Міністерство освіти і науки України
Криворізький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерних систем та мереж

Пояснювальна записка
до кваліфікаційної роботи бакалавра
за спеціальністю 123 «Комп'ютерна інженерія»

на тему: Розробка веб-додатку для візуалізації аналітичних даних з інтеграцією зовнішніх API

Проектував _____ Д. М. Безземяний

Керівник роботи _____ Д. І. Кузнєцов

Нормоконтроль _____ Д. І. Кузнєцов

Завідувач кафедри _____ А. І. Купін

Кривий Ріг
2026

Криворізький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерних систем та мереж

Ступінь вищої освіти — бакалавр
Спеціальність — 123 «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри, голова циклової комісії

_____ А. І. Купін

"____" _____ 20__ року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

_____Безземяному Дмитру Миколайовичу_____
(прізвище, ім'я, по батькові)

1. Тема роботи ____Розробка веб-додатку для візуалізації аналітичних даних з інтеграцією зовнішніх API_____
_____
керівник роботи ___Кузнєцов Денис Іванович, канд. техн. наук, доцент_____,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від "27"січня 2026 року №64с
2. Строк подання студентом роботи ____01.06.2026_____
3. Вихідні дані до роботи – метеорологічні дані, що отримуються із зовнішніх API сервісів погоди; назва населеного пункту або географічні координати місцевості; показники температури повітря, атмосферного тиску, вологості, швидкості та напрямку вітру; прогноз погодних умов на декілька днів; технології розробки: React, JavaScript, Redux Toolkit, Axios, CSS; засоби візуалізації даних: Recharts; середовище розробки Visual Studio Code._____
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)___Вступ; 1 Аналіз предметної області та сучасних технологій реалізації веб-додатків для візуалізації метеоданих; 2 Проєктування веб-додатку для візуалізації метеорологічних даних; 3 Реалізація веб-додатку для візуалізації метеорологічних даних; Висновки; Список використаних джерел; Додатки._____
_____
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)____20 рисунків; 6 таблиць; 10 слайдів презентації PDF._____
_____
_____

_____

_____

7. Дата видачі завдання_____01.02.2026_____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічних джерел за темою кваліфікаційної роботи	01.03.2026 – 10.03.2026	
2	Оформлення титульної сторінки, завдання та змісту кваліфікаційної роботи	11.03.2026 – 14.03.2026	
3	Написання вступу (актуальність, мета, завдання, об'єкт та предмет дослідження)	15.03.2026 – 20.03.2026	
4	Написання Розділу 1. Аналіз предметної області, сучасних погодних сервісів та технологій веб-розробки	21.03.2026 – 08.04.2026	
5	Проєктування архітектури веб-додатку, розробка структурних схем та написання Розділу 2	09.04.2026 – 24.04.2026	
6	Реалізація веб-додатку засобами React, інтеграція зовнішнього API та тестування функціоналу	25.04.2026 – 11.05.2026	
7	Написання Розділу 3. Опис реалізації програмного продукту, підготовка рисунків та додатків	12.05.2026 – 17.05.2026	
8	Формування загальних висновків, переліку скорочень, реферату українською та англійською мовами	18.05.2026 – 21.05.2026	
9	Оформлення списку використаних джерел, перевірка відповідності вимогам нормоконтролю	22.05.2026 – 24.05.2026	
10	Підготовка презентації та доповіді до передзахисту/захисту кваліфікаційної роботи	25.05.2026 – 29.05.2026	
11	Попередній захист кваліфікаційної роботи	31.05.2026	
12	Усунення зауважень та остаточне оформлення роботи	01.06.2026 – 03.06.2026	
13	Захист кваліфікаційної роботи	25.06.2026	

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка: 58 сторінок, 20 рисунків, 6 таблиць, 12 використаних джерел

Мета роботи – дослідити сучасні технології розробки веб-додатків для візуалізації аналітичних даних, розробити веб-додаток для відображення метеорологічної інформації з використанням зовнішніх API та реалізувати засоби графічного представлення погодних показників.

Проєкт складається з трьох розділів.

У першому розділі проведено аналіз предметної області метеорологічних інформаційних систем. Розглянуто сучасні погодні сервіси, принципи роботи зовнішніх API, особливості SPA-додатків та бібліотек візуалізації даних. Виконано порівняльний аналіз популярних погодних платформ та інструментів розробки веб-додатків.

У другому розділі виконано проєктування веб-додатку. Сформовано технічне завдання, визначено функціональні та нефункціональні вимоги до системи, розроблено архітектуру програмного продукту, структуру компонентів та алгоритми взаємодії із зовнішнім API. Також описано модулі обробки даних та візуалізації інформації.

У третьому розділі реалізовано веб-додаток засобами React. Розроблено модулі пошуку населених пунктів, отримання погодних даних, формування прогнозу погоди та відображення аналітичної інформації. Проведено тестування основних функціональних можливостей системи та перевірено коректність її роботи.

У результаті виконання роботи розроблено веб-додаток для візуалізації метеорологічних даних, який забезпечує отримання актуальної інформації із зовнішніх API та її представлення у зручному графічному вигляді.

ВЕБ-ДОДАТОК, REACT, API, ВІЗУАЛІЗАЦІЯ ДАНИХ, МЕТЕОРОЛОГІЧНІ ДАНІ, JAVASCRIPT, REDUX TOOLKIT, AXIOS, RECHARTS, SPA-ДОДАТОК.

Explanatory note: 58 pages, 20 figures, 6 tables, 12 references

The purpose of the work is to investigate modern technologies for developing web applications for analytical data visualization, design a web application for displaying meteorological information using external APIs, and implement graphical tools for presenting weather indicators.

The project consists of three chapters.

The first chapter analyzes the subject area of meteorological information systems. Modern weather services, principles of external APIs, features of SPA applications, and data visualization libraries are considered. A comparative analysis of popular weather platforms and web development technologies is carried out.

The second chapter is devoted to the design of the web application. Functional and non-functional requirements are defined, the software architecture is developed, component interaction is described, and algorithms for communication with external APIs are designed. Data processing and visualization modules are also specified.

The third chapter describes the implementation of the web application using React. Modules for city search, weather data retrieval, weather forecasting, chart generation, and analytical data presentation are developed. Functional testing of the system is performed and the correctness of its operation is verified.

As a result, a web application for meteorological data visualization has been developed. The application provides real-time weather information from external APIs and presents it in an interactive and user-friendly graphical format.

WEB APPLICATION, REACT, API, DATA VISUALIZATION, METEOROLOGICAL DATA, JAVASCRIPT, REDUX TOOLKIT, AXIOS, RECHARTS, SPA APPLICATION.

ЗМІСТ

ЗМІСТ 6
ПЕРЕЛІК СКОРОЧЕНЬ 8
ВСТУП 9
1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА СУЧАСНИХ ТЕХНОЛОГІЙ РЕАЛІЗАЦІЇ ВЕБ-ДОДАТКІВ ДЛЯ ВІЗУАЛІЗАЦІЇ МЕТЕОДАНИХ 10
1.1 Особливості метеорологічних інформаційних систем 10
1.2 Аналіз сучасних веб-технологій для створення SPA-додатків 13
1.3 Бібліотеки для візуалізації даних 16
1.4 Аналіз структури та формату метеорологічних API 17
Висновки за розділом 19
2. ПРОЄКТУВАННЯ ВЕБ-ДОДАТКУ ДЛЯ ВІЗУАЛІЗАЦІЇ МЕТЕОРОЛОГІЧНИХ ДАНИХ 20
2.1 Формування технічного завдання 20
2.1.1 Функціональні вимоги 20
2.1.2 Нефункціональні вимоги 20
2.1.3 Обмеження системи 20
2.2 Архітектура веб-додатку 21
2.2.1 Загальна архітектурна модель 21
2.2.2 Основні компоненти системи 22
2.2.3 Управління станом 22
2.2.4 Опис основних змінних та структур даних 22
2.2.5 Опис основних методів системи 24
2.3 Проєктування інтерфейсу користувача 25
2.3.1 Основні елементи інтерфейсу 26
2.3.2 Принципи UX/UI 29
2.3.2 Адаптивність 29
2.4 Проєктування взаємодії з API 30

2.4.1 Логіка виконання запитів 30
2.4.2 Обробка даних 31
2.4.3 Обробка помилок 31
Висновки за розділом 31
3. РЕАЛІЗАЦІЯ ВЕБ-ДОДАТКУ ДЛЯ ВІЗУАЛІЗАЦІЇ МЕТЕОРОЛОГІЧНИХ ДАНИХ 32
3.1 Загальна структура реалізованого веб-додатку 32
3.2 Загальна структура реалізованого веб-додатку 33
3.3 Реалізація модуля відображення поточної погоди 34
3.4 Реалізація модуля прогнозу погоди 35
3.5 Реалізація системи управління станом 36
3.6 Тестування реалізованого функціоналу 36
Висновок за розділом 39
ВИСНОВОК 40
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 41
ДОДАТКИ 43
Додаток А 43
Додаток Б 45
Додаток В 53
Додаток Г 56

ПЕРЕЛІК СКОРОЧЕНЬ

API - Application Programming Interface (прикладний програмний інтерфейс)

DOM - Document Object Model (об'єктна модель документа)

HTTP - HyperText Transfer Protocol (протокол передачі гіпертексту)

JSON - JavaScript Object Notation (нотація об'єктів JavaScript)

REST - Representational State Transfer (передача репрезентативного стану)

SPA - Single Page Application (односторінковий сайт)

UI - User Interface (інтерфейс користувача)

UX - User Experience (досвід користувача)

ВСТУП

У сучасному цифровому суспільстві обсяг інформації постійно зростає, що зумовлює потребу в ефективних інструментах її обробки та візуалізації. Особливої актуальності набувають веб-додатки аналітичного спрямування, які забезпечують швидкий доступ до структурованих даних та їх наочне представлення у вигляді графіків, діаграм і інформаційних панелей.

