МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ

ННІ/факультет Навчально-науковий інститут економіки та бізнес-освіти
Кафедра Кафедра економіки та цифрового бізнесу
Спеціальність Комп'ютерні науки
Форма навчання Денна форма

КВАЛІФІКАЦІЙНА РОБОТА

Сенько Владислава Михайловича
(прізвище, ім'я, по батькові здобувача)

на тему Розробка web-застосунку для агрегації і аналізу метеорологічних даних
(повна назва теми)

за матеріалами _____

(повна назва бази дослідження)

науковий керівник К. е. н., доцент _____ Соловйова В.В.
(наук. ступінь, вчене звання) *(підпис)* *(прізвище, ініціали)*

Робота допущена до захисту в ЕК
Протокол засідання кафедри
від 09 червня 2025 р. № 12

Завідувач кафедри

_____
(підпис)
К.е.н., доцент В.М. Радько
Наук. ступінь, вчене звання *Ініціали, прізвище*

Кривий Ріг – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЕКОНОМІКИ ТА БІЗНЕС-ОСВІТИ

Кафедра економіки та цифрового бізнесу
Освітній ступінь бакалавр
Спеціальність Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри _____В.М. Радько

"07" квітня 2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА ЗДОБУВАЧУ

_____Сенько Владислава Михайловича_____

1. Тема роботи Розробка web-застосунку для агрегації і аналізу метеорологічних даних____,
науковий керівник роботи Соловйова Вікторія Володимирівна _____,
затверджені наказом вищого навчального закладу від «04» квітня 2025 р. № 224-ст (д/ф)
№ 151-ст (з/ф)
2. Строк подання здобувачем роботи 31.05.2025р.

3. Зміст кваліфікаційної роботи бакалавра, об'єкт, предмет та мета дослідження:

Розділ 1 Теоретичні аспекти і сучасні технології проектування web-застосунку для агрегації і аналізу метеорологічних прогнозів _____
Розділ 2 Проєктування і розробка структури web-застосунку для агрегації і аналізу метеорологічних прогнозів _____
Розділ 3 Розробка і тестування web-застосунку для отримання метеорологічних прогнозів ____

Об'єкт дослідження – процеси розробки web-застосунку для агрегації і аналізу метеорологічних даних
Предмет дослідження – розробка web-застосунку для агрегації і аналізу метеорологічних даних

Мета кваліфікаційної роботи бакалавра – аналіз технологій проектування і розробка структури й функціоналу застосунку для метеорологічних прогнозів

4. Дата видачі завдання 04.04.2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Відмітка керівника про виконання етапів (дата, підпис)
1	Підготовка розділу 1	до 28.04.2025р.	25.04.2025
2	Підготовка розділу 2	до 16.05.2025р.	15.05.2025
3	Підготовка розділу 3	до 30.05.2025р.	29.05.2025
4	Реєстрація завершеної дипломної роботи	до 31.05.2025р.	30.05.2025
5	Отримання відгуку від наукового керівника	03-04.06.2025р.	04.06.2025
6	Отримання зовнішньої рецензії	05-06.06.2025р.	06.06.2025
7	Перевірка кваліфікаційної роботи на плагіат	02-09.06.2025р.	04.06.2025
8	Попередній захист кваліфікаційної роботи на кафедрі	03.06.2025р.	03.06.2025
9	Допуск кафедрою кваліфікаційної роботи до захисту	09.06.2025р.	09.06.2025
10	Підготовка студента до захисту в ЕК	до 17.06.2025р.	17.06.2025

Завдання підготував науковий керівник _____ Соловйова В.В._____
(підпис) (прізвище та ініціали)

Завдання одержав здобувач _____ _____Сенько В.М._____
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Обсяг роботи: 68 сторінок, 18 рисунків, 35 джерел, 1 додаток.

Об’єкт дослідження: процеси розробки web-застосунку для агрегації і аналізу метеорологічних даних.

Предмет дослідження – розробка web-застосунку для агрегації і аналізу метеорологічних даних.

Ціль роботи: створення функціонального web-застосунку для агрегації погодних даних, обробки даних і можливості збереженні історії запитів до окремих джерел.

Метод дослідження: аналіз доступних інструментів та технологій, моделювання системи застосунку, реалізація клієнт-серверної архітектури та інтерфейсу за допомогою обраних методів та інструментів розробки.

Результат дослідження: у результаті виконання кваліфікаційної роботи було розроблено повнофункціональний web-застосунок для агрегації погодних даних та отримання прогнозів, який відповідає визначеним вимогам та має необхідний функціонал і систему акаунтів для користувачів і перегляду історії запитів, а також можливість перегляду графіки показників та адаптацію роботи застосунку під мобільні пристрої.

Розроблений web-застосунок має налаштовану клієнт-серверну архітектуру на основні компонентного підходу, реалізовану за допомогою React й Node.js з підтримкою обробки даних і запитів до API, формування прогнозів і систему власних кабінетів для користувачів застосунку.

Результати роботи у вигляді дослідження і розробленого застосунку мають значення для аналізу сфери метеорологічних прогнозів, аспектів використання погодних даних і реалізації систем для надання оброблених даних у вигляді прогнозів або використання функціоналу застосунку у загальнодоступних системах для більшості користувачів.

Ключові слова: java script, прогноз погоди, метеорологічні дані, агрегація і аналіз, методи проектування веб-застосунку, робота з API, реляційна база даних.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API – Application Programming Interface

HTTP – HyperTextTransferProtocol

FTP – File Transfer Protocol

SQL – Structured Query Language

NoSQL – Not Only SQL

UI – UserInterface

UX – UserExperience

DOM – Document Object Model

CSS – Cascading Style Sheets

HTML – HyperText Markup Language

JS – JavaScript

JSON – JavaScript Object Notation

Npm – Node Package Manager

JWT – JSON Web Token

GFS – Global Forecast System

REST API – Representational State Transfer API

SOAP API – Simple Object Access Protocol API

ECMWF – European Centre for Medium-Range Weather Forecasts

NWP – Numerical Weather Prediction

IoT – Internet of Things

ВСТУП

В сучасному світі прогнозування погоди та побудова якісних та точних прогнозів є основною метою компаній та платформ, які надають метеорологічні дані у різних форматах, орієнтуючись на різних користувачів, таких як звичайних побутових споживачів, так і великих бізнес-структур, транспортних компаній або державних підприємств. Метеорологічні сервіси, які засновані на використанні цифрових технологій, обробці великих даних та формуванню точних та ясних прогнозів для конкретного користувача сервісу, застосовують продвинуті алгоритми, машинне навчання, штучний інтелект для обробки великої кількості даних та пошуку певних тенденцій, а також досліджень потреб користувачів сервісів прогнозу та їх інтересів для більш кращого представлення погодних даних та їх формування у сприятливому для людини вигляді.

Актуальність теми дослідження обумовлена важливістю відстеження змін атмосфери та погоди у різних місцевостях для користувачів усіх типів та різного призначення метеорологічних даних, від створення прогнозів збору врожаю для сільських господарств до попереджень вкрай небезпечних погодних явищ, що можуть завдати значної шкоди населенню та інфраструктурі. Метеорологічні прогнози застосовуються у сфері будівництва, промисловості, усіх видах транспортної галузі, проведення культурних та громадських заходів. Тому для швидкого надання потрібних даних та формування прогнозу використовують різноманітні методи – від моніторингу атмосфери і використання супутників, до застосування цілих мереж датчиків IoT і використання штучного інтелекту для прогнозування змін стану погоди. Це дозволяє у режиму реального часу оновлювати метеорологічні дані й формувати прогноз кожну хвилину, дозволяючи користувачам отримувати найактуальнішу інформацію про погоду, температуру та можливі несприятливі умови. Для більш кращого сприйняття використовують візуальне оформлення та графіки погодних показників, інтерактивні метеорологічні карти та можливий текстовий опис для повної зрозумілості наданих параметрів.

Основна мета роботи полягає в розгляді аспектів розробки і аналізу сучасних технологій проектування web-застосунку, проектуванні його структури, а також повноцінної розробки і тестування застосунку для агрегації і аналізу метеорологічних прогнозів.

Також в роботі були визначені наступні завдання: аналіз сучасних аспектів розробки застосунків та існуючих систем проектування та технологій розробки, створення архітектури web-застосунку і його структури, розробка визначеного вимогами функціоналу й тестування web-застосунку для агрегації і аналізу на предмет можливих проблем і помилок.

Об’єкт дослідження: процеси розробки web-застосунку для агрегації і аналізу метеорологічних даних.

Предмет дослідження – розробка web-застосунку для агрегації і аналізу метеорологічних даних.

У методах дослідження були використані функціональний аналіз, огляд і порівняння доступних інструментів, архітектури проектування системи на компонентній основі, використання API на основі HTTP-запитів й реалізацію клієнт-серверної моделі застосунку з інтеграцією реляційної бази даних. Було зроблено аналіз вимог та потреби користувачів, на основі яких визначені інструменти реалізації системи. У процесі розробки структури й архітектури застосунку за допомогою вибраних інструментів було реалізовано користувацьку та серверну частину web-застосунку, розроблено інтеграції з API й базою даних, а також формування прогнозів та візуалізацію у вигляді графіків показників.

Використання інформаційної бази дослідження у вигляді технічної документації, даних про інструменти та методи розробки, наукових і дослідницьких джерел дозволило реалізувати необхідний функціонал і виконати визначені вимоги при розробці.

РОЗДІЛ 1
ТЕОРЕТИЧНІ АСПЕКТИ І СУЧАСНІ ТЕХНОЛОГІЇ ПРОЕКТУВАННЯ WEB-ЗАСТОСУНКУ ДЛЯ АГРЕГАЦІЇ І АНАЛІЗУ МЕТЕОРОЛОГІЧНИХ ПРОГНОЗІВ

1.1 Поняття і принципи розробки веб-застосунків

Web-застосунок (Web-application) являє собою створене програмне забезпечення, яке складається з декількох частин – клієнта і сервера, що доступно через глобальну мережу і має розроблений інтерфейс для взаємодії з користувачем, що виключає необхідність встановлення програми на пристрій користувача. Це робить створені веб-програми більш комфортними для застосування, так як не потребують довгого завантаження, порівняння необхідних вимог використання й додаткових налаштувань, а лише доступу до інтернету та наявності належного веб-браузера [1].

Web-застосунок і його розробка базується на принципі розподілу відповідальності у вигляді клієнт-серверної архітектури, яка має доволі зрозумілу модель взаємодії у вигляді клієнта і сервера, що пов’язані між собою і створюють працездатну структуру застосунку. Клієнтом є програма або пристрій, яке створює і відправляє запити до сервера, а сервер працює у вигляді центрального компонента, який оброблює запити, виконує операції і обробку інформації, надаючи потрібні дані або послуги. Клієнт і сервер в цілому пов’язані через мережу за допомогою налаштованого протоколу, такого як HTTP для передачі запитів і веб-сторінок або FTP для надсилання файлів. Також в конфігураціях web-застосунках часто фігурують бази даних, які надають системі і застосунку можливість зберігати користувацькі дані, отриману інформацію з інших джерел, повідомлення тощо у вигляді SQL або NoSQL-баз.

При детальному розгляді побудови web-застосунку можна розділити типи архітектур на кілька варіантів, такі як дворівневу у вигляді клієнта і сервера[2], трирівневу з двома загальними елементами та розширеної логіки на сервері, і

багаторівневу, яка має ускладнену архітектуру і різні додаткові елементи, такі як підключені мікросервісі, шлюзи і застосунки. Це також враховує принцип масштабованості, який потребує вибору необхідної архітектури для подальшої розробки. Різні типи архітектур застосовуються для різних типів застосунків при їх розробці, наприклад при створенні невеликого додатка використовується дворівнева архітектура, для створення вже більш складного проекта, наприклад показу погоди або аналізу персональних витрат необхідно слідкувати за основними принципами і правилами трирівневої архітектури[3]. У створенні же великих застосунків, на кшталт онлайн-магазину або сервісу послуг, потрібно вже орієнтуватися на багаторівневу архітектуру, яка має багато розгалужень і можливостей для створення великого і складного web-застосунку.

