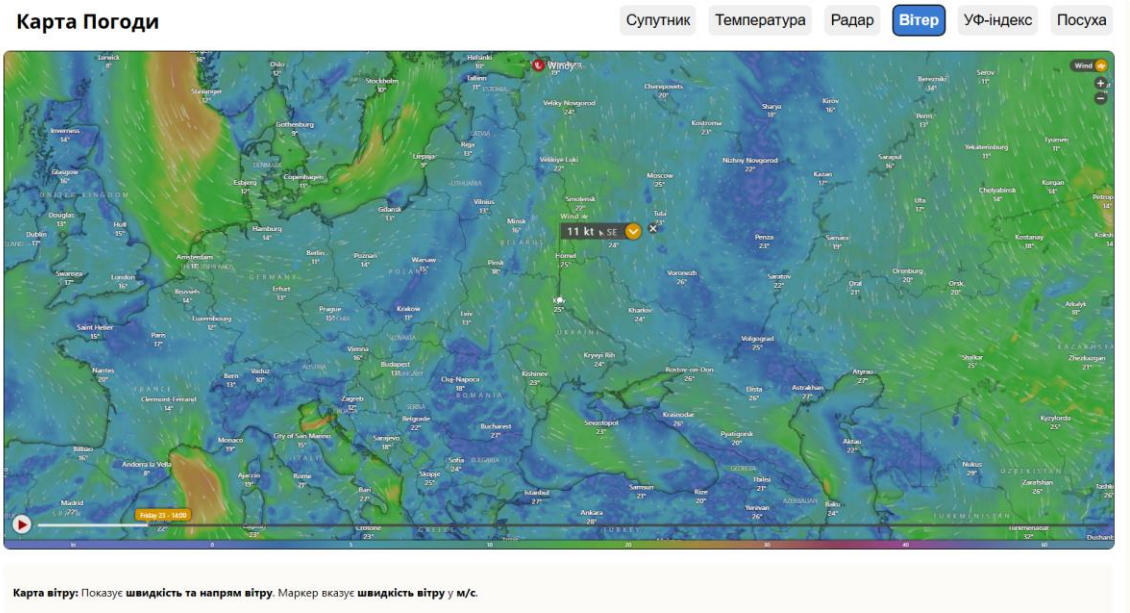


Рис. 4.13. Карта вітру

Карта вітру (рис. 4.14): показує швидкість та напрям вітру. Маркер вказує швидкість вітру у м/с.