Метеорологічна інформація є важливою складовою повсякденного життя людини та використовується в багатьох галузях: транспорті, аграрному секторі, туризмі, енергетиці, логістиці та інших сферах. Оперативний доступ до погодних даних та можливість аналізу їх змін у часі дозволяє підвищити ефективність прийняття рішень.

З розвитком веб-технологій та поширенням концепції API-економіки стало можливим інтегрувати сторонні джерела даних у власні інформаційні системи. Використання REST API дозволяє отримувати актуальні метеорологічні показники у режимі реального часу, що створює передумови для розробки сучасних інтерактивних веб-додатків.

Актуальність теми дипломної роботи зумовлена необхідністю створення зручних та функціональних веб-інструментів для візуалізації аналітичних метеорологічних даних із використанням сучасних frontend-технологій.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА СУЧАСНИХ ТЕХНОЛОГІЙ РЕАЛІЗАЦІЇ ВЕБ-ДОДАТКІВ ДЛЯ ВІЗУАЛІЗАЦІЇ МЕТЕОДАНИХ

1.1 Особливості метеорологічних інформаційних систем

У сучасному цифровому світі метеорологічна інформація є важливою для повсякденного життя користувачів, а також для професійних завдань у сферах транспорту, сільського господарства, енергетики, туризму та логістики. Програмні комплекси, які забезпечують збір, обробку, збереження та візуалізацію погодних даних, називають метеорологічними інформаційними системами. [1].

Основні функції метеорологічних систем:

- отримання даних з метеостанцій та супутників;
- формування прогнозів;
- збереження історичних показників;
- аналітика кліматичних тенденцій;
- візуалізація показників у зручному для користувача форматі.

Основні типи метеоданих:

- температура повітря;
- атмосферний тиск;
- вологість;
- швидкість та напрямок вітру;
- кількість опадів;
- індекс УФ-випромінювання;
- хмарність [2].

Багато веб-платформ пропонують доступ до метеорологічних даних. Популярними серед них є WeatherAPI, OpenWeather і AccuWeather. Проведемо детальний огляд і порівняльний аналіз кожного з сервісів.

1. OpenWeather

OpenWeather є одним із найпопулярніших сервісів для отримання погодних даних, який застосовується як у навчальних, так і в комерційних цілях. Ця платформа забезпечує глобальні метеорологічні дані в режимі реального часу, короткострокові та довгострокові прогнози, а також історичну кліматичну інформацію. Джерелами даних слугують метеорологічні станції, супутникові системи та числові моделі прогнозування. [3].

Основні можливості API:

- поточна погода;
- прогноз на 5–7 днів;
- погодні параметри (температура, вологість, тиск, вітер);
- погодні карти;
- історичні дані (у платних тарифах).

Переваги:

- широкий географічний охват;
- зрозумілий формат відповіді JSON;
- наявність безкоштовного тарифного плану;
- проста інтеграція з веб-додатками.

Недоліки:

- обмеження кількості запитів у безкоштовному плані;
- деякі розширені функції доступні лише за платною підпискою.

2. AccuWeather

AccuWeather — це міжнародна компанія, яка спеціалізується на точному прогнозуванні погоди та пропонує професійні метеорологічні рішення. Сервіс відзначається високою точністю прогнозів завдяки використанню власних алгоритмів моделювання атмосферних явищ. Його API розроблено для задоволення потреб як комерційних організацій, так і мобільних додатків. [4].

Основні можливості API:

- поточна погода;
- погодинний прогноз;
- прогноз на 10–15 днів;
- попередження про небезпечні погодні явища;
- індекси (UV-індекс, відчутна температура тощо).

Переваги:

- висока точність прогнозування;
- розширені погодні показники;
- підтримка багатьох мов.

Недоліки:

- обмежений безкоштовний доступ;
- складніша структура API;
- комерційна орієнтованість сервісу.

3. WeatherAPI

WeatherAPI — це сучасний сервіс, який пропонує зручний і детально задокументований API для доступу до погодних даних. Він орієнтований на розробників і відзначається простою структурою запитів і відповідей. WeatherAPI ідеально підходить для невеликих проєктів, стартапів і навчальних завдань. [5].

Основні можливості API:

- поточна погода;
- прогноз на 7–14 днів;
- погодинний прогноз;

- історичні дані;
- підтримка форматів JSON та XML.

Переваги:

- проста інтеграція;
- зрозуміла документація;
- зручна структура відповіді;
- достатній безкоштовний ліміт.

Недоліки:

- обмеження кількості запитів у безкоштовному плані;
- частина розширених функцій доступна лише у платній версії.

Таблиця 1.1 - Порівняльний аналіз сервісів

Критерій	OpenWeather	AccuWeather	WeatherAPI
Наявність безкоштовного плану	Так	Обмежений	Так
Простота інтеграції	Висока	Середня	Висока
Обсяг доступних даних	Широкий	Дуже широкий	Широкий
Точність прогнозів	Висока	Дуже висока	Висока

За кваліфікацією прогнози поділяються на:

1) Поточна погода (Current Weather)
2) Короткостроковий прогноз (1–7 днів)
3) Довгостроковий прогноз (до 30 днів)
4) Історичні кліматичні дані

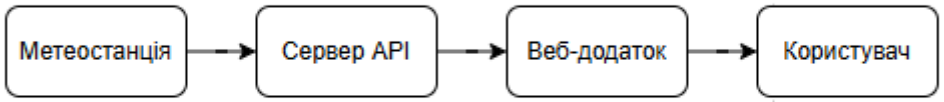

Рисунок 1.1 – Загальна схема роботи метеорологічними системами

1.2 Аналіз сучасних веб-технологій для створення SPA-додатків

Сучасні веб-додатки для аналітики та візуалізації даних здебільшого створюються за допомогою архітектурної моделі SPA (Single Page Application). Цей підхід забезпечує інтерактивність інтерфейсів із динамічним оновленням інформації без необхідності повного перезавантаження сторінки [6]. У разі розробки веб-додатку для відображення метеорологічних даних використання SPA-архітектури є виправданим, оскільки погодні показники змінюються в режимі реального часу, а інтерфейс має миттєво реагувати на вибір користувача, наприклад, міста чи періоду прогнозу.

SPA-додаток завантажується в браузері лише один раз, після чого обмін з сервером відбувається через асинхронні HTTP-запити. Інформація передається у форматі JSON, а оновлення інтерфейсу здійснюється за допомогою JavaScript [7].

Основні характеристики SPA:

- динамічне оновлення контенту;
- відсутність повного перезавантаження сторінки;
- швидша взаємодія з користувачем;
- розділення клієнтської та серверної логіки.

Переваги SPA для погодного додатку:

- миттєве перемикання між містами;
- оновлення графіків без перезавантаження;
- зменшення навантаження на сервер;
- кращий користувацький досвід (UX).

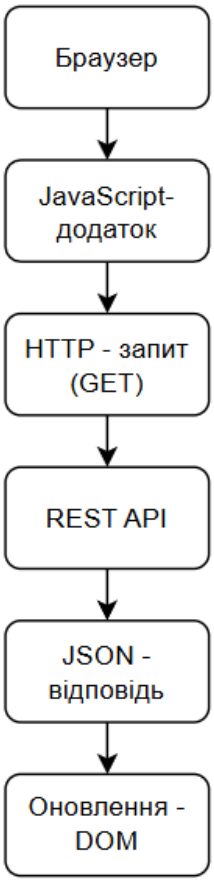

Рисунок 1.2 – Загальна схема SPA – архутектури

Серед найпопулярніших технологій для створення SPA-додатків виділяють:

- React
- Angular
- Vue.js

Розглянемо їх особливості окремо.

1. React

React — це бібліотека JavaScript для розробки користувацьких інтерфейсів, створена компанією Meta. Вона використовує компонентний підхід, що дає змогу створювати багаторазово застосовувані UI-компоненти [8].

Ключові особливості:

- Virtual DOM;
- компонентна структура;
- Hooks (useState, useEffect тощо);
- висока продуктивність.

React не є повним фреймворком. React потребує додаткових бібліотек для маршрутизації і HTTP-запитів.

2. Angular

Angular — це повноцінний frontend-фреймворк, який надає комплексне рішення для розробки SPA-додатків [9].

Особливості:

- TypeScript за замовчуванням;
- вбудована маршрутизація;
- строгі правила структури проєкту;
- dependency injection.

Angular підходить для великих корпоративних систем, однак має вищий поріг входу.

3. Vue.js

Vue.js — це прогресивний фреймворк, що поєднує простоту використання з достатньою гнучкістю [10].

Особливості:

- двостороннє зв’язування даних;
- простий синтаксис;
- швидке навчання;
- компактність.

Vue є зручним для невеликих і середніх проєктів.

Таблиця 1.2 - Порівняльна характеристика технологій

Критерій	React	Angular	Vue.js
Тип	Бібліотека	Фреймворк	Фреймворк
Поріг входу	Середній	Високий	Низький
Гнучкість	Висока	Середня	Висока
Масштабованість	Висока	Дуже висока	Висока
Підходить для аналітики	Так	Так	Так

Для реалізації веб-додатку візуалізації метеорологічних даних було обрано React з огляду на такі фактори [11]:

1) Компонентна архітектура

Дозволяє розділити інтерфейс на логічні частини:

- панель пошуку;
- блок поточної погоди;
- прогноз на 7 днів;
- модуль графіків.

2) Virtual DOM

Забезпечує оптимізацію оновлення інтерфейсу, що особливо важливо при побудові графіків.

3) Гнучкість інтеграції з бібліотеками візуалізації

React легко інтегрується з бібліотеками графіків.

4) Популярність та підтримка спільноти
Велика кількість документації та прикладів.
5) Можливість використання Hooks
Спрощує управління станом та життєвим циклом компонентів.

1.3 Бібліотеки для візуалізації даних

Візуалізація даних є ключовим елементом аналітичного веб-додатку. Графічне представлення інформації дозволяє швидко оцінити тенденції та закономірності.