При початку розробки більшість спеціалістів звертають увагу на кілька підходів розробки web-застосунків, щоб визначитися з цілями і пріоритетом, а також функціоналом застосунку і його можливостями. Це залежить від типу застосунку, його призначення, подальших цілей розробки та підтримки в майбутньому, так як обраний фундамент всього процесу створення застосунку у вигляді вибраного підходу розробки стане основою для створення всього продукту, і вплив вибору буде досить великим, що впливатиме на подальшу розробку і вибір інструментів, які знадобляться для завершення розробки і можливого вдосконалення додатка. Основний класичний підхід полягає в розподілі логіки між сервером та клієнтом, де більшість операцій обробки даних відбувається на сервері, що дає великі можливості налаштування інтерфейсу і його використання, при цьому централізована логіка спрощує контроль всього застосунку і безпеку. Другий же підхід – Single Page Application полягає в розташуванні функціоналу на стороні клієнта, де розміщується інтерфейс і більшість алгоритмів, а взаємодія з сервером і сторонніми сервісами відбувається через запити різних форматів і встановлених зв’язків. Це має певні переваги, таке як виконується динамічне оновлення без необхідності перезавантаження сторінок, але швидкість операцій обробки при такому підході знижується. Ще існує такі підходи, як Server Side Rendering, який полягає в

рендерингу сторінок і їх коду на стороні сервера, що знижує навантаження на клієнт, а Static Site Generation заснований на генерації статичних файлів для інтерфейсу заздалегідь, що робить високу швидкість завантаження сторінок і досить високий рівень безпеки.

Для взаємодії між клієнтом і сервером використовуються такі технології, як RESTful API та GraphQL, які базуються на використанні певних протоколів і дозволяють налаштовувати відповідні запити і обмін інформацією. Наприклад RESTful API на основні основного протоколу комунікації HTTP за допомогою принципів REST, що містять в собі встановлену архітектуру, необхідну взаємодію, єдиний інтерфейс і кешування даних дозволяє інтегрувати цю технологію в розроблюваний застосунок, відповідаючи усім стандартам популярного протоколу HTTP і маючи доволі просту реалізацію. Вибір RESTful API надає простоту при і відповідність, але в цій технології присутні проблеми з обмеженнями і надлишками даних, що не підходить для деяких видів застосунків і запитів різних типів. Альтернативою у вигляді схожого варіанта є GraphQL, який має гнучкі налаштування, можливість кастомізації запитів і створення складних запитів, але є недоліки у вигляді високого навантаження та складності інтеграції у додаток.

Ще одним важливим компонентом у архітектурі web-застосунку, є бази даних, які відповідають за збереження інформацію у структурованій формі. Вони поділяються на два типі – реляційні бази SQL, які зберігають дані у вигляді організації таблиць – MySQL, PostgreSQL та SQLite, а бази по типу NoSQL на прикладі MongoDB чи Firebase використовують інші формати збереження даних і підходять для більш гнучких розширених проектів з різними форматами і залежностями, дозволяючи вибрати необхідній інструмент під свій тип розробки та застосунку[4]. Самі ж бази даних позволяють зберігати велику кількість інформації, маючи досить великі можливості для інтеграції різних налаштувать і додаткових алгоритмів, наприклад автоматизованого підбору чи видалення зайвих даних, що робить їх одним з основних важелів при створенні продвинутих web-застосунків, які мають обробляти та зберігати велику кількість даних від

користувачів.

Велику роль у розробки застосунку і представлення його для користувацького використання є розробка дизайну і інтерфейсу, які називаються інтерфейсом користувача UI і користувацьким досвідом UX, які є доволі важливими компонентами для ефективної розробки web-застосунку і його впровадження для широкого користування. Основні принципи UI полягають у простоті, адаптивності та доступності, а також фактору передбаченості, що має орієнтувати інтерфейс додатка на більш стандартну і популярну структуру сторінок й сайтів у мережі, даючи користувачу швидкий старт і початок роботи без довгого періоду звикання з новою і незвичною структурою. Користувацький досвід UX же повинен мати доступну навігацію для швидкого користування, достатню інтерактивність при взаємодії користувача з інтерфейсом й швидке завантаження вмісту й роботу функцій без довгого очікування або завантаження, щоб забезпечити безперебійність користування web-застосунку й достатньо зручне використання без очікування або потенційних помилок . Для коректності і правильної роботи застосунку потрібно використовувати поширені CSS-технології, які допоможуть вдосконалити відображення інтерфейсу і покращити взаємодію з користувачем [5]. Відповідання основним характеристикам принципу адаптивності покращує використання застосунку на різних платформах й пристроях. Наприклад використання технології CSS Flexbox надає можливість адаптувати web-застосунок під різні пристрої з різним розширенням екрана. Також використання медіа-запитів і технології CSS-grid розширяє адаптивність і створює більшу кросплатформеність застосунку, для його використання від персональних комп’ютерів і ноутбуків до смартфонів та планшетів. Врахування цих факторів і технологій користувацького досвіду й створеного інтерфейсу допоможуть забезпечити більшу зручність та достатню практичність використання, а популярні CSS-технології забезпечать необхідну адаптивність і кросбраузерність веб - застосунку, що допоможе збільшите привабливість й доступність для багатьох категорій користувачів створеного web-застосунку.

1.2 Технології і засоби розробки веб-застосунків

Для розробки web-застосунків, які повинні відповідати різним вимогам й мати можливість оновлюватися для покращення програми, більшість використовує JavaScript – високорівневу інтерпретовану мову програмування, яка використовується для розробки web-застосунків і серверних рішень, мобільних і комп'ютерних програм[6]. Java Script має динамічну типізацію, виконання коду без попередньої компіляції та виконання асинхронних операцій без всяких доповнень і проблем. Також JS є кросплатформеною мовою і підтримується у багатьох браузерах, серверах та інших середовищах. Java Script як поширена мова програмування підтримує декілька підходів до програмування, що робить його доволі універсальним інструментом: процедурне програмування, об'єктно-орієнтоване програмування й функціональне програмування. В цілому JS використовується для створення інтерактивних web-застосунків, разом з розміткою HTML й стилями CSS, що допомагає створювати функціональний і гарний інтерфейс. Також JS може використовуватися на стороні сервера завдяки Node.js , для створення і підтримки серверної частини програмного продукту. Ще важлива можливість даної мови полягає в взаємодії з зовнішніми API, що дозволяє інтегрувати сторонні сервіси й широкі можливості без складних проблем.

Однією з самих поширених бібліотек в JavaScript є React, який призначений для створення інтерфейсів і розробки динамічних web-застосунків, які розробляються на основі побудови типу SPA, тобто оновлення даних без необхідності перезавантаження сторінки[7]. Основна концепція React – це компонентний підхід**.** Усі елементи UI представлені у вигляді незалежних компонентів, які можна перевикористовувати та вкладати один в один. Компонентний підхід є доволі великою перевагою у розробці, який дозволяє розробникам створювати елементи і забезпечувати високу продуктивність веб-додатків. Використання бібліотеки підвищує продуктивність й має можливість інтеграції с іншими популярними доповненнями, тому вважається одною з

популярних бібліотек JS. Також бібліотека використовує технологію DOM, що спрощує оновлення компонентів і підвищує продуктивність. Завдяки можливості використання хуків можна створювати запиті до API, налаштовуючи більшість компонентів для отримання даних про погоду. Через це використовується в більшості процесів розробки веб-додатків, комерційних платформ, адміністративних панелей, сайтів новин та мобільних застосунків. Структурування компонентів дозволяє розділити додаток на різні логічні компоненти,а адаптивність і динамічне оновлення даних дозволяє налаштовувати компоненти дуже швидко, з оновленням інтерфейсу і структури у реальному часі.

Бібліотека React Router DOM дозволяє керувати маршрутизацією у React-застосунках, дозволяючи створювати застосунки з динамічними змінами та оновленням без перезавантаження сторінки. Має більшість особливостей, такі як декларативний підхід, створення динамічних маршрутів, реалізацію програмної навігації та роботу з URL-запитами, а також обробку помилок та підтримку захищених маршрутів. Використовується у сфері розробці динамічних web-застосунків, структуризації веб-сайтів, обробки динамічних маршрутів й реалізацію вкладених сторінок при можливості оптимізації, гнучкого налаштування навігації та кросбраузерної підтримки та адаптації під різни типи пристроїв. Завдяки широким можливостям навігації та динамічному налаштуванню використовується для створення й підтримки сайтів з великою кількістю сторінок, такі як інтернет-магазини, сайти новин, блоги або різні довідники, а також панелі керування сервісами й платформи налаштування.

Також популярним є Node.js, який є асинхронним середовищем виконання JavaScript на сервері, що дозволяє виконувати створений код без браузера та легко створювати серверні застосунки за допомогою наявних доступних можливостей[8]. Він має безліч можливостей, таких як обробку запитів та відповідей, роботу з файловою системою, асинхронним програмуванням та обробкою потоків, а також використовується для роботи і взаємодії з базами даних. Маючи оптимальну модель виконання, містить однопоточну архітектуру

для зменшення навантаження на систему й легку масштабованість, що підходить для більшості програм, застосунків та мікросервісів для оптимізації і покращення продуктивності роботи. Використовується доволі широко у створенні серверних застосунків та веб-додатків, обробці API та даних у реальному часі, потокової передачі даних різних форматів та мікросервісної архітектури. Через велику систему доступних модулів для інтеграції і взаємодії з базами даних застосовується у більшості серверних додатків, месенджерів та чатів, а також у сфері торгівлі у системах електронної комерції та замовлень, має можливості застосування у системах типу розумних будинків або трекерів відстежування змін стану погоди та середовища завдяки можливості обробки великих масивів інформації та стабільної роботи.

Ще одним популярним напрямом у розробці на JavaScript є підтримка і інтеграція API - інтерфейсу взаємодії між застосунками, який дозволяє отримувати дані від серверів, інтегрувати сторонні сервіси та автоматизувати запити. Це дозволяє оптимізувати процеси, взаємодіяти та отримувати дані з сторонніх сервісів без створення власних методів обміну даними, оскільки популярні формати даних для отримання на сервер підтримують більшість API, а багато платформ й сайтів надають можливості отримання чи придбання ключа API для легкої інтеграції та взаємодії, що виводить створення різних web-застосунків на новий рівень.

Доступна бібліотека Axios створена для виконання HTTP-запитів у веб-браузері та Node.js, що надає гарні можливості та розширений функціонал[9]. Крім підтримки всіх прийнятих HTTP-методів, бібліотека має можливість інтеграції з серверними платформами, встановлення визначених глобальних параметрів для запитів, а також розширену обробку помилок та оптимізацію продуктивності. Зазначена бібліотека Axios дозволяє взаємодіяти з REST API для отримання та відправки даних, реалізацію та обробку даних при процесі авторизації, оптимізацію запитів та підтримку проксі-серверів для більш швидкої й стабільної роботи. Axios використовується у розробці динамічних веб-додатків, вдосконалених систем авторизації та аутентифікації, погодних

застосунках та різних видах чат-ботів й месенджерів, що потребують великої кількості запитів та отримання інформації з різних джерел або серверів.

Для створення змісту сторінок й структури у web-застосунку використовується HTML – мова розмітки[10], яка за створення логічної структури та елементи сторінок, щоб користувач міг розібратися в наповненні окремих частин або всього, для розміщення мета-даних й основного контенту сторінки. Основні характеристики HTML містять в собі простий синтаксис та гіперпосилання, а також можливість використання семантичних тегів й інтеграції адаптивності по типу вбудованих скриптів з іншими засобами розробки. Має підтримку кросбраузерності, підтримку мультимедіа та веб-компонентів. Завдяки простоті та основним принципам розробки HTML застосовується у створенні більшості сайтів та різних платформ, динамічних веб-додатків, блогів чи навіть власних сторінок, які містять загальну структуру та елементи HTML. Найчастіше саме HTML використовується для створення веб-сторінок, а саме формування структури контенту, роботи з формами, адаптивної верстки, розмітки статичних та динамічних сторінок, але також може бути корисним для взаємодії з іншими інструментами або для SEO-оптимізації.