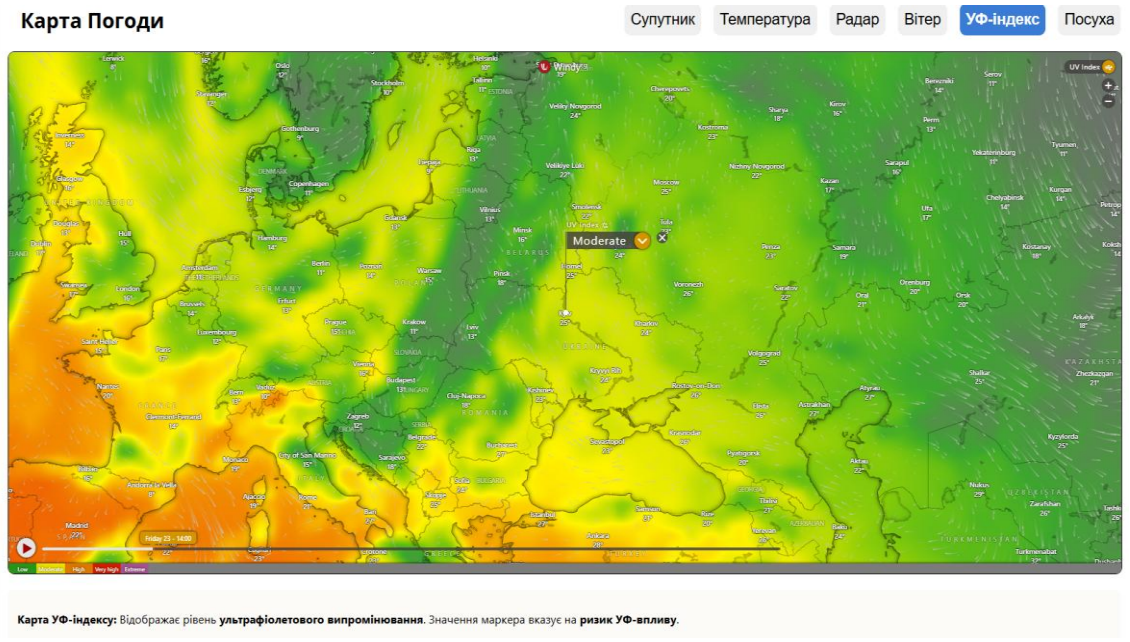


Рис. 4.14. Карта УФ-індексу

Карта УФ-індексу (рис. 4.15): відображає рівень ультрафіолетового випромінювання. Значення маркера вказує на ризик УФ-впливу.

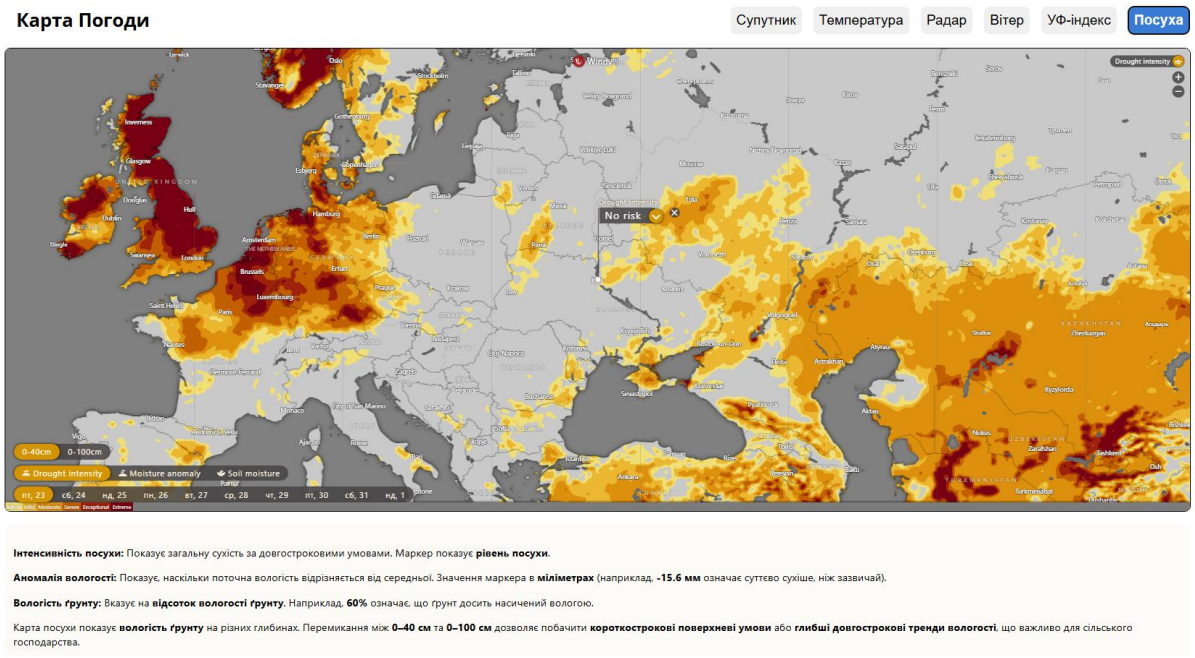


Рис. 4.15. Карта посух

Інтенсивність посухи: показує загальну сухість за довгостроковими умовами. Маркер показує рівень посухи.

Аномалія вологості: показує, наскільки поточна вологість відрізняється від середньої. значення маркера в міліметрах (-15.6 мм означає суттєво сухіше, ніж зазвичай).