Популярні бібліотеки для візуалізації:

- Recharts [12];
- Chart.js [13];
- D3.js [14].

Таблиця 1.3 – порівняння бібліотек

Критерій	Recharts	Chart.js	D3.js
Інтеграція з React	Відмінна	Середня	Складна
Гнучкість	Висока	Середня	Дуже висока
Складність	Низька	Низька	Висока

Для реалізації погодного додатку доцільно використати Recharts, оскільки вона оптимізована для React та дозволяє швидко створювати інтерактивні графіки температур, вологості та швидкості вітру.

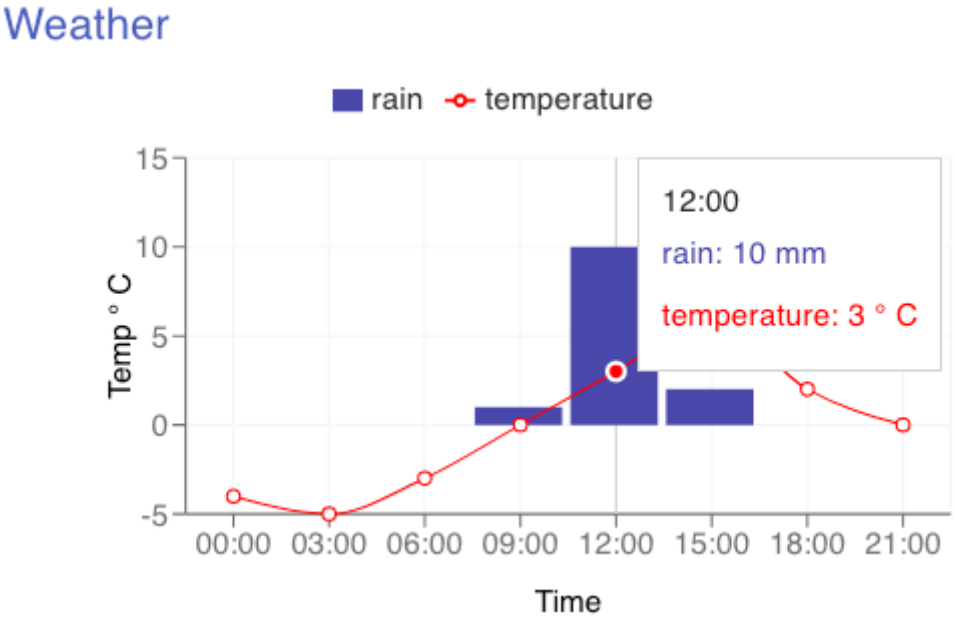

Рисунок 1.3 — Приклад графіка температури

1.4 Аналіз структури та формату метеорологічних API

Для створення веб-додатку, що відображає метеорологічні дані, необхідно звертатись до зовнішніх джерел інформації. Найбільш розповсюдженим методом отримання таких даних є застосування REST API (інтерфейсу програмування додатків), який забезпечує зв'язок між клієнтською частиною додатку та сервером, де зберігаються дані [15].

У цьому розділі описані основні принципи функціонування REST API, формат передачі метеоданих, особливості аутентифікації, а також технічні нюанси впровадження у SPA-додаток.

REST (Representational State Transfer) — це архітектурний підхід для розробки веб-сервісів, що базується на протоколі HTTP.

У випадку погодного додатку взаємодія відбувається за такою схемою:

1. Користувач вводить назву міста.
2. Додаток формує HTTP-запит.
3. Сервер API обробляє запит.
4. Повертається відповідь у форматі JSON.
5. Дані відображаються у вигляді графіків та інформаційних блоків.

Основні HTTP-методи [16]**:**

- GET — отримання даних;
- POST — передача даних на сервер;
- PUT — оновлення ресурсів;
- DELETE — видалення ресурсів.

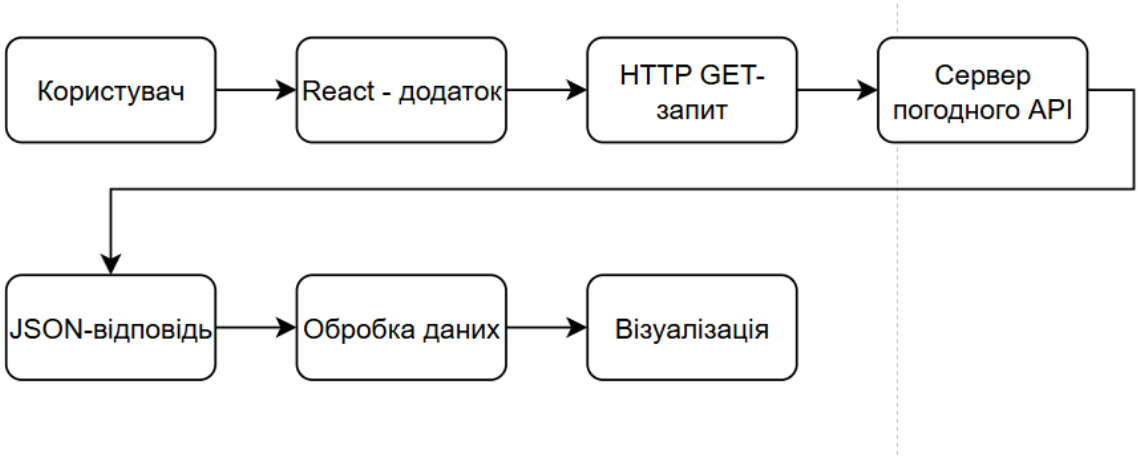

Рисунок 1.4 – Схема взаємодії клієнта з API

Більшість сучасних API використовують формат JSON (JavaScript Object Notation), який є текстовим, компактним та легко обробляється у JavaScript-середовищі [17].

Приклад JSON:

```
{
  "location": {
```

```
    "name": "Kyiv",
    "country": "Ukraine"
  },
  "current": {
    "temp_c": 17.4,
    "humidity": 65,
    "wind_kph": 12.3,
    "condition": {
      "text": "Partly cloudy"
    }
  }
}
```

Основні характеристики JSON:

- структурованість (ключ-значення);
- підтримка вкладених об’єктів;
- легкість парсингу;
- компактність передачі даних.

У React-додатку JSON перетворюється у JavaScript-об’єкти, після чого значення використовуються для побудови інтерфейсу та графіків [18].

Типовий запит до погодного API має такий вигляд:

`https://api.weatherprovider.com/current?city=Kyiv&units=metric&apikey=XXXX`

Де:

- city — назва міста;
- units — одиниці вимірювання (metric/imperial);
- apikey — унікальний ключ доступу;
- lang — мова відповіді (за потреби).

Apikey - є унікальним ідентифікатором користувача, що дозволяє контролювати кількість запитів та забезпечує захист від несанкціонованого використання сервісу [19].

Також під час роботи з API можливі різні типи помилок які потрібно обробляти. Код помилки та причина його виникнення буде наведена у таблиці 1.4 [20].

Таблиця 1.4 – Типи API помилок

Код	Причина
400	Некоректний запит
401	Невірний API-ключ
404	Місто не знайдено
429	Перевищено ліміт
500	Помилка сервера

Висновки за розділом

У першому розділі дипломної роботи здійснено всебічний аналіз предметної області метеорологічних інформаційних систем та сучасних технологій, які застосовуються для розробки аналітичних веб-додатків. Розглянуто особливості роботи погодних сервісів і принципи надання метеорологічних даних через зовнішні API. Проведено порівняння популярних платформ для отримання погодних даних, що дозволило обґрунтувати доцільність інтеграції API-сервісів для забезпечення актуальної інформації у веб-додатку.

Досліджено архітектурні характеристики SPA-додатків і виконано порівняльний аналіз сучасних frontend-технологій. Вибір React аргументовано його компонентною структурою, високою продуктивністю та широкими можливостями для інтеграції.

Крім того, проведено огляд популярних бібліотек для візуалізації даних і визначено доцільність використання спеціалізованих інструментів для створення інтерактивних графіків. Особливу увагу приділено структурі та формату передачі даних через REST API, а також питанням автентифікації, обробки помилок і оптимізації запитів.

2. ПРОЄКТУВАННЯ ВЕБ-ДОДАТКУ ДЛЯ ВІЗУАЛІЗАЦІЇ МЕТЕОРОЛОГІЧНИХ ДАНИХ

2.1 Формування технічного завдання

Проєктування інформаційної системи починається з розробки технічного завдання, яке визначає функціональні можливості, вимоги до системи та обмеження її реалізації. Метою розробки є створення веб-додатку, який використовує зовнішні API для відображення та аналізу метеорологічних даних.

2.1.1 Функціональні вимоги

Головні функції системи включають:
- пошук міста за назвою;
- отримання й відображення актуальної погоди;
- показ прогнозу погоди на кілька днів вперед;
- відображення додаткових параметрів, таких як вологість, вітер і тиск;
- адаптацію інтерфейсу під різні типи пристроїв.

2.1.2 Нефункціональні вимоги

Система повинна відповідати таким нефункціональним вимогам:
- швидкодія — мінімальний час завантаження;
- надійність — коректна обробка помилок API;
- зручність використання (UX);
- масштабованість — можливість розширення функціоналу;
- кросбраузерність;
- адаптивність.

2.1.3 Обмеження системи

- залежність від зовнішнього API;
- обмеження кількості запитів;
- необхідність використання API-ключа;
- робота тільки за наявності інтернет-з’єднання.

2.2 Архітектура веб-додатку

Для впровадження системи обрана архітектура SPA (Single Page Application), при якій основна обробка та відображення даних відбуваються на стороні клієнта.

2.2.1 Загальна архітектурна модель

Рисунок 2.1 — Архітектура системи

2.2.2 Основні компоненти системи

Архітектура додатку спирається на компонентний підхід.
Ключові компоненти включають:
- SearchBar — введення назви міста;
- WeatherCard — відображення поточних погодних умов;
- ForecastList — прогноз погоди на кілька днів;
- WeatherChart — графічне представлення температури;
- Sidebar — навігаційна панель (за потребою).