CSS є однією з важливих технологій для розробки веб-додатків, що забезпечує створення візуальних елементів і інтерфейсів [11]. Каскадні таблиці стилів визначають, як елементи сторінки повинні виглядати на екрані, відображатися з встановленими параметрами і налаштуваннями. Це є одним із основних інструментів створення гарних і привабливих інтерфейсів і елементів, завдяки різноманіттю функцій і можливостей CSS, які дозволяють доволі просто редагувати, застосовувати необхідні правила і налаштовувати різні аспекти стилю відображення: від кольорів і шрифтів до розмірів і розташувань. Завдяки постійним оновленням і підтримці сучасних технологій CSS мають інструменти для анімацій, перевикористання стилів і багато іншого, що допомагає створити доволі функціональний дизайн для веб-додатка, даючи користувачам можливість перебувати у доволі налаштованому середовищі. Завдяки популярності й широким можливостям CSS застосовується при розробці

більшості веб-сайтів, мобільних сайтах та різних онлайн платформах, а також в окремих застосунках на кшталт графічних веб-редакторів чи анімованих програмах, забезпечуючи великі можливості оформлення та створення приємного інтерфейсу для користувачів.

Для роботи з базами даних й створення запитів використовується SQL, яка призначена для роботи з реляційними базами даних, такими як MySQL, SQLite, PostgreSQL[12]. Сама ж мова запитів дозволяє зберігати дані в таблицях, виконувати операції по створенню, оновленню та видаленню, а також фільтрувати та агрегувати дані. Крім основних операцій, мова запитів дозволяє виконувати складні операції обробки даних, вкладених запитів, а також проводити доволі безпечні й надійні транзакції, які можуть бути налаштовані та оптимізовані за бажанням. SQL використовується в бекенді для зберігання та обробки даних у web-застосунках, управління базами даних й взаємодії з сервером. Через свої можливості та функціонал використовується у безлічі фінансових сервісів для обробки великої кількості інформації, соціальних мережах, логістичних системах, аналітиці та систем обліку клієнтів чи товарів, які потребують обробки даних та надійності.

Однією з популярних систем управління базами даних є MySQL, що використовується для збереження, обробки та управління даними[13]. Крім підтримки усіх функцій SQL, вона має підтримку масштабованості, забезпечення безпеки даних через надійне шифрування й аутентифікацію, містить властивості типу ACID для коректної роботи, а також підтримує реплікацію і інтеграцію з популярними веб-технологіями. Завдяки гнучкому масштабуванню й сумісністю з поширеними мовами програмування типу JavaScript, Python та PHP користується великою популярністю при розробці web-застосунків та великих систем, роботі з додатками для збереження необхідної інформації й обробки даних, а також реплікації та масштабування середніх та великих проектів, які потребують автоматизації операцій обробки даних й збереження потрібної інформації при забезпеченні достатньої безпеки та повної надійності при налаштованому використанні.

1.3 Аналіз сучасних фреймворків і бібліотек для створення застосунку

React - це бібліотека JavaScript, яка використовується для створення інтерфейсів користувача. Основною особливістю бібліотеки є компонентний підхід, який дозволяє розробникам створювати елементи і забезпечувати високу продуктивність веб-додатків. Ця бібліотека доволі часто застосовується для створення динамічних веб-додатків, включаючи погодні, де відображаються дані, які отримані за допомогою запитів і API. За допомогою компонентного підходу в React кожен елемент інтерфейсу є компонентом, який має власну логіку і стан, що спрощує структуру додатка і його подальшу підтримку. Також бібліотека використовує DOM, що значно спрощує процес розробки й оновлення компонентів застосунку і підвищує продуктивність. Завдяки використанню хуків можна реалізовувати запиті до API, налаштовуючи потрібні компоненти для отримання даних про погоду. Можливість використання серверного рендерингу для швидкого завантаження сторінок й обробки великого потоку запитів через серверні компоненти створює великий потенціал для застосування бібліотеки у великих проектах та масштабованих системах для створення великих застосунків й реалізації складних структур. Широкий потенціал та великі можливості створюють гарні умови для використання бібліотеки React у якості важливої загальної основної компоненти у процесі розробки структури та частин застосунку, враховуючи можливість масштабування, розширення та вдосконалення окремих компонентів й архітектури застосунку.

Vue - це JavaScript-фреймворк із відкритим вихідним кодом , використовуваний для створення користувацьких інтерфейсів і одно-сторінкових застосунків. Основною особливістю Vue є підтримка системи зв'язування даних, яка дозволяє автоматично оновлювати DOM при зміні даних моделі. Vue Router забезпечує навігацію між сторінками без перезавантаження всього застосунку, що покращує користувацький досвід та користування розробленим застосунком. Фреймворк підтримує реалізацію компонентної архітектури, де кожен компонент має свій шаблон, логіку та стилі, що спрощує

розробку складних інтерфейсів і їх повторне використання у подальшій розробці або вдосконаленні. Використовує різні інструменти для керування станом, які більш ефективні в невеликих розробках, ніж у великих або середніх системах. Підтримує роботу з сторонніми API через влаштовані функції інтеграції, а також більш легші налаштування без складних конфігурацій. Бібліотека має багато можливостей для реалізації малих й середніх застосунків та систем, розрахованих на середній об'єм обчислень та навантаження, що повністю відповідає вимогам при розробці серверних застосунків середнього масштабу.

Node.js - це середовище виконання JavaScript на стороні сервера, побудоване на JavaScript-рушії версії V8 від Google [14]. Також підтримує кластеризацію для використання багатоядерних систем, протоколів мережі для покращення продуктивності, сучасні функції JavaScript для інтерактивної розробки та реалізації серверної частини. Це платформа з відкритим кодом, яка дозволяє розробникам використовувати JavaScript для створення різноманітних серверних застосунків замість традиційних мов програмування на сервері. Node.js використовує асинхронну, подієво-орієнтовану модель, що робить його легким та ефективним для обробки багатьох численних запитів одночасно. Сам же Node.js є доволі ефективним для створення власних API зі своїми налаштуваннями, так як має наявність легкої обробки JSON-даних, які є поширеним форматом для обміну та передачі даних в більшості наявних API. Завдяки наявності у Node.js асинхронності та масштабованості надає можливості розподілу навантаження й забезпечення високої продуктивності у розробці та реалізації різних застосунків. Однією з переваг Node.js є можливості інтеграції та використання підтримуваних модулів та бібліотек через доступний менеджер типу npm, що дозволяє використовувати доступні та готові рішення й алгоритми для функціональності та покращеної розробки. У Node.js існує доволі потужний інструментарій, який має вбудовані інструменти для роботи з файловою системою, мережевими протоколами, буферами, потоками даних. Node.js має широке використання у сфері мікросервісної архітектури, який ідеально підходить для створення легких, автономних мікросервісів, що можуть бути

розгорнуті незалежно один від одного, забезпечуючи комфортну й безперебійну роботу окремих компонентів або цілих систем.

Express.js – фреймворк для розробки веб-додатків й роботи з API, створений переважно для взаємодії з Node.js та серверної розробки[15]. Загалом використовується для створення серверних застосунків, реалізації та налаштування API, обробки HTTP-запитів за допомогою механізму маршрутизації. За допомогою свого мінімального функціоналу та наявності так званого проміжного програмного забезпечення Middleware можливо легко розширювати функціональність додатків за допомогою модулів, які виконуються під час обробки запитів, забезпечуючи такі функції як авторизацію, логування чи обробку статичних файлів. Фреймворк Express надає зручні методи для роботи з HTTP-запитами та відповідями, включаючи управління заголовками, статусними кодами та типами вмісту[17]. Ще фреймворк має добру інтеграцію з різними базами даних через наявні та доступні модулі Node.js, можливість масштабування через доступну кластеризацію та та розширюваність функціональності завдяки великій кількості додатків npm, що розширює базову загальну функціональність для вирішення різних задач.

Axios є популярною HTTP бібліотекою для JavaScript, яка полегшує створення HTTP-запитів як у браузері, так і в середовищі Node.js[16]. Axios забезпечує безперебійну роботу API в різних середовищах, що дозволяє створювати та обробляти асинхронні операції, використовуючи сучасні функції JavaScript. Підтримка можливості перехоплення та модифікації запитів або відповідей, що розширює можливості та корисно для різних задач, таких як управління авторизацією або логування запитів. Використовується як на клієнтській стороні часто з додатковими React і Vue, так і на стороні сервера по типу Node.js та Express. Бібліотека підтримує автоматичну обробку трансформації JSON-даних, що означає оптимізацію коду для більш швидкої роботи. Наявність надійних механізмів обробки помилок дозволяє розробникам відстежувати та керувати різними типами помилок, таких як мережеві проблеми чи HTTP-помилки різних типів. Завдяки широкому набору опцій конфігурації

бібліотеки користувачі Axios отримують контроль над HTTP-запитами та їх налаштуванням, дозволяючи реалізовувати різні сценарії та завдання, від викликів API до складних продуктивних систем запитів.

PostgreSQL - об'єктно-реляційна система управління базами даних з відкритим кодом, яка багато років розробляється та оновлюється командою спільноти розробників, що отримала репутацію надійного, функціонального та високопродуктивного рішення[19]. PostgreSQL забезпечує високий рівень відповідності стандартам SQL, одночасно розширюючи їх власними можливостями для складних задач та сценаріїв використання. Підтримує паралельне виконання запитів для швидшої обробки великих даних. Завдяки можливості розширювання та підтримки складних типів даних система має власний інструментарій для створення власних типів даних, користувацьких типів та налаштованих масивів, що значно поширює можливості використання PostgreSQL. Через використані розширені механізми транзакцій, підтримки синхронної та асинхронної реплікації система забезпечує працездатність та надійність системи з функціями відмовостійкості та захисту, враховуючи використання управління доступом та автентифікації. PostgreSQL краще пристосована для аналітичних навантажень та роботи з великими базами, тому широко використовується у великомасштабних проектах**.**

MySQL - це одна з найпопулярніших реляційних систем управління базами даних з відкритим кодом, яка використовується для збереження, обробки та управління даними за допомогою наявного інструментарію [18]. Ця система має простий та зрозумілий інтерфейс та багато додаткових інструментів налаштування, що значно полегшує роботу з базою даних. MySQL оптимізована для швидкого виконання операцій, має ефективні та надійні алгоритми обробки запитів та кешування результатів, що забезпечує високу продуктивність навіть при значних навантаженнях. Має базовий рівень безпеки та систему контролю, підтримує різні механізми зберігання та має кросплатформність на різних операційних системах, а також використання стандартів SQL та налічує безліч додаткових бібліотек та додатків для моніторингу, інтеграції та контролю, що

полегшує процес використання та робить його більш налаштованим під різні завдання. Завдяки своїм можливостям MySQL широко використовується у розробці web-застосунків, корпоративних систем та платформ, у яких потребується швидка та надійна база даних з підтримкою та інтеграцією.

Oracle Database – реляційна система управління базами даних, яка використовується для обробки й управління досить великими обсягами даних[20]. Вона має розширені можливості для роботи з транзакціями, використовуючи передові механізми кешування, оптимізації запитів та автоматичного управління ресурсами, що робить її стабільної навіть при високих навантаженнях при великих обсягах інформації. Oracle Database підтримує різні моделі зберігання даних, включаючи реляційну, документно-орієнтовану та графову, а також має вбудовані можливості для аналітики, машинного навчання та обробки великих даних. Високий рівень безпеки забезпечується механізмами аутентифікації, шифруванням, аудитом доступу та контролем прав користувачів. Підтримує кросплатформеність, працюючи на Windows, Linux, UNIX та хмарних середовищах, що робить її універсальним рішенням для різних підприємств. Використовується в фінансових установах, корпораціях та масштабних бізнес-структурах, які потребують надійності та безпеки при досить величезних обсягах даних та механізмах їх обробки й збереження.

Враховуючи необхідні вимоги при створенні застосунку, бібліотека React завдяки своїм параметрам підходить для її використання у процесі розробці. Також для реалізації серверної частини застосунку краще за все підходить Node.js та фреймворк Express, які допомагають створювати необхідні компоненти та серверні функції для працездатності web-застосунку. Для різних операцій з даними та API у розроблюваному застосунку бібліотека Axios забезпечить безперебійну роботу та необхідні налаштування для подальшої розробки. При потребі зберігання різноманітних даних, таких як даних користувачів або історії пошуку, потрібна реляційна система управління базою даних з можливістю алгоритмів обробки запитів, достатньої продуктивності та можливості керування й редагування, що робить MySQL необхідним вибором у

встановлених параметрах, необхідних для застосування обробки й збереження даних невеликого обсягу і обробки операцій з великої швидкістю та можливою інтеграцією з іншими модулями та компонентами за допомогою наявних можливостей.