Вологість ґрунту: вказує на відсоток вологості ґрунту. Наприклад, 60% означає, що ґрунт досить насичений вологою.

Карта посухи показує вологість ґрунту на різних глибинах. Перемикання між 0–40 см та 0–100 см дозволяє побачити короткострокові поверхневі умови або глибші довгострокові тренди вологості, що важливо для сільського господарства.

5. ПЕРЕГЛЯД РЕЗУЛЬТАТІВ ЗВІТІВ ТА СПОВІЩЕНЬ ПРО ПОГОДНІ УМОВИ

Користувач може завантажувати звіти про поточні погодні умови (рис. 5.1 та рис. 5.2), нещодавні попередження про негоду (рис. 5.5), звіт аналітики клімату (рис. 5.4) та знімок графіку (рис. 5.3).

```
Downloaded on: Friday, May 23, 2025, 02:59:02 PM

Weather Categories for toronto

Location: Toronto, CA
Clouds: overcast clouds
Temperature: 8.1°C
Feels like: 4.6°C
Pressure: 1008 hPa
Humidity: 91%
Wind: Speed 6.69 m/s, Gust NaN m/s, Direction 290°
Rain (last 3h): 0 mm
Sunrise/Sunset: Sunrise at 12:44, Sunset at 03:44
```

Рис. 5.1. Звіт про поточну погоду в локації Торонто

```
Downloaded on: Friday, May 23, 2025, 03:01:48 PM

Weather Alerts for kyiv

Alert 1:
Headline: Thunderstorms
Description:
Severity: Severe
Source: Alert |
Expires: 23.05.2025, 21:00:00

Alert 2:
Headline: Rain
Description:
Severity: Moderate
Source: Alert |
Expires: 23.05.2025, 21:00:00

Alert 3:
Headline: Wind
Description:
Severity: Severe
Source: Alert |
Expires: 23.05.2025, 21:00:00
```