2.2.3 Управління станом

Для зберігання даних застосовуються:
- локальний стан компонентів (useState);
- глобальні дані передаються через props або Context API.

Основні стани включають:
- погодні дані;
- стан завантаження (loading);
- обробка помилок (error);
- вибране місто.

2.2.4 Опис основних змінних та структур даних

Для забезпечення роботи веб-додатку використовується набір змінних стану, які відповідають за зберігання отриманих метеорологічних даних та керування інтерфейсом користувача.

Таблиця 2.1 - Основні змінні системи

Змінна	Тип даних	Призначення
city	String	Назва міста для пошуку
weatherData	Object	Дані поточної погоди
forecastData	Array	Масив прогнозу на декілька днів
loading	Boolean	Стан завантаження даних
error	String	Повідомлення про помилку

Приклад структури weatherData:

```
{
    "coord": {
        "lon": 33.3918,
        "lat": 47.9103
    },
```

```
    "weather": [
        {
            "id": 803,
            "main": "Clouds",
            "description": "broken clouds",
            "icon": "04n"
        }
    ],
    "base": "stations",
    "main": {
        "temp": 282.32,
        "feels_like": 281.15,
        "temp_min": 282.32,
        "temp_max": 282.32,
        "pressure": 1012,
        "humidity": 83,
        "sea_level": 1012,
        "grnd_level": 1002
    },
    "visibility": 10000,
    "wind": {
        "speed": 2.28,
        "deg": 338,
        "gust": 3.17
    },
    "clouds": {
        "all": 75
    },
    "dt": 1780272713,
    "sys": {
        "country": "UA",
        "sunrise": 1780278715,
        "sunset": 1780335413
    },
    "timezone": 10800,
    "id": 707663,
    "name": "Artema",
    "cod": 200
}
```


Приклад структури forecastData:


```
[{
    "cod": "200",
    "message": 0,
    "cnt": 40,
    "list": [
```

```
{
    "dt": 1780282800,
    "main": {
        "temp": 19.68,
        "feels_like": 19.79,
        "temp_min": 18.33,
        "temp_max": 19.68,
        "pressure": 1016,
        "sea_level": 1016,
        "grnd_level": 1007,
        "humidity": 80,
        "temp_kf": 1.35
    },
    "weather": [
        {
            "id": 800,
            "main": "Clear",
            "description": "clear sky",
            "icon": "01n"
        }
    ],
    "clouds": {
        "all": 2
    },
    "wind": {
        "speed": 0.73,
        "deg": 65,
        "gust": 1.21
    },
    "visibility": 10000,
    "pop": 0,
    "sys": {
        "pod": "n"
    },
    "dt_txt": "2026-06-01 03:00:00"
}]
```


2.2.5 Опис основних методів системи

Логіка веб-додатку реалізована через набір функцій та методів, що забезпечують взаємодію із зовнішнім API та обробку отриманих даних.

Метод отримання координат міста

```
export const fetchData = createAsyncThunk(
  "weather/fetchData",
```

```
      async (city) => {
        try {
          const geoResponse = await axios.get(
            `http://api.openweathermap.org/geo/1.0/dire
ct?q=${city}&limit=1&appid=${API_KEY}`
          );

          const location = geoResponse.data[0];

          const lat = location.lat;
          const lon = location.lon;

          const weatherResponse = await axios.get(
            `https://api.openweathermap.org/data/2.5/we
ather?lat=${lat}&lon=${lon}&units=metric&appid=${API_KE
Y}`
          );
          const weatherForecast = await axios.get(
            `https://api.openweathermap.org/data/2.5/fo
recast?lat=${lat}&lon=${lon}&units=metric&appid=${API_K
EY}`
          );
          return {
            weather: weatherResponse.data,
                cityName:  location.local_names?.uk  ||
location.name,
            forecast: weatherForecast.data,
          };
        } catch (error) {
          console.log(error);
        }
      }
    );
```

Призначення:

- виконує пошук міста;
- отримує географічні координати;
- передає координати для наступних запитів.
- отримує актуальні погодні данні та данні на 5 днів.

2.3 Проєктування інтерфейсу користувача

Інтерфейс відіграє ключову роль у системі, оскільки від нього залежить комфортність і зручність користувача під час взаємодії з додатком.