Висновки до розділу 1

В першому розділі була розглянута й проаналізована інформація про розробку й створення web-застосунків, які базуються на визначених архітектурних підходах, наприклад таких як клієнт-серверній моделі у вигляді дворівневої архітектури, яка надає широкі можливості для повноцінної реалізації та роботи web-застосунку. Також були розглянути типи різних архітектур, підходи розробки web-застосунків й різні технології для реалізації окремих компонентів й функціональних можливостей додатків. Популярні RESTful API та GraphQL потрібні для взаємодії між клієнтом та сервером, обміну інформації та можливістю створити безпечне, швидке та надійне з’єднання для обміну необхідними даними. Важливими факторами при виборі цих технологій та засобів є інтерактивність, можливість інтеграції до свого застосунку, прийнятна масштабованість та адаптивність.

Важливість вибору технологій web-застосунків є одним з надважливих пунктів при розробці архітектури, компонентів та основної частини застосунку. Використовувана мова програмування JavaScript є однією з найпопулярніших інструментів та використовується для розробки web-застосунків і серверних рішень різних параметрів та обсягів. При цьому виборі у розробці підходить бібліотека React, який призначений для створення інтерфейсів і розробки динамічних web-застосунків. Основна концепція та перевага бібліотеки React – це компонентний підхід. Усі елементи UI представлені у вигляді незалежних компонентів, які можна перевикористовувати та вкладати один в один, що забезпечує високу продуктивність та зручність при процесі розробці застосунку. Разом з тим у розробці використовується фреймворк Node.js, який є

асинхронним середовищем виконання JavaScript на сервері, що дозволяє виконувати створений код без браузера та доволі легко створювати серверні застосунки за допомогою наявних доступних можливостей і налаштувань. Використовується при створенні серверних застосунків та веб-додатків, обробці API та даних у реальному часі, потокової передачі даних та мікросервісних застосунків.

Бібліотека React Router DOM підтримує й забезпечує ефективну маршрутизацію при розробці веб-додатків та створення окремих діючих компонентів, а Axios досить добре підходить для створення HTTP-запитів в різних середовищах та умовах, налаштування та роботи з API та різними викликами створення запитів. Ці інструменти дозволяють налаштовувати обмін даних, власні мережі та виклики API за потребою, що доволі важливо в реалізації серверних застосунків.

Для збереження й обробки даних у web-застосунках потрібно використовувати бази даних, які мають бути надійними та безпечними, а також мати можливість налаштування та інтеграції. Були розглянути існуючи системи управління базами даних, такі як PostgreSQL, MySQL та Oracle Database, які мають різні характеристики та призначені для різних умов й середовищ роботи. Враховуючи розрахований обсяг даних та кількість дій з ними, найкраще всього для реалізації сховища підходить MySQL, яка підтримує доступну кросплатформеність, надійні алгоритми та необхідні інструменти налаштування для інтеграції та створення запитів різних видів. MySQL добре оптимізована для швидкого виконання різних операцій, роботи з обсягами даних середнього розміру та високу продуктивність при різних можливих навантаженнях, важких запитів та обробки помилок. Використання технологій та інструментів у вигляді бібліотек та фреймворків забезпечують процес розробки web-застосунку, надаючи можливість забезпечення швидкої обробки даних, необхідних запитів, високої продуктивності та надійної працездатності застосунку.

РОЗДІЛ 2
ПРОЕКТУВАННЯ І РОЗРОБКА СТРУКТУРИ WEB-ЗАСТОСУНКУ ДЛЯ АГРЕГАЦІЇ І АНАЛІЗУ МЕТЕОРОЛОГІЧНИХ ПРОГНОЗІВ

2.1 Методи прогнозування для агрегації і аналізу метеорологічних прогнозів

Прогнозування погоди є одною з важливих частин повсякденного життя багатьох людей, які залежать від навколишнього середовища. Зміни природи та постійне нестабільний стан атмосфери потребує постійного слідкування та формування на основі отриманих даних метеорологічних прогнозів на визначений час. З появою глобальної мережі й розвитком цифрових технологій формування точних прогнозів у будь-якій точці планети та їх поширення стало набагато простіше та надійніше, а нові технології у вигляді приладів відслідковування, супутників та інших приладів покращили точність прогнозів у багато раз, надаючи дані на більш довший період прогнозування. Це надало широкі можливості прогнозування погоди з більш поширеними даними, попереджувати о шкідливих або надзвичайних явищах, дізнаватися за короткий час стан атмосфери, що дозволяє краще планувати свої дії, корегувати плани та завчасно знати про опади чи туман, що можуть стати фактором зміни розпорядку дня звичайної людини в будь-якій місцевості.

В багатьох сферах життєдіяльності та економічного розвитку також не обійтися без метеорологічних прогнозів та точних даних зміни навколишнього середовища, таких як насамперед сфер сільського господарства, промисловості, мореплавства, авіаційних перевезень та будівництва. Компаніям, торговим підприємствам і малому бізнесу це стає в нагоді для планування і розробки своєї стратегії, а також планування розвитку, розширення і продажу своєї продукції чи надання послуг. Більшість сільських та фермерських господарств користуються сервісами надання погодних для стабільного розвитку свого господарства та росту врожаю при сприятливих умовах. Сучасні перевезення пасажирів, будь-то

авіаційним або морським транспортом також залежать від погодних умов, тому найактуальніші прогнози погоди в повітряних гаванях та морських портах користуються попитом серед авіакомпаній і диспетчерів судноплавства. Сфера будівництва також доволі залежна від погодних явищ та опадів, особливо будівництво великих проектів на кшталт транспортних узлів, доріг та стадіонів. Для державних структур метеорологічні сервіси надають доволі розширену й актуальну інформацію, що дає можливість не тільки для цілісного планування і управління ресурсами в регіонах, але й для попередження населення й підготовки спеціальних служб у випадку екстремальних погодних умов та природних катастроф, будь то висока температура, швидкий вітер, землетрус або цунамі. Аерокосмічні компанії та агенства розвитку космонавтики використовують прогнози для планування запуску супутників й обладнання у космос, так як для запуску ракет за межі Землі потрібна безхмарна стабільна погода без сильного вітру та опадів. В енергетичній сфері прогнозування погоди загалом використовуються для управління розподілу електроенергії, убезпечення електромереж від пошкодження внаслідок сильного вітру, опадів чи грози, а також планування ремонтів або побудови енергетичної інфраструктури.

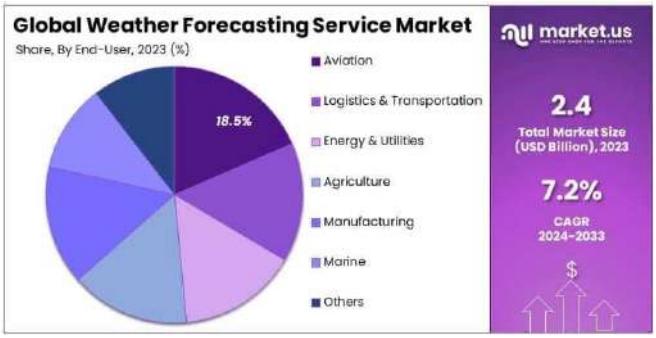


Рис. 2.1. Використання прогнозів погоди у різних сферах життєдіяльності

(розроблено з використанням матеріалів Market.us)

Популярність та актуальність даної теми підтверджується використання погодних сервісів більшістю населення та різних слоїв економіки, транспорту та промисловості (рис. 2.1.). Фактори зміни погодного середовища, температури і

атмосферних явищ можуть по різному впливати на сфери життя, промисловість, бізнеси і транспорт, тому в останній час розвиток технологій прогнозування, інструментів аналізу і передбачення дало змогу створити можливість отримувати якісні короткострокові і довгострокові прогнози, передбачуючи навіть кількість опадів, атмосферний тиск, температури води і повітря, та ще багато іншого. Через інтенсивний темп розвитку деяких сфер транспорту, кількості авіакомпаній, руху морських лайнерів для спеціалістів потрібні доволі точні дані про погодні умови в визначений місцевості, а також усі можливі зміни погодних умов чи небезпечні явища, що можуть порушити діяльність людини або нанести шкоду життю людей, промисловості чи інфраструктурі. Завдяки цим потребам та швидкого розвитку технологій достовірність і якість метеорологічних прогнозів стала доволі точною і відображає більшість змін погоди і різних показників, використовуючи як різні методи отримання даних, так і їх обробки, наприклад за допомогою побудованих алгоритмів або штучного інтелекту.

Також компанії, які надають послуги надання метеорологічних прогнозів і попередження про зміну погодних явищ, постійно змінюються під впливом нових технологій і комп’ютерних систем, а засоби спостереження за погодою і прилади отримання погодних показників, таких як температури або тиску, стають все точніше і краще, покращуючи результати спостережень і зменшуючи кількість можливих помилок при прогнозі [21]. Після розвитку портативних гаджетів нового типу, такі як смартфони, ноутбуки та інші типи переносних гаджетів, виросла роль мобільних додатків для надання користувачу даних про погоду в визначеній місцевості і прогнозу на деякий час, забезпечуючи доступ до потрібних даних з будь-якого місця при наявності доступу до глобальної мережі. Мобільні додатки надання прогнозу погоди, маючи добре налаштований інтерфейс й користувацький дизайн, все більше користуються попитом у більшості населення, особливо великих міст і агломерацій, а можливість за пару секунд отримати доволі точний прогноз на потрібний час стала необхідністю на сьогодні. Розвиток мобільних застосунків, що призвело до появи нових функцій, додання таких інтерактивних елементів, як погодних карт і функцій

відображення метеорологічних показників, покращило користування сервісами надання прогнозів погоди і зробило їх більш зручнішими. Створення нових алгоритмів й еволюція штучного інтелекту дала можливість обробляти великі масиви даних, отриманих з різних пристроїв і датчиків моніторингу погоди, значно покращили швидкість обробки даних і складання метеорологічних прогнозів, що призвело до появи можливостей в реальному часі спостерігати за погодними явищами і мати уяву про подальші зміни становища.

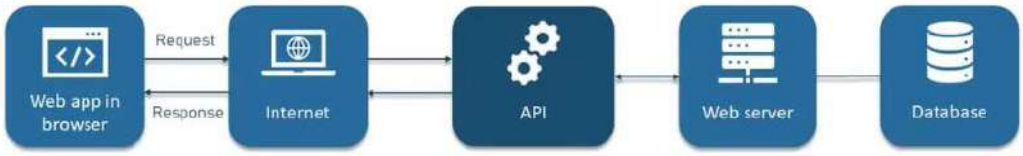


Рис. 2.2. Схема роботи API

(розроблено з використанням [35])

Метеорологічні сервіси надання прогнозних даних, крім основних послуг прогнозів, мають можливості надання інструментів, такі як API, для використання спеціалістами у своїх вебсайтах, мережах та програмних продуктах[22]. API, він ще інтерфейс програмування додатків, дає широкі можливості для взаємодії та отриманні різних метеорологічних даних, які доступні в рамках наданого плану користування цим інструментом за визначеними умовами (рис. 2.2.). Це дозволяє інтегрувати отримані прогнози погоди, метеорологічні дані й елементи відображення погодних явищ у веб-сайти, додатки на комп'ютері або телефоні та інших пристроях. API поділяються на різні типи, такі як публічні, приватні, партнерські та складні. Публічні API мають вільний відкритий доступ і розподіляються на безкоштовні і платні, які мають свої обмеження надання даних і взаємодії. Приватні API використовуються компаніями або бізнесом у своїх мережах для взаємодії між застосунками і програмним забезпеченням, які не мають відкритого доступу для інших. Партнерські API розробляються компаніями або фірмами для надання партнерів у своїй сфері, наприклад для синхронізації даних або обміну

повідомленнями. Складні API складаються з кількох API та мають складну структуру, тому що налаштовані виконувати багато дій, що дозволяє таким API працювати швидше за звичайні. Різні види API працюють за різними технологіями, наприклад REST API, SOAP API, WebSocketAPI й gRPC API. У більшості метеорологічних компаній та сервісів використовується REST API, який виглядає як URL-адрес, а для роботи з ними використовуються стандартні методи HTTP, дані цього API передаються у форматі JSON або XML. Для більшої надійності та складних даних використовується SOAP API, а для постійного та швидкого з'єднання без помилок використовуються WebSocketAPI або gRPC API, забезпечуючи високу швидкість передачі даних та стабільне з'єднання.