Рис. 5.2. Звіт про сповіщення про негоду в локації Київ

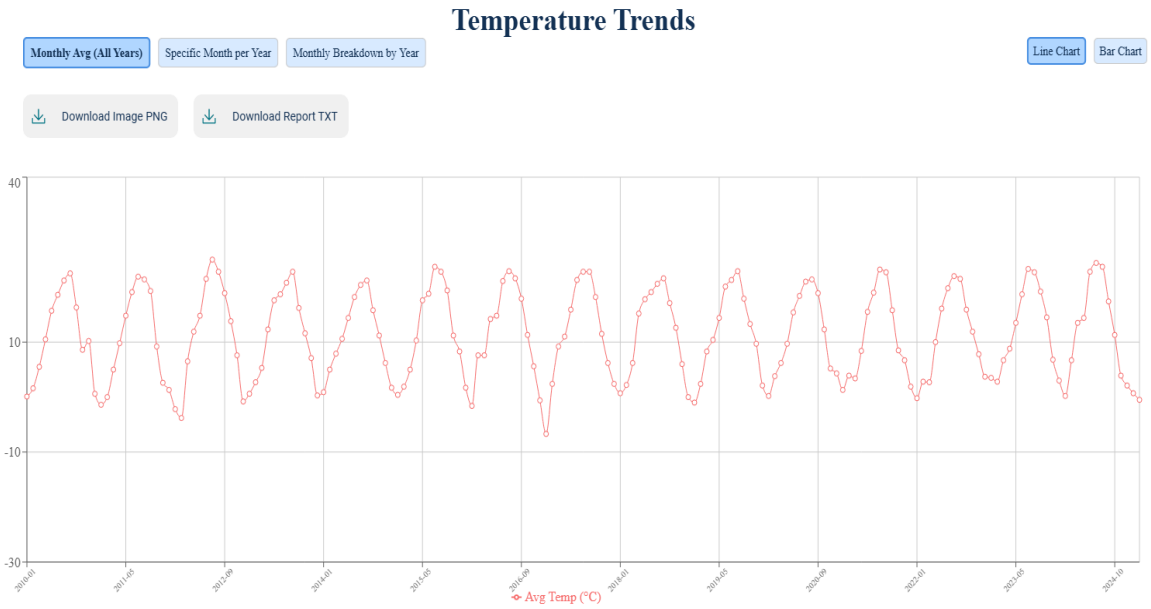


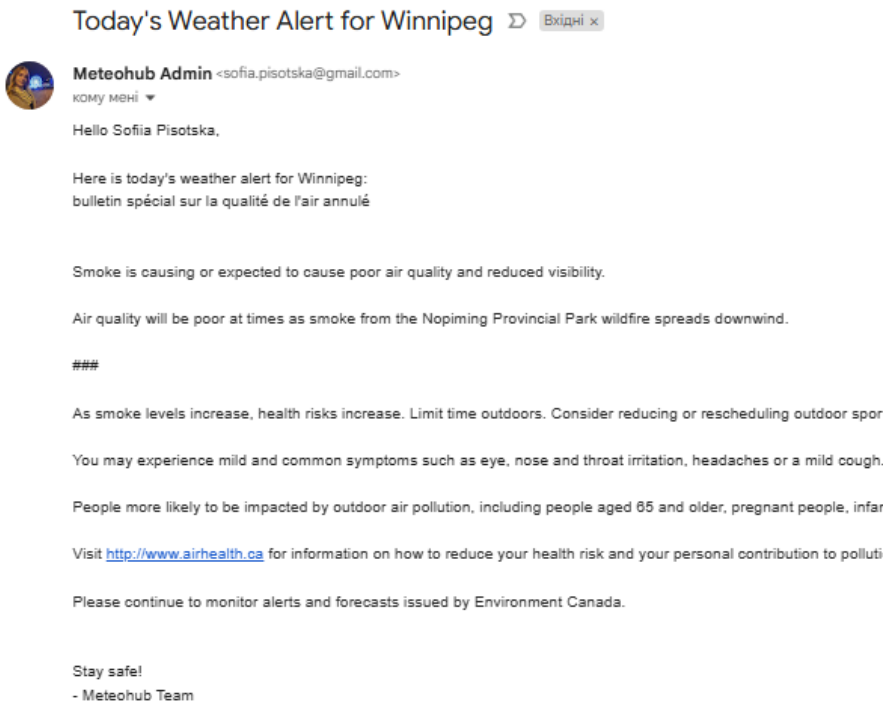
Рис. 5.3. Фотозвіт про аналітику клімату в локації Софія

```
Climate Report for sofia

Chart: Temperature
View Mode: monthPerYear
Selected Month: 01

Date            Value
--------------------------
2010    0.1
2011    -1.4
2012    -2.2
2013    0.6
2014    0.9
2015    0.4
2016    -1.6
2017    -6.7
2018    0.7
2019    -1
2020    0.2
2021    1.3
2022    -0.2
2023    3.5
2024    0.2
2025    0.7
```

Рис. 5.4. Звіт про аналітику клімату для січня в локації Софія


Today's Weather Alert for Winnipeg Вхідні ×

Meteohub Admin <sofia.pisotska@gmail.com>
кому мені

Hello Sofiia Pisotska,

Here is today's weather alert for Winnipeg:
bulletin spécial sur la qualité de l'air annulé

Smoke is causing or expected to cause poor air quality and reduced visibility.

Air quality will be poor at times as smoke from the Nopiming Provincial Park wildfire spreads downwind.

###

As smoke levels increase, health risks increase. Limit time outdoors. Consider reducing or rescheduling outdoor spor

You may experience mild and common symptoms such as eye, nose and throat irritation, headaches or a mild cough.

People more likely to be impacted by outdoor air pollution, including people aged 65 and older, pregnant people, infar

Visit http://www.airhealth.ca for information on how to reduce your health risk and your personal contribution to polluti

Please continue to monitor alerts and forecasts issued by Environment Canada.

Stay safe!
- Meteohub Team

Рис. 5.5. Повідомлення для підписника розсилки сповіщень про негоду у локації Вінніпег