2.3.1 Основні елементи інтерфейсу

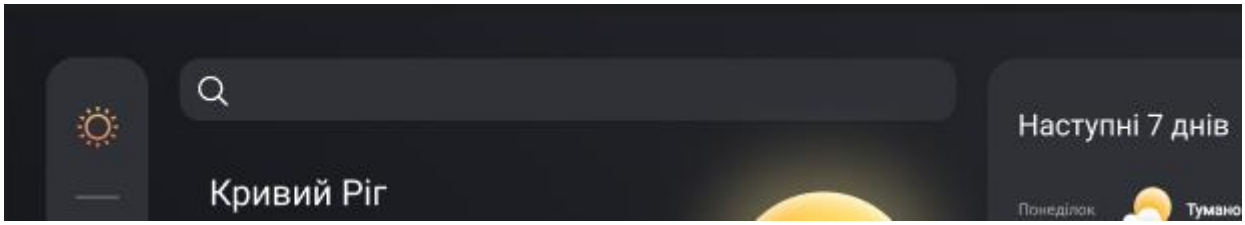

Рисунок 2.2 - Пошук міста за назвою

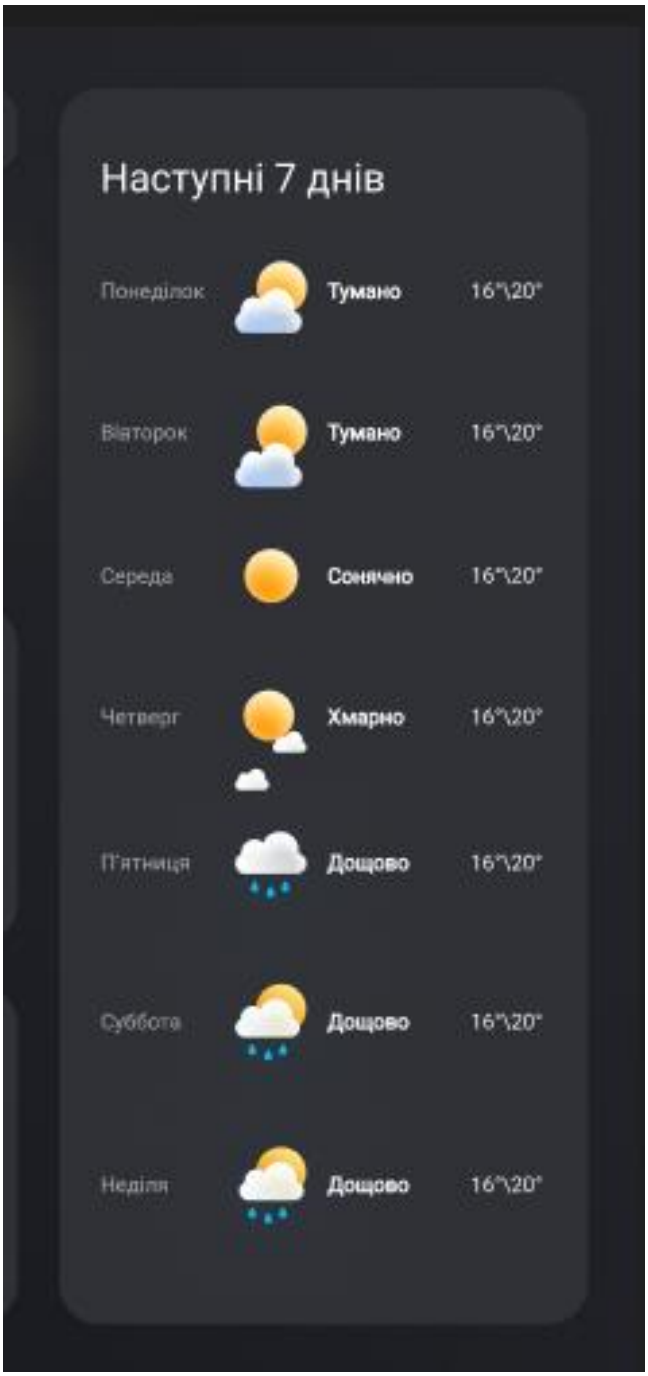

Рисунок 2.3 - Показ прогнозу погоди на кілька днів вперед

Рисунок 2.4 - Отримання й відображення актуальної погоди

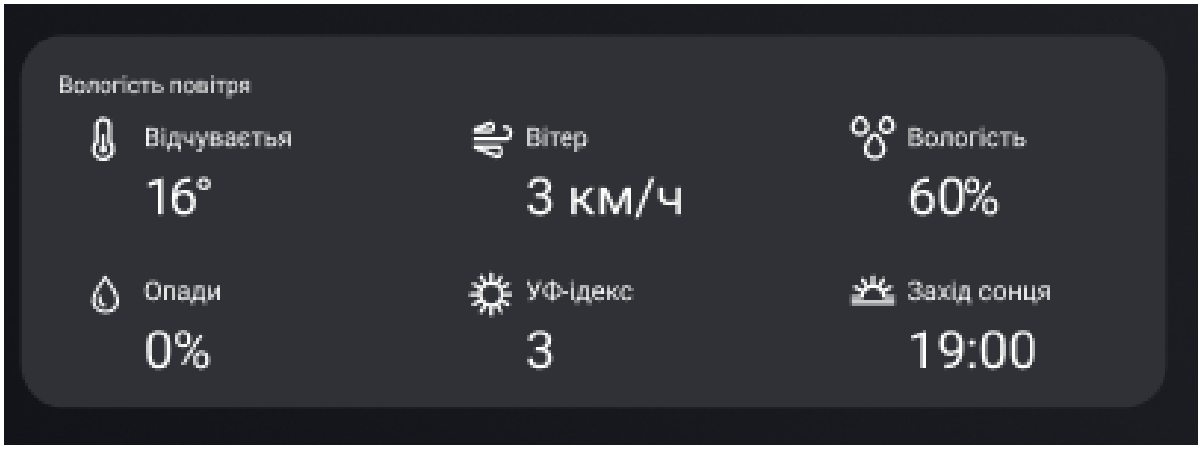

Рисунок 2.5 - Показ додаткових параметрів

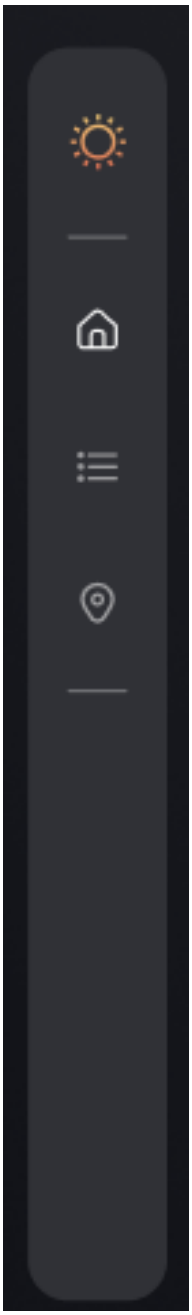

Рисунок 2.6 – Меню навігації

Рисунок 2.7 – Мобільна версія

2.3.2 Принципи UX/UI

Під час розробки інтерфейсу були враховані такі принципи:
- мінімалізм;
- логічна структура;
- висока читабельність;
- швидкий доступ до необхідної інформації;
- адаптивний дизайн.

2.3.2 Адаптивність

Інтерфейс підтримує:
- десктоп;
- планшет;

- мобільні пристрої.

Використовується:

- flexbox;

2.4 Проєктування взаємодії з API

Інтеграція з API є ключовою частиною системи.

2.4.1 Логіка виконання запитів

Рисунок 2.8 — Алгоритм роботи з API

2.4.2 Обробка даних

Дані з API:

- парсяться у JavaScript-об’єкти;
- фільтруються;
- передаються у компоненти.

2.4.3 Обробка помилок

Реалізовано:

- повідомлення про помилки;
- fallback UI;
- повторний запит.

Висновки за розділом

У другому розділі виконано проєктування веб-додатку для візуалізації метеорологічних даних. Було розроблено технічне завдання та визначено функціональні й нефункціональні вимоги до системи.

Створено архітектуру додатку на базі SPA-підходу та компонентної моделі React. Проєкт інтерфейсу користувача враховує принципи UX/UI і адаптивності.

Описано процес інтеграції із зовнішнім API, обробки даних і оптимізації запитів. Також окреслено методи реалізації модуля для візуалізації аналітичних даних.

Отримані результати слугують фундаментом для подальшої розробки програмного продукту.

3. РЕАЛІЗАЦІЯ ВЕБ-ДОДАТКУ ДЛЯ ВІЗУАЛІЗАЦІЇ МЕТЕОРОЛОГІЧНИХ ДАНИХ

3.1 Загальна структура реалізованого веб-додатку

На основі результатів проєктування було реалізовано веб-додаток для отримання та візуалізації метеорологічних даних. Система розроблена з використанням React та побудована за компонентним принципом, що забезпечує гнучкість, масштабованість та зручність підтримки програмного коду.

Основний функціонал додатку включає:

- пошук населених пунктів;
- отримання погодних даних із зовнішнього API;
- відображення поточних погодних умов;
- формування прогнозу погоди;
- візуалізацію даних у графічному вигляді;
- підтримку адаптивного інтерфейсу.

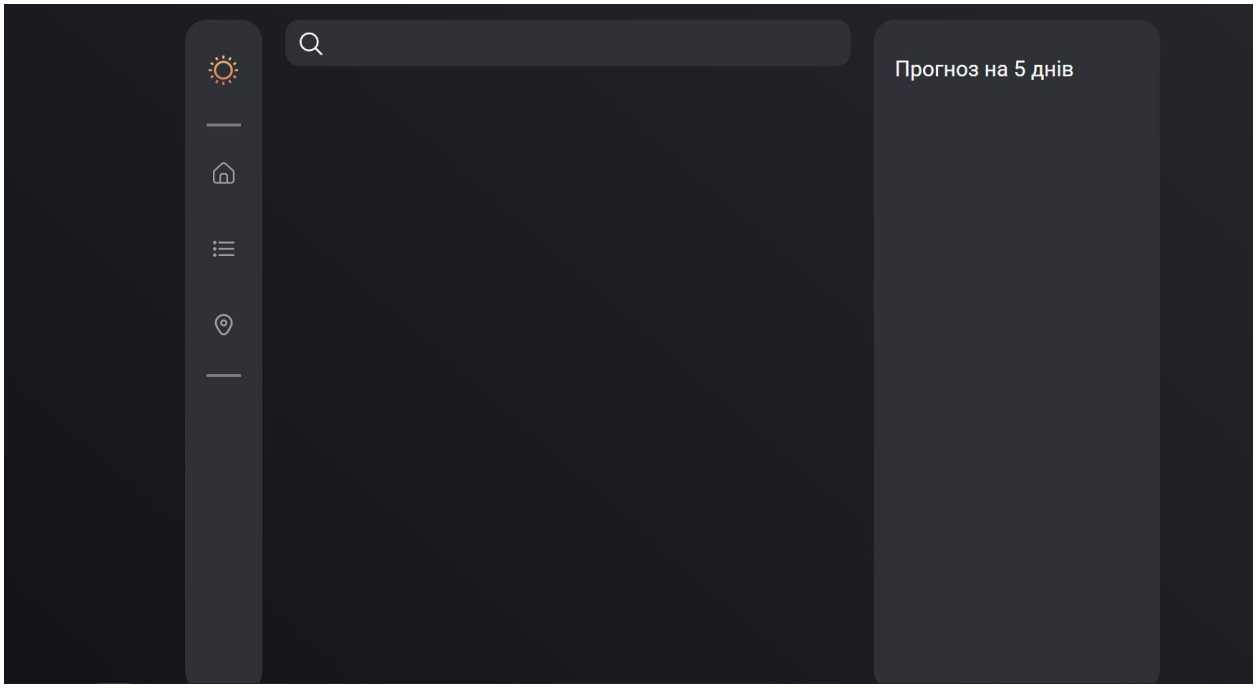

Рисунок 3.1 – Головна сторінка веб-додатку

На рисунку 3.1 представлено головну сторінку системи до завантаження погодних даних.

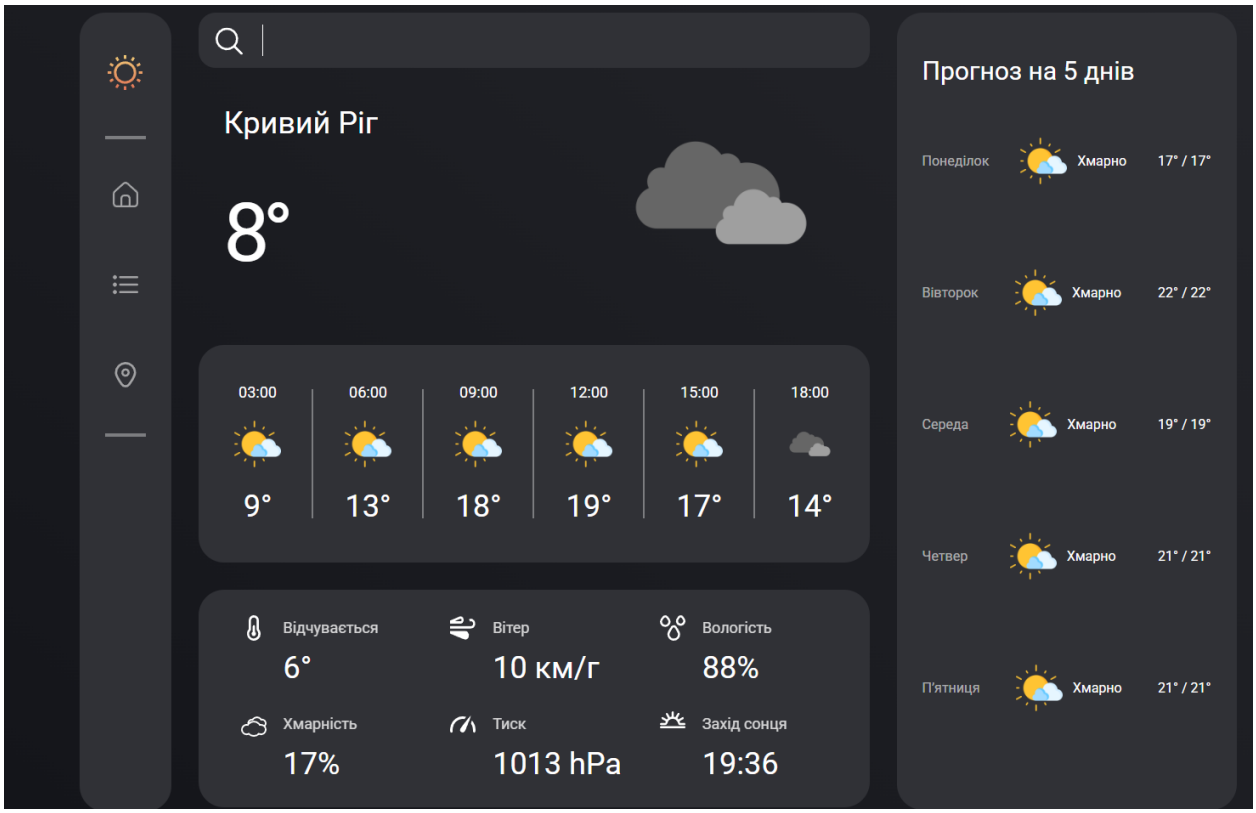

Рисунок 3.2 – Головна сторінка веб-додатку

На рисунку 3.2 представлено головну сторінку системи після завантаження погодних даних. Інтерфейс побудовано за принципами мінімалізму та забезпечує швидкий доступ до основної інформації

3.2 Загальна структура реалізованого веб-додатку

Одним із ключових компонентів системи є модуль пошуку міста. Основним завданням компонента є:

- введення назви населеного пункту;
- перевірка коректності введених даних;
- запуск процесу отримання погодної інформації;
- взаємодія із сервісом геокодування.