Основна частина алгоритмів для отримання якісних прогнозів погоди, базується на обробці супутникових даних, прогнозування за допомогою різних моделей, таких як GFS і ECMWF, чисельних методів прогнозування під назвою NWP і агрегації даних для коригування наявних прогнозів. Також для аналізу великих отриманих даних для збільшення точності та швидкості обробки частіше використовується штучний інтелект, який стає все краще і дозволяє сервісам отримувати покращені результати за більш короткий час[23]. Ще одною особливістю метеорологічних API від інших є те, що вони налаштовані на роботу у реальному часі, і мають різні параметри й налаштування для роботи з прогнозами різного часу, враховуючи необхідність відображення різних шарів, використання сотень параметрів атмосфери і постійні зміни погодних мап або показників. Актуальність у даному контексті грає дуже важливу роль, тому API і додаткові налаштування систем взаємодії налаштовані на постійні зміни отримуваних даних і оновлення прогнозів у деяких випадків дуже часто, для потреб користувачів у різних сферах і галузях. Завдяки використанню інтеграції з моделями прогнозування, які мають великі можливості при достатній кількості великих обчислювальних ресурсів і спеціалізованих алгоритмів, навіть у самих віддалених місцевостях без більшості якісної та ясної інформації є можливість заповнити прогалини у прогнозах і білі плями за допомогою розвинутих

інструментів передбачення і прогнозування. Більшість популярних метеорологічних сервісів, таких як OpenWeatherMap або AccuWeather, використовують складні продвинуті інструменти обробки та прогнозування, програми аналізу даних і штучного інтелекту для отримання якісних і швидких метеорологічних прогнозів. Інші продвинуті сервіси, наприклад такі як WeatherBit, використовують більш кращі налаштування та оброблюють багато даних для різних показників, а метеорологічний сервіс Tomorrow.io використовує штучний інтелект і машинне навчання для створення детальних прогнозів погоди. Метеорологічні API є доволі складними і інноваційними системами, оскільки вони працюють на перетині метеорології, пристроїв зчитування стану погоди, обчислювальної техніки й аналізу великих даних[24]. Це робить їх важливими не тільки для бізнесу або транспорту, але й для повсякденного життя, промисловості, будівництва, сільського господарства, туризму і інших сфер життєдіяльності людини у будь-якій точці планети.

2.2 Створення основної структури і схеми веб-застосунку для агрегації і аналізу метеорологічних прогнозів

Для реалізації основної структури web-застосунку, в якому використовується клієнт-серверна архітектура для надання необхідних можливостей користувачу, є потрібність створення та інтеграції необхідних функціональних компонентів у вигляді сторінок, елементів дизайну та бази даних з можливостями збереження та отримання необхідних даних за допомогою налаштованих запитів. Це допомогає дотримуватися основних вимог при розробці та досягти необхідних цілей розробки у вигляді повного функціонального web-застосунку, який є багатофункціональним та має обробку даних різних типів й призначення, які потрібні для роботи алгоритмів й забезпечення використання деяких функцій для користувача web-застосунку. Правильне функціонування компонентів та їх взаємодія між собою для забезпечення безперебійної роботи застосунку та процесу надання прогнозу

погоди з необхідною інформацією є основною задачею при реалізації застосунку та налаштування його частин для виконання необхідних зазначених функцій (рис. 2.3.). Це потребує створення необхідної визначеної архітектури і правильності зв’язків між частинами застосунку й компонентами, які відповідають за різні задачі, але можуть бути залежними від інших окремих елементів архітектури web-застосунку.

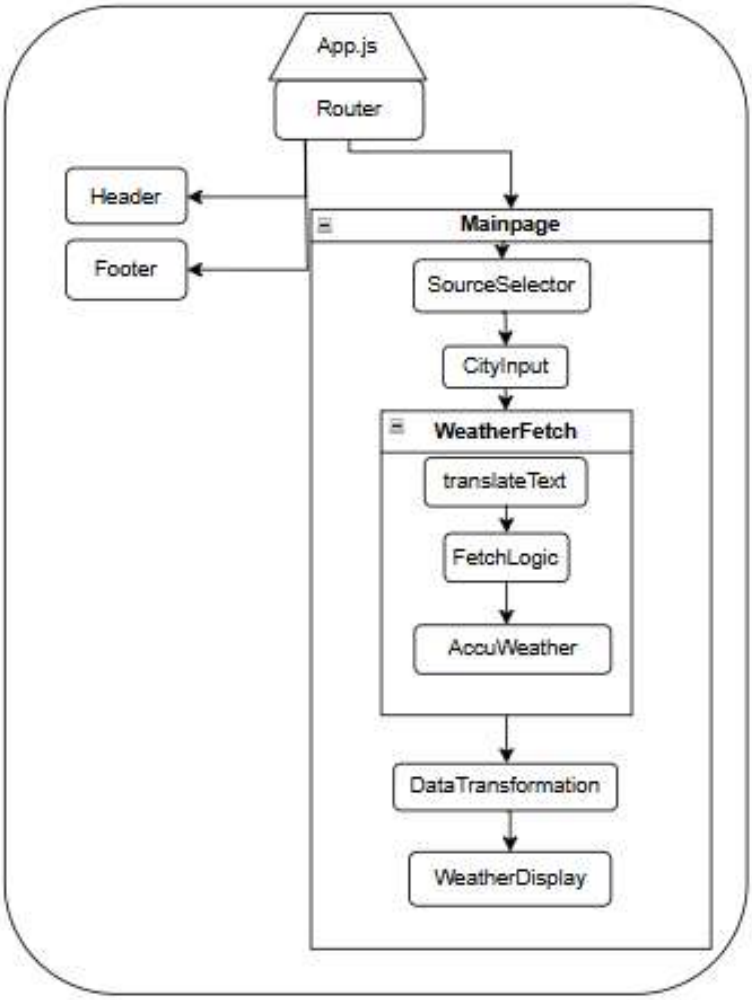


Рис. 2.3. Алгоритм відображення прогнозів

(розроблено автором)

Як можна зрозуміти, архітектура web-застосунку являє собою складну інтегровану систему для виконання конкретних задач, яка об’єднує між собою передові технології веб-розробки, принципи програмування додатків за допомогою React і зовнішні джерела метеорологічної інформації у вигляді API

для отримання актуальних прогнозів. Все це створює складну взаємопов'язану систему з розділенням на клієнтську і серверну частину (рис. 2.4.), де частина компонентів відповідає за відображення даних та створений інтерфейс з відповідним форматом надання даних користувачу, а інша – за працездатність серверної частини, обробку операцій з маніпулюванням даних та внутрішні алгоритми роботи деяких компонентів, маршрутизації та роботи з базою даних.

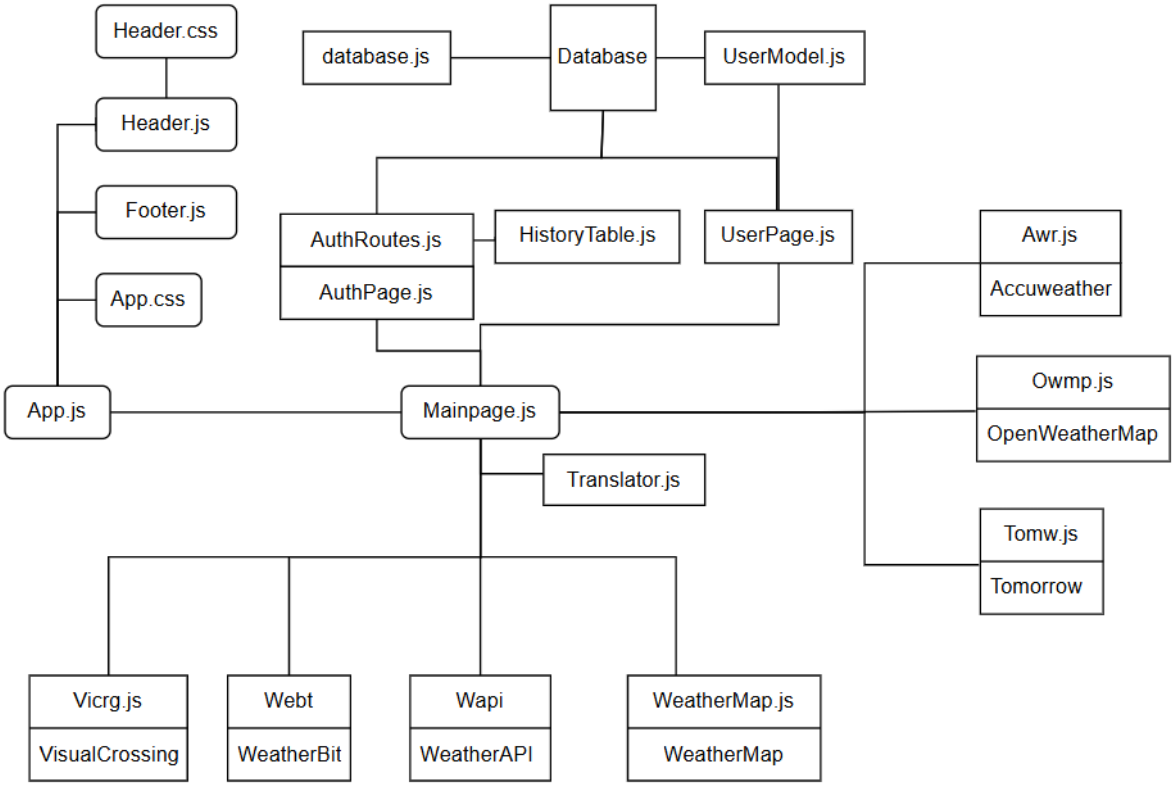


Рис. 2.4. Діаграма компонентів web-застосунку

(розроблено автором)

Основним кореневим компонентом є App.js, який об'єднує та координує всі інші частини, взаємодіючи як с Mainpage.js – ключовим компонентом роботи застосунку, так і компонентами для частин інтерфейсу і стилістичного оформлення – Header.js, Footer.js, App.css. App.css відповідає за глобальне стилістичне оформлення сторінки, забезпечуючи уніфікований дизайн для всіх елементів. Header.js відповідає за верхню частині інтерфейсу, яка містить заголовки та інші елементи. Footer.js формує нижній колонтитул сторінки,

забезпечуючи повноту і цілісність дизайну web-застосунку. Ключовий компонент в цій архітектурі є головною сторінкою – Mainpage.js, яка виконує основну логіку роботи в застосунку. Цей елемент відповідає за взаємодію з усіма компонентами джерел погодних даних, таких як Awr.js, Owmp.js, Tomw.js, Vicrg.js, Wapi.js, Webt.js і WeatherMap.js. Ці налаштовані компоненти реалізують взаємодію з відповідними зовнішніми API метеорологічних сервісів, надаючи основному елементу у вигляді Mainpage.js необхідну інформацію для відображення прогнозу[25]. Компонент AuthPage являє собою окрему сторінку з формою для авторизації користувача або реєстрації при першому користуванні, у разі чого створюється окремий акаунт з логіном та паролем, які надійно зберігаються в базі даних. Після входу до власної сторінки, яка відображається за допомоги елемента UserPage.js, користувач отримує можливість переглянути список історії запитів з різних вибраних доступних джерел web-застосунку, яка функціонує завдяки HistoryTable.js й відображає дату та час запиту, вибране джерело та місто, яке було в запиті до вибраного сервісу. Крім списку запитів, є можливість видалення усієї історії за необхідністю, а результаті потреби збереження є можливість її завантаження у текстовому форматі типу TXT , JSON або CSV, що доволі зручно для роботи з отриманими даними в різних редакторах чи програмах. Компонент Database.js забезпечує створення надійного з'єднання з базою даних MySQL, яке використовується всіма іншими компонентами системи. Компонент UserModel.js містить методи для реєстрації та входу користувачів до свого акаунта, надсилає запити та валідує необхідну інформацію для підтвердження входу користувача. Ще додаткові серверні компоненти, такі як AuthPage.js, authRoutes.js й userRoutes є наборами маршрутів окремих частин web-застосунку та існують для надання інформації про користувача для отримання необхідної інформації, обробки запитів для реєстрації та входу, а також для управління історії запитів, таких як додавання, перегляд або видалення збережених даних. Створена реляційна база даних дозволяє зберігати доволі значні обсяги інформації, включаючи логіни, імена, запити та дані про історію, забезпечуючи швидкий доступ до них завдяки надійним алгоритмам та

функціональним можливостям.