Рисунок 3.3 – Компонент пошуку міста

Для реалізації пошуку використовується окремий React-компонент SearchBox, який взаємодіє з модулем отримання даних та передає введене користувачем значення до системи обробки запитів. Код наведений у додатку А.

Після введення назви міста формується запит до API геокодування, результатом якого є отримання координат необхідного населеного пункту.

3.3 Реалізація модуля відображення поточної погоди

Після успішного отримання даних система виконує відображення основних погодних показників.

Рисунок 3.4 – Блок поточної погоди

Відображення поточних погодних умов реалізовано за допомогою компонентів WeatherTodayList та WeatherAdditList. Код компонентів наданий у додатку Б.

До складу компонентів входять такі елементи:

- назва міста;
- поточна температура;
- погодний опис;
- іконка погодних умов;
- швидкість вітру;
- атмосферний тиск;
- рівень вологості.
- як відчувається
- хмарність
- захід сонця

Компонент отримує дані зі сховища стану додатку та автоматично оновлюється після надходження нової інформації від API.

3.4 Реалізація модуля прогнозу погоди

Для надання користувачу розширеної інформації реалізовано модуль прогнозування погодних умов.

Рисунок 3.5 – Блок прогнозу погоди на декілька днів

Для формування прогнозу використовується компонент WeatherDays, який відповідає за відображення масиву прогнозних даних.

Кожен елемент прогнозу містить:

- дату;
- максимальну та мінімальну температуру;
- погодну іконку;
- короткий опис погодних умов.

Використання окремого компонента забезпечує можливість масштабування та модифікації функціоналу без впливу на інші частини системи. Код компонента наданий у додатку В.

3.5 Реалізація системи управління станом

Для централізованого управління даними використовується Redux Toolkit. Код наданий у додатку Г.

Основними елементами системи управління станом є:

- Store;
- Slice;
- AsyncThunk;
- Reducers.

Централізоване зберігання даних дозволяє уникнути дублювання інформації та спрощує взаємодію між компонентами.

3.6 Тестування реалізованого функціоналу

Після завершення розробки було проведено перевірку працездатності основних функціональних модулів системи.

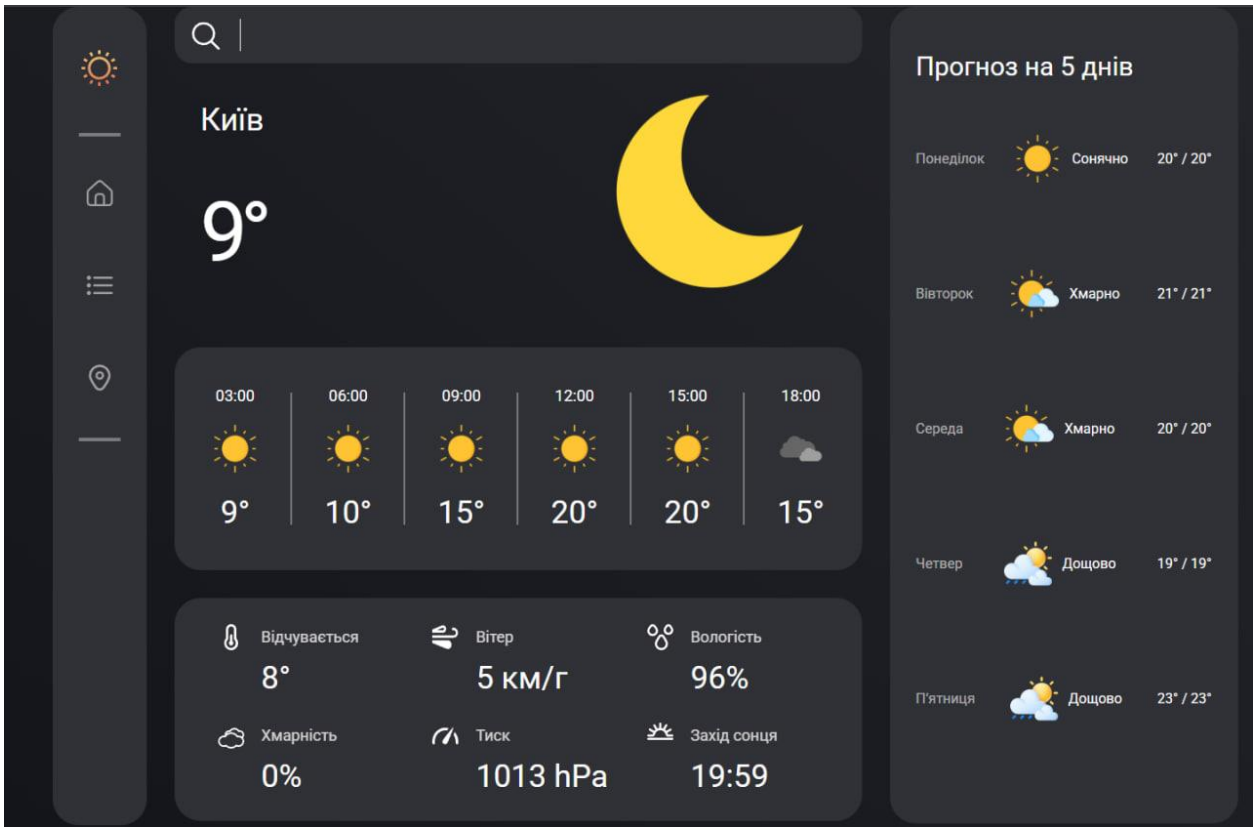

Рисунок 3.6 – Перевірка функціоналу

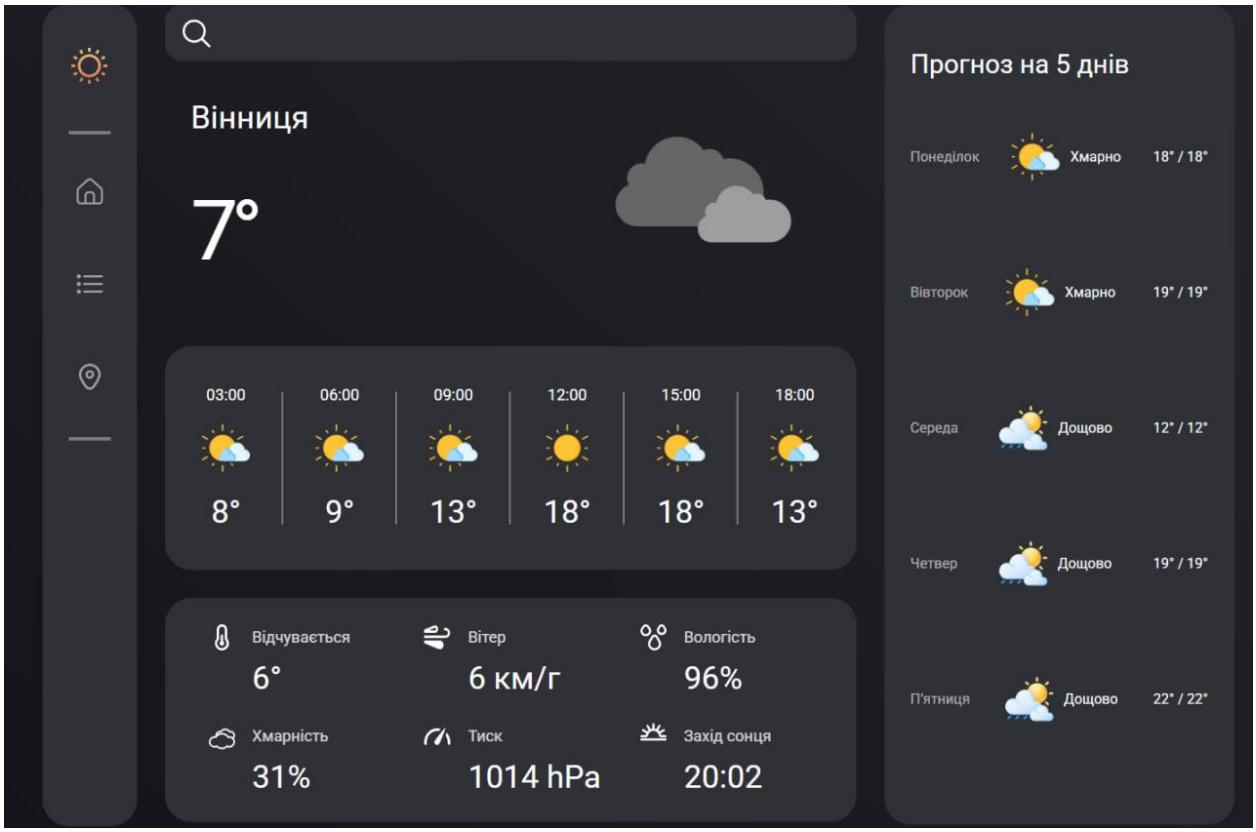

Рисунок 3.7 – Перевірка функціоналу

Рисунок 3.8 – Перевірка функціоналу

Рисунок 3.9 – Перевірка функціоналу

Таблиця 3.1 - Результати тестування

Функціональний модуль	Результат
Пошук міста	Успішно
Отримання погодних даних	Успішно
Відображення поточної погоди	Успішно
Формування прогнозу	Успішно

Результати тестування підтвердили коректне функціонування всіх основних модулів веб-додатку.

Висновок за розділом

У третьому розділі було розглянуто процес реалізації веб-додатку для візуалізації метеорологічних даних. Описано структуру програмного продукту, принципи реалізації основних функціональних модулів та особливості взаємодії компонентів системи.

Було реалізовано модулі пошуку населеного пункту, відображення поточної погоди, прогнозування погодних умов та графічної візуалізації даних. Окрему увагу приділено питанням управління станом додатку, локалізації інтерфейсу та забезпеченню адаптивності системи.

Проведене тестування підтвердило працездатність та відповідність реалізованого програмного продукту поставленим вимогам.

ВИСНОВОК

У результаті виконання дипломної роботи було розроблено веб-додаток для візуалізації аналітичних метеорологічних даних з інтеграцією зовнішніх API.

У ході виконання роботи проведено аналіз сучасних погодних сервісів, досліджено особливості використання зовнішніх API та сучасних frontend-технологій для створення інтерактивних веб-додатків. Виконано порівняння бібліотек візуалізації даних і обґрунтовано вибір технологічного стеку, що включає React, Redux Toolkit, Axios та Recharts.

На етапі проєктування сформовано технічне завдання, визначено функціональні та нефункціональні вимоги до системи, розроблено архітектуру програмного продукту та алгоритми його роботи. Особливу увагу приділено питанням отримання даних із зовнішніх API, їх обробки та подальшого відображення у вигляді графіків і інформаційних блоків.

У процесі реалізації створено модулі пошуку населених пунктів, отримання поточних погодних даних, формування прогнозу погоди, побудови графіків зміни температури та інших метеорологічних показників. Також реалізовано адаптивний інтерфейс користувача та механізми обробки помилок під час взаємодії із зовнішніми сервісами.

Проведене тестування підтвердило працездатність системи та відповідність реалізованого функціоналу поставленим вимогам. Розроблений веб-додаток забезпечує зручне отримання, аналіз та візуалізацію метеорологічної інформації, а також може бути використаний як основа для подальшого розширення функціональних можливостей та інтеграції додаткових джерел аналітичних даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Meteorological Data Systems → Term. *Climate → Sustainability Directory*. URL: http://climate.sustainability-directory.com/term/meteorological-data-systems/#:~:text=Meaning%20→%20Networks%20observing,%20collecting,Earth,%20and%20even%20radar%20installations. (date of access: 16.01.2026).

2. *НУБіП | Національний університет біоресурсів і природокористування України*. URL: https://nubip.edu.ua/sites/default/files/u150/НМК_Метеорологія%20%20кліматологія_Б.pdf (дата звернення: 16.01.2026).

3. OpenWeatherMap API guide. *Current weather and forecast*. URL: https://openweathermap.org/guide (date of access: 16.01.2026).

4. Core Weather quick-start | AccuWeather Developer. *AccuWeather Developer | Personal Weather API | AccuWeather Developer*. URL: https://developer.accuweather.com/documentation/core-weather-quick-start (date of access: 17.01.2026).

5. World Weather API - Online Weather API - WeatherAPI.com. *Free Weather API - WeatherAPI.com*. URL: https://www.weatherapi.com/api.aspx (date of access: 17.01.2026).

6. Single Page Application - что это такое, как работает SPA. *Skillfactory media*. URL: https://blog.skillfactory.ru/glossary/single-page-application/ (дата звернення: 20.01.2026).

7. SPA-програмування OTUS. *OTUS - Онлайн-освіта*. URL: https://otus.ru/journal/spa-programmirovanie/ (дата звернення: 02.02.2026).

8. React. *React*. URL: https://react.dev/ (date of access: 02.02.2026).

9. What is Angular? • Angular. *Home • Angular*. URL: https://angular.dev/overview (date of access: 10.02.2026).

10. Vue.js. *Vue.js - The Progressive JavaScript Framework | Vue.js*. URL: https://vuejs.org/guide/introduction.html (date of access: 12.02.2026).

11. Навіщо потрібен React. Плюси и мінуси. *Хабр*. URL: https://habr.com/ru/sandbox/241172/ (дата звернення: 12.02.2026).

12. Recharts. *Recharts*. URL: https://recharts.github.io/ (date of access: 13.02.2026).

13. Chart.js. *Chart.js | Open source HTML5 Charts for your website*. URL: https://www.chartjs.org/ (date of access: 13.02.2026).

14. D3 by Observable | The JavaScript library for bespoke data visualization. *D3 by Observable | The JavaScript library for bespoke data visualization*. URL: https://d3js.org/ (date of access: 13.02.2026).

15. Powell P., Smalley I. What Is a REST API (RESTful API)? | IBM. *IBM*. URL: https://www.ibm.com/think/topics/rest-apis#:~:text=A%20REST%20API%20is%20an,APIs%20or%20RESTful%20web%20APIs. (date of access: 13.02.2026).

16. Методи HTTP запиту - HTTP | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/ru/docs/Web/HTTP/Reference/Methods (дата звернення: 18.02.2026).

17. What Is a JSON File? How JSON Files Help Your Website (2026) - Shopify. *Shopify*. URL: https://www.shopify.com/blog/what-is-a-json-file?term=&adid=803689462498&campaignid=19724533104&utm_medium=cpc&utm_source=google&matchtype=&network=g&gad_source=1&gad_campaignid=19724533104&gbraid=0AAAAAC3Mv8_5siqYmZUvvR88zQTRnY54v&gclid=Cj0KCQjwof_QBhCgARIsADaMzOfdK8mIBZnLVlBQJgfJ6Zu5aj1HLL1Qn_ruWyccdekDtf3tH6HEqPIaAu40EALw_wcB (date of access: 18.02.2026).

18. Ольга. Що таке JSON. *Хабр*. URL: https://habr.com/ru/articles/554274/ (дата звернення: 22.02.2026).

19. What Is an API Key? How To Use API Keys - Shopify. *Shopify*. URL: https://www.shopify.com/blog/api-key-explained?term=&adid=803689462498&campaignid=19724533104&utm_medium=cpc&utm_source=google&matchtype=&network=g&gad_source=1&gad_campaignid=19724533104&gbraid=0AAAAAC3Mv8_5siqYmZUvvR88zQTRnY54v&gclid=Cj0KCQjwof_QBhCgARIsADaMzOcnqPNx-V--ql9JV2rymdf7B4Q5weCpv4A2ObABnJxYRMxEWHtC8cYaAp0BEALw_wcB (date of access: 22.02.2026).

20. Типи API помилок. *Microsoft Learn: Build with answers in reach*. URL: https://learn.microsoft.com/ru-ru/rest/api/storageservices/common-rest-api-error-codes (дата звернення: 23.02.2026).

ДОДАТКИ

Додаток А

Лістинг А.1 – Код файлу «SearchBox.jsx»

```
import { useState } from "react";
import { useDispatch } from "react-redux";
import { fetchData } from "../../redux/Operations/SearchOps";
import s from "./SearchBox.module.css";
const SearchBox = () => {
  const [city, setCity] = useState("");
  const dispatch = useDispatch();
  const handleSubmit = (e) => {
    e.preventDefault();

    if (!city.trim()) return;
    dispatch(fetchData(city));
    setCity("");
  };
  return (
    <form onSubmit={handleSubmit} className={s.searchBox}>
      <svg className={s.icon}>
        <use href="/src/assets/icons.svg#icon-search"></use>
      </svg>
      <input
        className={s.input}
        type="text"
        value={city}
        onChange={(e) => setCity(e.target.value)}
      />
    </form>
  );
};

export default SearchBox;
```

Лістинг А.2 – Код файлу «SearchBox.module.css»

```
.input {
  box-sizing: border-box;
  width: 790px;
  height: 65px;
  border-radius: 20px;
  background-color: #303136;
```

```
  border: none;
  outline: transparent;
  outline-offset: 0;
  font-size: 30px;
  color: white;
  padding-left: 75px;
}

.icon {
  width: 32px;
  height: 32px;
  fill: white;
  position: absolute;
  transform: translateY(+55%);
  padding-left: 20px;
}

.searchBox {
  position: relative;
  width: 100%;
}
```

Додаток Б

Лістинг Б.1 – Код файлу «WeatherTodayList.jsx»

```
import { useSelector } from "react-redux";
import s from "./WeatherTodayList.module.css";
import { getWeatherImage } from "../../utils/getWeatherImage.js";

const WeatherTodayList = () => {
    const weather = useSelector((state) => state.weather.current);

  if (!weather) return null;

  const condition = weather.weather[0].main;

  const isDay =
    weather.dt >= weather.sys.sunrise && weather.dt < weather.sys.sunset;
  const image = getWeatherImage(condition, isDay);
  return (
    <main className={s.box}>
      <textInfo>
        <h1 className={s.city}>{weather.name}</h1>
                                                <h2 className={s.degree}>{weather.main.temp.toFixed(0)}°</h2>
      </textInfo>
      <imgInfo className={s.imgBox}>
        <img
          className={s.img}
          src={image}
          alt={weather.weather[0].description}
        />
      </imgInfo>
    </main>
  );
};

export default WeatherTodayList;
```

Лістинг Б.2 – Код файлу «ForecastList.jsx»

```
import { useSelector } from "react-redux";
import ForecastCard from "./ForecastCard";
```

```
    import s from "./WeatherTodayList.module.css";

    const ForecastList = () => {
        const forecast = useSelector((state) =>
state.weather.forecast);

      if (!forecast.length) return null;

      return (
        <div className={s.forecastBox}>
          {forecast.slice(0, 6).map((item) => (
            <ForecastCard key={item.dt} item={item} />
          ))}
        </div>
      );
    };

    export default ForecastList;
```

Лістинг Б.3 – Код файлу «ForecastCard.jsx»

```
    import { getWeatherImage } from
"../../utils/getWeatherImage.js";
    import s from "./WeatherTodayList.module.css";

    const ForecastCard = ({ item }) => {
      const condition = item.weather[0].main;

      const isDay = item.sys.pod === "d";

      const image = getWeatherImage(condition, isDay);

      return (
        <div className={s.card}>
           <p  className={s.time}>{item.dt_txt.slice(11,
16)}</p>

                <img  className={s.icon}  src={image}
alt={item.weather[0].description} />

                                          {/*          <p
className={s.weather}>{item.weather[0].main}</p> */}

                                                       <p
className={s.temp}>{item.main.temp.toFixed(0)}°</p>
        </div>
```

```
  );
};

export default ForecastCard;
```