Основним компонентом архітектури є сам web-застосунок, створений з урахуванням визначених вимог й потреб користувачів. Механізм роботи web-застосунку базується на динамічному взаємозв’язку між компонентами, кожен компонент виконує специфічну функціональну роль. Важливий кореневий компонент App виступає контейнером усієї структури, який реалізує логіку маршрутизації за допомогою Router. Він забезпечує цілісне відображення компонентів MainPage, UserPage, AuthPage, Header і Footer, створюючи повний інтерфейс web-застосунку. Головна сторінка застосунку під назвою MainPage, є ключовим функціональним модулем у всій архітектурі, так як відповідає за всі операції з отриманням і відображенням погодних даних. AuthPage є сторінкою авторизації користувача, а UserPage забезпечує реалізацію сторінки користувача зі збереженою історією та іншими додатковими функціями. Архітектура головної сторінки побудована за допомогою хуків React, що дає можливість ефективно керувати станом компонентів. У web-застосунку використовується такі джерела як OpenWeatherMap, AccuWeather, WeatherAPI, Weatherbit, VisualCrossing і Tomorrow.io, тому користувач отримує можливість вибрати бажаний сервіс і отримати прогноз погоди у зазначеній місцевості. Кожне джерело реалізовано у вигляді окремого модуля з уніфікованим інтерфейсом прогнозу, що надає зручність та зрозумілість під час операцій з інтерфейсом головної сторінки. Механізм роботи застосунку передбачає послідовність дій, починаючи з вибору джерела погодних даних і введення міста, після чого виконується асинхронний запит до обраного метеорологічного сервісу і вже після чого застосунок відображає прогноз погоди у вказаному користувачем місті. Також окремим джерелом виступає WeatherMap за підтримки OpenWeatherMap, яка є інтерактивною картою погоди з різними шарами показників та можливістю отримати метеорологічні дані за допомогою використання маркера. Метеорологічна карта погоди надає різноманітні показники, такі як опади, вітер, тиск, вологість та температуру. Крім цього, додаткова інформація містить опис погодних умов, назву регіону чи населеного

пункту, а також координати місцевості у вибраній точці, де знаходиться поставлений користувачем маркер. Для перекладу з різних мов на англійську, яка є стандартом для використання багатьох метеорологічних сервісів, використовується налаштований перекладач – Translator.js, за допомогою використання API Google Translate, маючи можливість розпізнання тексту майже на всіх мовах світу й швидкому автоматичному перекладі на англійську для формування запита до вибраного джерела. Для неперервного процесу розроблена обробка потенційних помилок, таких як неправильно введене місто, недоступність мережі чи перекладача або метеорологічного сервісу. Використання можливостей асинхронних операцій є доволі важливим аспектом при розробці web-застосунку, тому як це забезпечує правильну та ефективну роботу з мережевими запитами, дозволяючи уникати блокувань і помилок під час завантаження даних, що доволі важливо при безперервній роботі та обробки даних при потенційних умовах великого навантаження [26]. Розроблена система висвітлення помилок надає користувачу необхідну інформацію у разі виникнення проблем роботи сторінок чи окремих компонентів, враховуючи внутрішні помилки, проблеми з мережею чи недоступність деяких сервісів. Інтерфейс web-застосунку доволі зрозумілий і простий завдяки використанням доступних інтерактивних елементів і механізмів введення даних для отримання метеорологічного прогнозу.

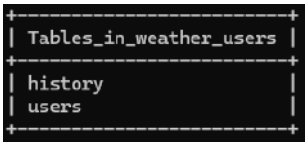


Рис. 2.5. База даних MySQL з таблицями користувачів та історії запитів

(розроблено автором)

Також важливим механізмом у web-застосунку є використання реляційної бази даних (рис. 2.5.) з системою керування MySQL, яка використовується для збереження даних користувача, таких як логін і пароль окремих акаунтів, історії

запитів у окремій таблиці, зберігаючи дані про дату та час окремих запитів, вибраних метеорологічних джерел та міст, які були у запитах до вибраних користувачем API. Це дозволяє отримувати доступ до щойно збереженої інформації та відображати необхідні дані в потрібних випадках, таких як історія запитів окремих сервісів чи використання збережених даних для аутентифікації користувачів web-застосунку. Використовувана при реалізації серверної частини web-застосунку СУБД MySQL має можливість збереження великої кількості даних, забезпечення безпеки даних через можливе шифрування й аутентифікацію, а ще гарну оптимізацію та достатній функціонал для забезпечення виконання операцій з даними різних масштабів та типів. Це дозволяє працювати з досить великими обсягами даних, створювати різноманітні запити та зберігати більшість таблиць та даних без критичних помилок та можливості їх втрати чи порушення безпеки, враховуючи переваги MySQL. Завдяки налаштованій маршрутизації, використання можливості авторизації та аутентифікації через спеціалізовані токени JWT для більшої конфіденційності та безпеки даних користувачів[27], а також створеного надійного механізму запитів до бази даних з необхідними параметрами вхід до власного кабінету або авторизація користувача відбувається без проблем та ускладнень, а налаштована конфігурація та з’єднання з базою даних дозволяють миттєво зберігати інформацію про здійснені запити та відображати їх у створеному списку для кожного користувача web-застосунку окремо, показуючи додаткові параметри та необхідну інформацію щодо виконаних дій у web-застосунку різних часових періодів.

2.3 Алгоритми і методи роботи застосунка і його властивостей для агрегації і аналізу метеорологічних прогнозів

Web-застосунок для агрегації і аналізу метеорологічних прогнозів має багато різних компонентів для виконання необхідних задач, які допомогають виконувати основні завдання у вигляді надання потрібних даних користувачу

застосунку. Усі реалізовані компоненти, для роботи з даними чи сервером мають налаштовані алгоритми роботи й механізми взаємодії друг з другом, створюючи єдину взаємопов’язану архітектурну систему. Більшість алгоритмів у структурі та створених компонентах орієнтована на обробку та надання метеорологічних даних у різних форматах для користувача, а також збереження інформації чи її відображення, які повинні забезпечити безперебійну роботу застосунку при можливості виведення помилок у різних ситуаціях, наприклад помилки обробки даних або недоступності вибраного джерела.

Основна задача застосунку для користувача основана на наданні отриманих метеорологічних даних, сформованих в потрібний прогноз на можливий термін часу, який залежить від джерел та можливостей надання прогнозу погоди на майбутній час. Це здійснено завдяки функціоналу головної сторінки, взаємодії компонентів джерел та запитів до доступних API, щоб отримати дані у потрібному вигляді та представити їх вже у зручному для зору користувача web-застосунку. Важливу роль для роботи web-застосунку відіграють сторонні API метеорологічних сервісів, таких як AccuWeather, OpenWeatherMap, WeatherAPI, Weatherbit, VisualCrossing і Tomorrow.io, що надають досить масштабні можливості для отримання доступу до погодних даних і прогнозів, які мають багато різноманітних показників і додаткових даних. Це забезпечує для користувача отримання актуальної й подробної інформації, можливості швидкого оновлення даних через налаштовані запити до API, які отримують найактуальнішу інформацію з бази даних відповідного джерела і надсилають її у відповіді, після чого відповідні механізми web-застосунку формують на їх основі прогноз у визначеному форматі. Це дозволяє за лічені секунди отримувати більшість доступних даних погодного стану від метеорологічного сервіса у визначеній місцевості, від температури та швидкості вітру до тиску та погодних явищ в поточному стані, що дозволяє отримати достатню інформацію о погодних умовах. Також не варто забувати про функціональність окремих компонентів web-застосунку для працездатності алгоритму отримання даних, а створена структура окремо для цього повністю

задовольняє необхідні потреби та відповідає визначеним вимогам.

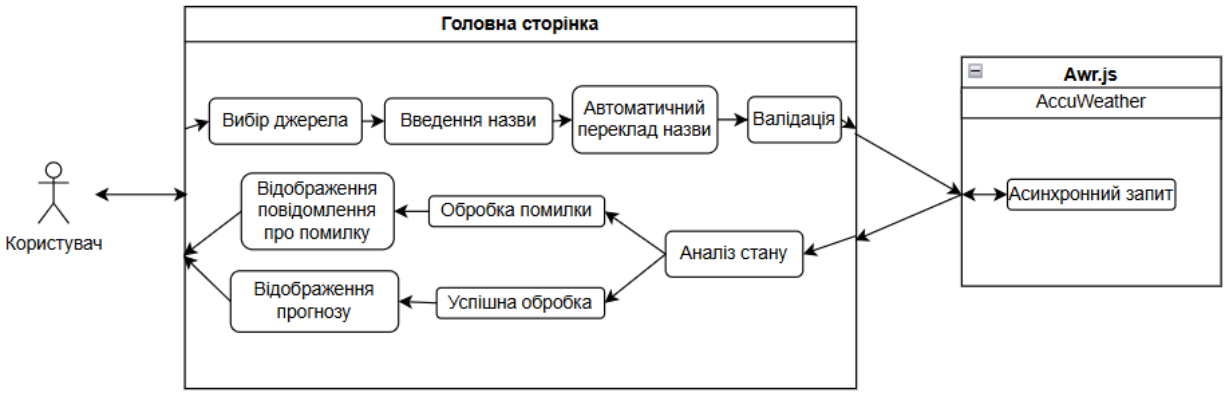


Рис. 2.6. Діаграма прецедентів - алгоритм отримання прогнозу погоди (розроблено автором)

Зазначена на рисунку 2.6 діаграма прецедентів показує алгоритм роботи web-застосунку для отримання прогнозу погоди починаючи з дій користувача і закінчуючи отриманням метеорологічних даних. Початковою точкою взаємодії є сам користувач, який починає користування web-застосунком і робить вибір джерела, в даному випадку постачальника погодних даних під назвою AccuWeather, що забезпечується знаходженням у Mainpage змінної станом, яка зберігає поточний вибір користувача web-застосунку. Інтерактивні кнопки з іншими джерелами дозволяють користувачу вибирати іншого постачальника даних про погоду на своє вподобання, даючи великий вибір і можливість використання інших метеорологічних сервісів. Після вибору користувач переходить до введення назви міста у відповідне поле вводу, яке також реалізоване у Mainpage. При введенні відбувається валідація для перевірки того, чи ввів користувач непорожнє значення у поле вводу, щоб запобігти можливим проблемам в наступному етапі цього процесу. Вбудований перекладач на основі Google Translate автоматично перекладає назви на англійську[28], якщо користувач ввів у текстове поле іншою мовою, тому що більшість метеорологічних сервісах використовує дані й інформацію тільки англійською мовою. Після цього відбувається процес отримання даних про погоду від

обраного джерела, який реалізовано в компоненті Awr.js. Тут визначено функцію fetchWeather, яка створює запит до визначеного згідно з вибраним джерелом API сервісу, використовуючи вказаний API-ключ відповідного сервісу. Функція надсилає асинхронний запит, обробляє отриману відповідь і повертає структуровані метеорологічні дані з показниками. Після чого у Mainpage реалізується аналіз стану, який впливає на подальші дії. Якщо відповідь успішна і немає помилок, інформація про прогноз зберігається за допомогою weatherData і відображаються користувачеві у спеціальному форматі прогнозу погоди. Якщо відбувається помилка при завантаженні даних або підключенні деяких сервісів, значення змінної error оновлюється і користувачеві застосунку відображається відповідне повідомлення про помилку, що може надати ясну інформацію про збій роботи web-застосунку.