Лістинг Б.4 – Код файлу «WeatherTodayList.module.css»

```
.box {
  box-sizing: border-box;
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding-left: 30px;
  padding-right: 30px;
}

.city {
  font-size: 36px;
  font-weight: 400;
  color: white;
}

.degree {
  margin-top: 50px;
  font-weight: 400;
  font-size: 96px;
  letter-spacing: -0.08em;
  color: white;
}

.imgBox {
  width: 290px;
  height: 230px;
  text-align: center;
}

.img {
  width: 100%;
  height: 100%;
  object-fit: contain;
}

.forecastBox {
  display: flex;
  gap: 60px;
  overflow-x: auto;
```

```
  margin-top: 32px;
  background-color: #303136;
  border-radius: 30px;
  justify-content: center;
  padding-top: 45px;
  padding-bottom: 45px;
}

.card {
  max-width: 70px;
  /* padding: 20px; */

  border-radius: 20px;
  /* background-color: #303136; */

  text-align: center;
  color: white;
  position: relative;
}

.card::after {
  content: "";
  display: block;
  height: 150px;
  border: 1px solid rgba(255, 255, 255, 0.37);
  /* margin-right: 32px; */
  position: absolute;
  right: -30px;
  top: 50%;
  transform: translateY(-50%);
}

.forecastBox .card:last-child::after {
  display: none;
}
.time {
  font-size: 18px;
}

.temp {
  margin-top: 20px;
  font-size: 36px;
}

.weather {
```

```
    margin-top: 10px;
    opacity: 0.7;
  }

  .icon {
    margin-top: 22px;
    width: 70px;
    height: 55px;
    object-fit: contain;
  }
```

Лістинг Б.5 – Код файлу «WeatherAdditList.jsx»

```
  import { useSelector } from "react-redux";
  import Item from "./Item";
  import s from "./WeatherAdditList.module.css";
  const WeatherAdditList = () => {
      const weather = useSelector((state) =>
state.weather.current);

    if (!weather) return null;
    console.log(weather);
    const items = [
      {
        title: "Відчувається",
                                                value:
`${weather.main.feels_like.toFixed(0)}°`,
        icon: "icon-degree",
      },
      {
        title: "Вітер",
                 value:    `${(weather.wind.speed    *
3.6).toFixed(0)} км/г`,
        icon: "icon-wind",
      },

      {
        title: "Вологість",
        value: `${weather.main.humidity}%`,
        icon: "icon-humidity",
      },
      {
        title: "Хмарність",
        value: `${weather.clouds.all}%`,
        icon: "icon-cloudy",
      },
```

```
        {
          title: "Тиск",
          value: `${weather.main.pressure} hPa`,
          icon: "icon-pressure",
        },
        {
          title: "Захід сонця",
               value:  new  Date(weather.sys.sunset  *
1000).toLocaleTimeString([], {
            hour: "2-digit",
            minute: "2-digit",
          }),
          icon: "icon-sunset",
        },
      ];
      return (
        <ul className={s.list}>
          {items.map((item) => (
            <Item key={item.title} item={item} />
          ))}
        </ul>
      );
    };

    export default WeatherAdditList;
```

Лістинг Б.6 – Код файлу «Item.jsx»

```
    import s from "./WeatherAdditList.module.css";
    const Item = ({ item }) => {
      return (
        <li className={s.card}>
          <div className={s.top}>
            <svg className={s.icon}>
                                                  <use
href={`/src/assets/icons.svg#${item.icon}`} />
            </svg>

                                                 <span
className={s.title}>{item.title}</span>
          </div>

          <p className={s.value}>{item.value}</p>
        </li>
      );
    };
```

```
export default Item;
```

Лістинг А.7 – Код файлу «WeatherTodayList.module.css»

```
.list {
  display: flex;
  flex-wrap: wrap;
  gap: 30px;
  padding: 30px;
  background: #303136;
  border-radius: 30px;
  max-width: 790px;
  padding-left: 50px;
  padding-top: 30px;
}

.card {
  width: calc((100% - 60px) / 3);
}

.card {
  color: white;
}

.top {
  display: flex;
  align-items: center;
  gap: 20px;
}

.icon {
  width: 30px;
  height: 30px;
  fill: white;
}

.title {
  opacity: 0.8;
  font-size: 18px;
}

.value {
  margin-left: 50px;
  margin-top: 10px;
```

```
  font-size: 36px;
}
```

Додаток В

Лістинг В.1 – Код файлу «WeatherDays.jsx»

```
import { useSelector } from "react-redux";
import { getWeatherImage } from "../../utils/getWeatherImage.js";
import s from "./WeatherDays.module.css";

import Item from "./Item.jsx";

const WeatherDays = () => {
    const forecast = useSelector((state) => state.weather.forecast);
    // if (!forecast.length) return null;
    const dailyForecast = forecast
                        .filter((item) => item.dt_txt.includes("15:00:00"))
      .map((day) => ({
                                dayName: new Date(day.dt_txt).toLocaleDateString("uk-UA", {
          weekday: "long",
        }),
        date: day.dt_txt.slice(0, 10),
        temp: day.main.temp,
        min: day.main.temp_min,
        max: day.main.temp_max,
        weather: day.weather[0].main,
        isDay: day.sys.pod === "d",
      }));
    return (
      <div className={s.forecastBox}>
        <p className={s.title}>Прогноз на 5 днів</p>
        {dailyForecast.slice(0, 5).map((day) => (
          <Item key={day.date} day={day} />
        ))}
      </div>
    );
};

export default WeatherDays;
```

Лістинг В.2 – Код файлу «Item.jsx»

```
import { useSelector } from "react-redux";
import {
  getWeatherImage,
```

```
    getWeatherText,
  } from "../../utils/getWeatherImage.js";
  import s from "./WeatherDays.module.css";
  const Item = ({ day }) => {
    const condition = day.weather;

    const formattedDayName =
            day.dayName.charAt(0).toUpperCase()    +
day.dayName.slice(1);
       const   image   =   getWeatherImage(condition,
day.isDay);
    const text = getWeatherText(condition);
    return (
      <div className={s.card}>
                                                    <p
className={s.dayName}>{formattedDayName}</p>
        <div className={s.weather}>
          {" "}
         <img className={s.img} src={image} alt={text}
/>
          <p className={s.discription}>{text}</p>
        </div>

        <p className={s.maxMin}>
                          {day.min.toFixed(0)}°     /
{day.max.toFixed(0)}°
        </p>
      </div>
    );
  };

  export default Item;
```

Лістинг В.3 – Код файлу «Item.jsx»

```
  .forecastBox {
    font-size: 16px;
    box-sizing: border-box;
    border-radius: 30px;
    padding: 50px 30px;
    width: 400px;
    max-height: 938px;
    background-color: #303136;
  }

  .title {
```

```
  font-size: 30px;
  color: white;
  margin-bottom: 60px;
}

.card {
  display: flex;
  gap: 22px;
  justify-content: space-between;
  align-items: center;
}

.card:not(:last-child) {
  margin-bottom: 100px;
}

/* .card:first-child {
  margin-top: 40px;
} */

.dayName {
  color: rgba(255, 255, 255, 0.6);
}

.weather {
  display: flex;
  gap: 12px;
  justify-content: center;
  align-items: center;
}

.img {
  width: 55px;
  height: 55px;
}

.discription {
  color: white;
}

.maxMin {
  color: white;
}
```

Додаток Г

Лістинг Г.1 – Код файлу «Store.js»

```
import { configureStore } from "@reduxjs/toolkit";
import { WeatherDataReducer } from "./WeatherData/WeatherDataSlice";

export const store = configureStore({
  reducer: {
    weather: WeatherDataReducer,
  },
});
```

Лістинг Г.2 – Код файлу «WeatherDataSlice.js»

```
import { createSlice } from "@reduxjs/toolkit";
import { fetchData } from "../Operations/SearchOps";

const initialState = {
  current: null,
  forecast: [],
  loading: false,
  error: null,
};

const slice = createSlice({
  name: `weather`,
  initialState,
  reducers: {},
  extraReducers: (builder) => {
    builder
      .addCase(fetchData.pending, (state) => {
        state.loading = true;
      })

      .addCase(fetchData.fulfilled, (state, action) => {
        state.loading = false;
        state.current = action.payload.weather;
                                    state.forecast = action.payload.forecast.list;
                                  state.current.name = action.payload.cityName;
      })
```

```
            .addCase(fetchData.rejected, (state, action)
=> {
              state.loading = false;
              state.error = action.error.message;
            });
      },
    });

    export const WeatherDataReducer = slice.reducer;
```

Лістинг Г.3 – Код файлу «SearchOps.js»

```
    import { createAsyncThunk } from "@reduxjs/toolkit";
    import axios from "axios";

    const API_KEY = "53730d6999267440358ba56988ad3717";

    export const fetchData = createAsyncThunk(
      "weather/fetchData",

      async (city) => {
        try {
          const geoResponse = await axios.get(
            `http://api.openweathermap.org/geo/1.0/dire
ct?q=${city}&limit=1&appid=${API_KEY}`
          );

          const location = geoResponse.data[0];

          const lat = location.lat;
          const lon = location.lon;

          const weatherResponse = await axios.get(
            `https://api.openweathermap.org/data/2.5/we
ather?lat=${lat}&lon=${lon}&units=metric&appid=${API_KE
Y}`
          );
          const weatherForecast = await axios.get(
            `https://api.openweathermap.org/data/2.5/fo
recast?lat=${lat}&lon=${lon}&units=metric&appid=${API_K
EY}`
          );
          // console.log(weatherForecast.data);
          // console.log(weatherResponse.data);
          return {
            weather: weatherResponse.data,
```

```
                cityName:  location.local_names?.uk  ||
location.name,
              forecast: weatherForecast.data,
            };
          } catch (error) {
            console.log(error);
          }
        }
      );
```