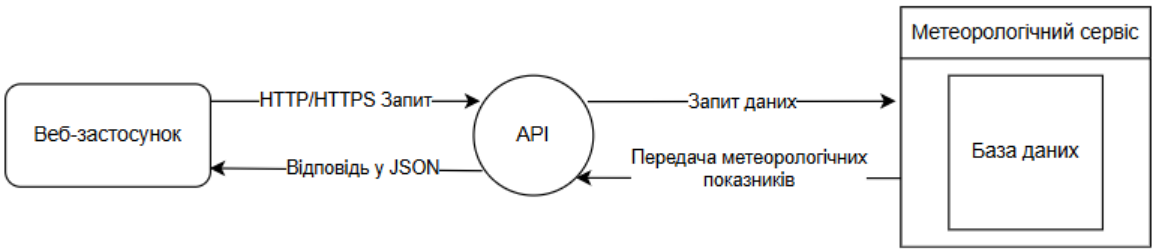


Рис. 2.7. Механізм взаємодії застосунку з сервісом через API

(розроблено автором)

Ще одним доволі важливим механізмом web-застосунку є взаємодія застосунку з сервісом прогнозу погоди, оскільки саме через цю взаємодію відбувається отримання актуальних метеорологічних даних для складання прогнозу погоди і його подальшого відображення користувачу web-застосунку[29]. Цей процес, зображений на рисунку 2.7, включає кілька послідовних етапів, які забезпечують швидкій і надійний обмін метеорологічних даних між застосунком і сервером зовнішнього API. Основними етапами цієї взаємодії є формування запиту, відправка серверу, отримання відповіді в стандартизованому форматі, обробка даних у web-застосунку і відображення їх

у вигляді прогнозу погоди користувачу.

На першому етапі користувач web-застосунку вводить назву міста або, яке передається як змінна у функцію, що відповідає за формування запиту до сервера відповідного API. Переклад робить переклад введеного тексту на англійську мову у разі необхідности, якщо користувач шукає місто або регіон іншою мовою. Перед відправкою запита проводиться перевірка даних, щоб уникнути можливих помилок, таких як порожнє поле або недопустимі символи. У разі виявлення помилки користувач отримує відповідне повідомлення про помилку, сповіщаючи про проблему і її характер по можливості.

На другому етапі після перевірки web-застосунок формує запит до зовнішнього API через використання методу Get для отримання даних. Запит складається з кількох елементів: URL-адреси API, параметрів самого запиту і унікального API-ключа, який необхідний для ідентифікації клієнта й безпечного доступу для обміну даних і підключення.

Після отримання запиту на наступному етапі сервер вибраного API приймає запит й обробляє його, аналізуючи зміст і звертаючись до наявної бази даних метеорологічного ресурсу, після чого отримує усю необхідну інформацію й формує відповідь. Відповідь надсилається у вигляді структурованого об’єкта, найчастіше у форматі JSON – легким для обробки та розпізнання і важливим для забезпечення чіткої цілісності структури даних[30]. У сформованій відповіді міститься необхідна інформація про стан погоди й показники для вказаного міста або регіону.

Після отримання відповіді від сервера алгоритм web-застосунку переходить до її фактичної обробки, трансформуючи дані таким чином, щоб вони могли бути легко використані для відображення в інтерфейсі web-застосунку. Текстовий формат погодних умов передається у відповідну секцію з текстом, а числові параметри формуються у зручному візуальному вигляді для користувача.

Останнім етапом є відображення отриманих даних у визначеному елементі інтерфейсу застосунку. Для цього використовуються технологія DOM, яка

дозволяє динамічно змінювати вміст веб-сторінки без її перезавантаження. Розроблений алгоритм забезпечує надійність і швидкість обробки запитів, а також зручність і простоту використання для користувачів web-застосунку.

Досить важливою частиною у системі web-застосунку є взаємодія користувача з базою даних через web-застосунок, а точніше збереженою інформацією та історією запитів, яка зберігається для надання користувачу у візуальному вигляді в власному кабінеті застосунку.

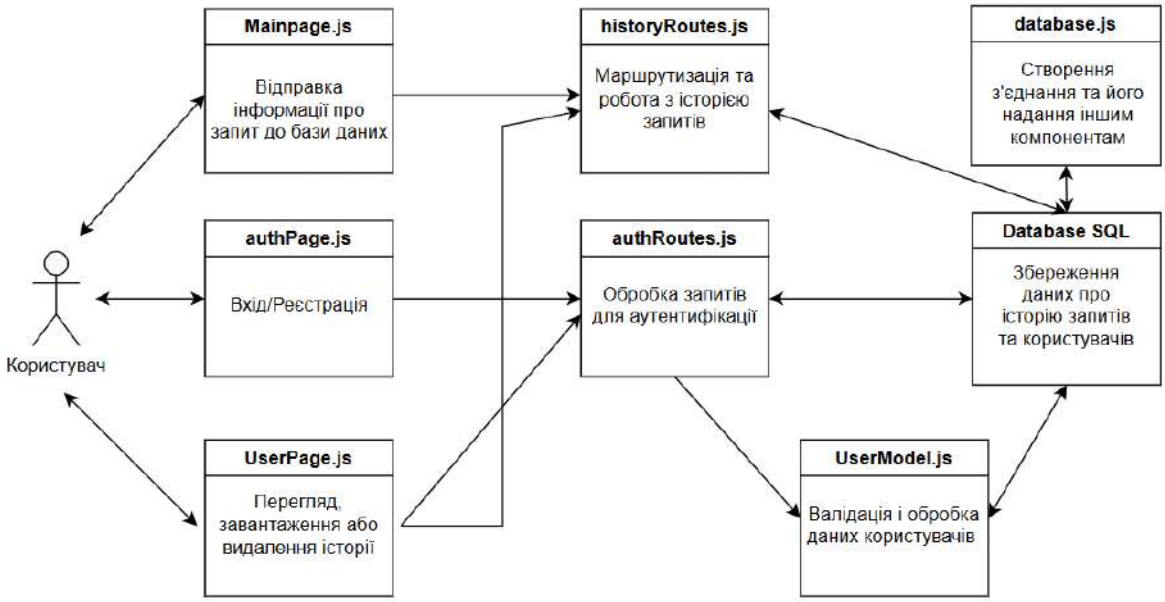


Рис. 2.8. Взаємодія користувача з базою даних

(розроблено автором)

Реалізований механізм отримання користувачем даних і можливістю аутентифікації у кабінету для перегляду історії запитів передбачає реалізацію багатофункціональної структури для обміну даних (рис. 2.8.), їх обробки, валідації інформації користувача та забезпечення конфіденційності і безпеки. Збереження історії запитів відбувається в Mainpage.js, коли авторизований користувач отримує прогнози від визначених джерел, а компонент відправляє інформацію про місто, джерело та час для збереження у базі даних, створюючи список історії запитів. Компонент AuthPage.js є інтерфейсом сторінки входу і відповідає за процес авторизації або аутентифікації користувача, відправляючи

дані до сервера та після обробки відповіді у випадку успішної процедури та валідації перенаправляє користувача на власну сторінку кабінету з історією запитів. Сама сторінка, яка є компонентом під назвою UserPage.js, являється власним кабінетом користувача зі збереженою історією запитів та можливістю її перегляду, видалення або завантаження у різних форматах даних для зручності. UserPage.js взаємодіє з іншими компонентами та базою даних, отримуючи інформацію про відповідного користувача та для отримання його збереженої історії запитів.

Наступний компонент historyRoutes.js, який є важливим модулем маршрутизації, відповідає за додавання записів до історії запитів в базі даних, отримання історії запитів для конкретно визначеного користувача та очищення історії. authRoutes.js існує для обробки запитів, які пов’язані з аутентифікацією і відповідає за можливості входу та виходу користувача з акаунта, а також отримання інформації про поточного користувача web-застосунку. Компонент authRoutes.js при успішному вході і валідації пароля генерує окремий JWT-токен доступу, а ще може очищати його при виході користувача застосунку з власного кабінету. Компонент UserModel.js підтримує працездатну логіку роботи з даними користувачів, містить методи для реєстрації і входу користувачів, обробляє введені дані перед їх збереженням у базі даних. Компонент database.js є досить важливим для робочої конфігурації системи й роботи web-застосунку, оскільки є єдиною точкою підключення до бази даних і надає іншим компонентам цю можливість. Крім цього, даний компонент містить налаштовані параметри підключення до бази даних і стабільного з’єднання, тому це є важливим аспектом для функціонування серверної частини web-застосунку. Реляційна база даних MySQL використовується у web-застосунку для зберігання даних користувачів і історії запитів, а серверна частина реалізована за допомогою фреймворку Express[31], який забезпечує надійну взаємодію з реалізованою базою даних.

Висновки до розділу 2

В другому розділі були розглянуті методи прогнозування й аналізу метеорологічних прогнозів, основна інформація прогнозування та використання метеорологічних сервісів серед користувачів, бізнесу та промисловості, а також для попередження стихійних явищ та забезпечення безпеки населення. Було визначено, що в багатьох сферах життєдіяльності та економічного розвитку не обійтися без прогнозування погоди та постійного спостереження за станом атмосфери, оскільки не тільки плани або стратегії розвитку залежать від погодних умов, а ще більшість логістичних систем та транспортних комунікацій, наприклад авіація або морські судна, які постійно знаходяться у русі та забезпечують стабільні зв'язки між державами, населенням та економічними структурами. Більшість компаній, які надають послуги надання метеорологічних прогнозів і попередження про зміну погодних явищ, вдосконалюються і використовують новітні технології слідкування та прогнозування, такі як супутники, радари, інтернет речей IoT та штучний інтелект, який все краще допомагає вирішувати задачі з обробкою масивів великих даних та знаходити можливості будувати точніші та кращі прогнози з можливістю виявлення нових методів та вдосконалення існуючих інструментів. Більшість метеорологічних сервісів надають не тільки основні послуги в якості прогнозування або попередження, але мають можливості надавати послугу використання API у власних або комерційних цілях, для створення власних застосунків, отримання погодних даних або інших системах чи програмних продуктах. API часто поділяються на різні типи, такі як публічні, приватні, партнерські та складні, але в більшості випадків використовуються публічні та приватні, хоча партнерські й складні також мають свою вагу в кооперації бізнесу, промисловості та держави. API широко використовуються серед більшості додатків, інтеграції у застосунки та платформи, користуючись попитом серед багатьох розробників платформ, web-застосунків та іншого програмного забезпечення.

Для створення структури web-застосунку, були розглянути етапи

створення та інтеграції необхідних функціональних компонентів у вигляді сторінок, елементів інтерфейсу та бази даних для збереження та отримання необхідних даних за допомогою запитів, що є загальною важливою частиною для повної розробки web-застосунку. Було визначно, що правильне і стабільне функціонування компонентів та їх взаємодія для забезпечення безперебійної роботи застосунку та процесу надання актуального прогнозу погоди з необхідною інформацією є основною задачею при реалізації web-застосунку та налаштування його частин для виконання необхідних й зазначених вимог. Розглянуто створення клієнтської й серверної частини web-застосунку, де реалізована частина компонентів відповідає за відображення даних та інтерфейс застосунку з відповідним форматом надання даних користувачу, а інша – за працездатність серверної частини, обробку операцій з маніпулюванням даних та внутрішні алгоритми роботи деяких компонентів, маршрутизації й роботи з базою даних. Проаналізовано існуючи компоненти та їх основні функції, важливість у створеній системі та взаємопов’язані зв’язки з іншими елементами архітектури web-застосунку, які надають потрібні можливості та потрібні надавати необхідну інформацію користувачу без проблем, а в випадку помилок чи відсутності з’єднання надавати необхідні повідомлення про проблему з причиною помилки або недоступності зазначеної функції. Також СУБД MySQL, яка використовується для зберігання даних користувачів й історії запитів, має можливість збереження великої кількості різноманітних даних, забезпечення безпеки даних через шифрування й аутентифікацію, а ще достатню оптимізацію та необхідний функціонал для забезпечення виконання операцій з даними великих масштабів та різних типів.

Web-застосунок для забезпечення потреб й взаємодії для користувачів використовує налаштовані алгоритми роботи й механізми взаємодії компонентів застосунку друг з другом, створюючи єдину взаємопов’язану систему. Розглянута основна задача застосунку для користувача, яка основана на наданні отриманих метеорологічних даних від обраних сервісів, сформованих в необхідний для користувача метеорологічний прогноз на можливий термін часу,

який залежить від джерел та можливостей надання прогнозу погоди на майбутній час. Було проаналізовано, яку важливу роль для роботи web-застосунку і його можливостей відіграють сторонні API метеорологічних сервісів, які забезпечують для користувача web-застосунку отримання актуальної й детальної інформації про стан погоди й різну інформацію у визначеній місцевості. Розглянуто алгоритми отримання прогнозу погоди, відповідно застосовані у процедурі компоненти та роль деяких елементів функціональної частини web-застосунку, визначено важливість механізму взаємодії застосунку з сервісом через API для отримання актуальних метеорологічних даних і формування прогнозу погоди. Багатофункціональна структура для обміну даних дозволяє користувачу зберігати історії запитів при авторизації у web-застосунку та при необхідності видаляти чи завантажувати її на свій пристрій. Налаштована взаємодія компонентів дозволяє забезпечувати зберігання даних в реляційній базі даних MySQL, яка має ефективну працездатність та інтеграцію з серверними застосунками, що добре підходить для збереження даних web-застосунку.

РОЗДІЛ 3
РОЗРОБКА І ТЕСТУВАННЯ WEB-ЗАСТОСУНКУ ДЛЯ АГРЕГАЦІЇ І АНАЛІЗУ МЕТЕОРОЛОГІЧНИХ ПРОГНОЗІВ

3.1 Огляд інструментів і додатків для реалізації системи і виконання вимог розробки для агрегації і аналізу метеорологічних прогнозів

Для реалізації системи web-застосунку і певних характеристик, які потрібні для працездатності системи й компонентів, потрібен надійний і ефективний інструмент. Саме WebStorm, який є інтегрованим середовищем розробки й розроблений компанією JetBrains, забезпечує усім необхідним функціоналом й можливостями для реалізації потреб у вигляді системи web-застосунку. Цей інструмент, який є доволі професійним середовищем, має нахил в сторону розробки й підтримки web-додатків й серверних застосунків з використанням мов програмування JavaScript, TypeScript, HTML та CSS, а також підтримки популярних фреймворків по типу React, Node, Vue та інших додатків та платформ, які доповнюють основну програму для повноцінної розробки визначеного продукту[32].

Застосування WebStorm у процесі розробки web-застосунку обумовлене його розширеним функціоналом і широкими можливостями, які забезпечують продуктивність, зручність та підтримку достатньо багатьох наявних інструментів та технологій, що є просто необхідним при розробці застосунків і їх компонентів. Це дозволяє реалізовувати необхідні елементи структури у комфортному режимі з можливістю перегляду змін та можливих доповнень, забезпечуючи необхідний рівень розробки та реалізації взаємопов’язаних структур. Підтримка більшості доступних фреймворків, які дозволяють при використанні JavaScript й CSS створювати складні структури та реалізовувати багатофункціональні елементи, забезпечує можливість створення повноцінного застосунку і необхідних механізмів його роботи для потреб користувача.

Однією з важливих переваг WebStorm є інтегроване в середу розробки

автодоповнення кода, що значно прискорює реалізацію необхідних компонентів і зменшує ймовірність помилок при розробці. Крім цієї вбудованої властивості, WebStorm забезпечує статичний аналіз коду в режимі реального часу, що дозволяє оперативно виявляти потенційні помилки, порушення стилю кодування та інші дефекти ще на етапі розробки програмного продукту. Також ще однією суттєвою можливістю середовища є вбудоване тестування розроблюваного застосунку і перевірка JavaScript-коду як у браузері, так і на сервері через застосовані платформи, наприклад Node.js, що забезпечує зручну перевірку коректності роботи модулів без необхідності використання сторонніх засобів і довгої ручної перевірки усього контексту й алгоритмів.

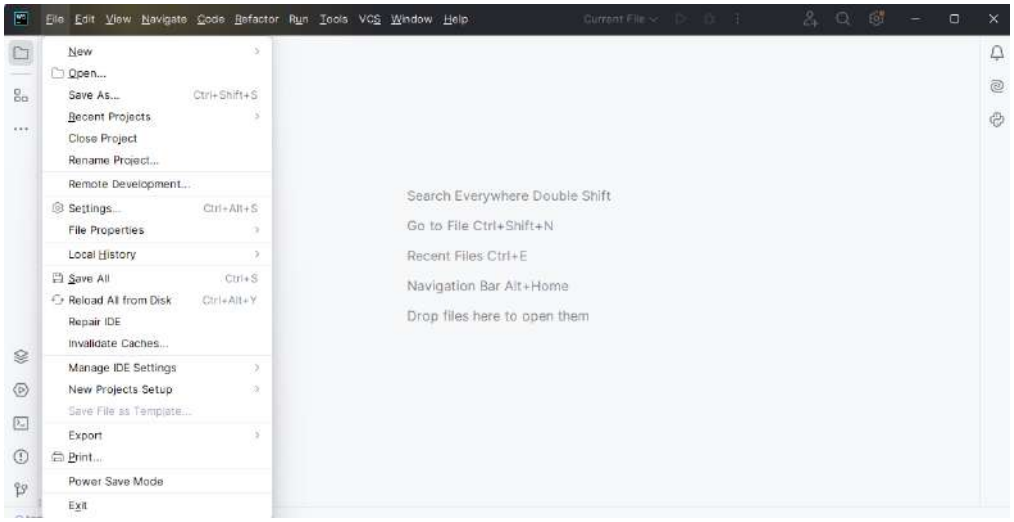


Рис. 3.1. Інтерфейс середовища розробки WebStorm

(розроблено автором)

WebStorm має механізм зберігання змін окремих частин або усього проекту, що дозволяє у разі помилкового рішення або неможливості знаходження і вирішення проблеми повернутися до попередніх версій, маючи можливість перегляду самих змін і частин нового коду, що доволі зручно для середніх й великих проектів. Інтерфейс середовища та наявні можливості редагування та налаштувань забезпечує усім необхідним при розробці різноманітних програм та застосунків (рис. 3.1.). Середовище гарно підходить для роботи навіть з великими та масштабними проектами, де WebStorm завдяки

своїй оптимізації продуктивності та функціям навігації по коду суттєво полегшує орієнтування у складних кодових базах. Також дане середовище добре підтримує автоматичне форматування коду відповідно до обраних стандартів, використання доступних й кастомних шаблонів коду, а також інтеграцію з менеджерами пакетів, такими як npm та yarn, що значно спрощує роботу з поточними залежностями проекту. Окремої уваги також заслуговують налаштування та встановлення плагінів, налаштувань й додаткових інструментів, які можливо встановлювати прямо з відповідного розділу загальних налаштувань, яке дозволяє переглядати доступні додаткові можливості і встановлені плагіни з можливістю відключення або оновлення у разі потреби. Це дозволяє фактично здійснювати прямий та швидкий контроль за встановленими можливостями в окремих проектах, підключати важливі модулі або фреймворки, і мати під рукою багатий інструментарій з широкими можливостями, які доступні в будь-який час і можуть допомогти як при розробці невеликих додатків або програм, так і достатньо великих застосунків або багатокомпонентних систем.

Використовувана при розробці web-застосунку мова програмування JavaScript, яка застосовується для розробки застосунків і серверних рішень, мобільних і комп’ютерних програм, доволі добре підходить для реалізації web-застосунків з серверною частиною. JS є кросплатформною мовою і підтримується у багатьох браузерах, серверах та інших середовищах. Мова підтримує об'єктно-орієнтоване, функціональне та подієво-орієнтоване програмування, що забезпечує велику гнучкість і можливості під час розробки різноманітних систем. JavaScript виконується в середовищі браузера або на сервері без необхідності компіляції, що дозволяє здійснювати миттєву перевірку та інтеграцію змін. Крім цієї функціональності, JavaScript має доступність асинхронного програмування через різноманітні доступні механізми, що є критично важливим і необхідним для розробки високопродуктивних web-застосунків із великою кількістю мережевих запитів. У етапі розробки JS використовується разом з розміткою HTML й стилями CSS, що допомагає

створювати функціональний і гарний інтерфейс разом з працездатними компонентами й алгоритмами певних елементів. JS може використовуватися на стороні сервера завдяки Node.js, для створення і підтримки серверної частини застосунку.

Для реалізації web-застосунку і виконання вимог, таких як створення інтерфейсу, застосовується бібліотека React у якості інструмента для розробки користувацьких інтерфейсів, яка була розроблена Facebook з метою забезпечення високої продуктивності при розробці, великої масштабованості та достатньої гнучкості у побудові сучасних web-застосунків. Загальна основна концепція бібліотеки React існує у вигляді компонентного підходу, де елементи UI представлені у вигляді незалежних компонентів, які можна перевикористовувати та вкладати один в один, що дозволяє розробникам розмежовувати інтерфейсну частину на незалежні, багаторазово використовувані самостійні блоки з власною логікою та станом. Використання цієї бібліотеки підвищує продуктивність розробки застосунків й має можливість інтеграції с іншими бібліотеками та фреймворками, тому вважається одною з популярних та широко використовуваних бібліотек в JavaScript. Однією з ключових характеристик й переваг React є впровадження і підтримка **Virtual** DOM — абстрактного представлення дерева розподілу елементів інтерфейсу, яке дозволяє значно мінімізувати кількість маніпуляцій із реальним DOM браузера, що сприяє підвищенню швидкості оновлення сторінки та загальну продуктивність застосунку, оскільки оновлення виконуються лише в тому місці, де дійсно відбулися зміни у стані даних. Завдяки можливості використання хуків для керування станом можна створювати запиті до API, налаштовуючи більшість компонентів для ефективної працездатності і роботи web-застосунку. Бібліотека React підтримує використання додаткового й корисного JSX — розширення синтаксису JavaScript, яке дозволяє описувати структуру інтерфейсу у вигляді типу HTML коду при реалізації та налаштуванні JavaScript-скриптів у час процесу розробки та створення компонентів. Це полегшує процес реалізації необхідних компонентів й скриптів та підвищує зручність сприйняття

створеного коду завдяки наочності структури компонентів.

Ще одним важливим інструментом є Node.js, який є асинхронним середовищем виконання JavaScript на сервері, який дозволяє розширювати можливості JavaScript за межі браузера і забезпечувати ефективну обробку запитів у реальному часі, високу продуктивність та здатність обслуговувати велику кількість одночасних підключень, що робить його надзвичайно придатним та важливим для розробки сучасних web-застосунків. Node.js має багато можливостей, таких як обробку запитів та отриманих відповідей, роботу з наявною файловою системою, асинхронним програмуванням та обробкою потоків, а також використовується для доступної взаємодії з використовуваними базами даних. Також Node.js підтримує виконання JavaScript-коду на сервері, що дає можливість використовувати єдину мову як для клієнтської, так і для серверної частини web-застосунку, що сприяє процесу розробки, зменшує кількість необхідних технологій і полегшує реалізацію повної структури й необхідних компонентів. Великою перевагою Node.js є наявність доступною екосистеми модулів, яка управляється через менеджер пакетів **npm**, найбільшого репозиторію відкритого коду у світі, що дозволяє швидко й безпечно інтегрувати в проєкт готові бібліотеки для своїх потреб у ході розробки. Node.js часто використовується для реалізації мікросервісної архітектури завдяки своїй легкості, швидкості запуску процесів та підтримці обміну повідомленнями між сервісами. У багатьох випадках саме завдяки Node.js реалізується ефективна взаємодія між web-застосунком та сторонніми сервісами через API.

Також при розробці web-застосунку в нагоді стає Express.js – фреймворк для розробки застосунків й роботи з API, який створений переважно для взаємодії з Node.js та серверної розробки. Сам Express за допомоги своєї зрозумілої структури та функціоналу забезпечує розширення можливостей застосунків і їх функціональності, маючи доволі зручні методи для роботи з HTTP-запитами та відповідями. Фреймворк застосовується для розробки серверної частини застосунку, а також різних механізмів обробки даних та відповідей від API[33]. Ще даний фреймворк має достатньо налаштовану

інтеграцію з базами даних через наявні та доступні модулі середовища Node.js, можливість масштабування через доступну кластеризацію й розширюваність, що дає змогу використовувати фреймворк у системі web-застосунку та для реалізації деяких його компонентів, забезпечуючи роботу серверних функцій та взаємодію з API.

Важливу роль також має бібліотека Axios, яка призначена для створення HTTP-запитів як у браузері, так і у середовищі Node.js. Вона має достатні можливості та зручний інтерфейс для роботи з асинхронними запитами та їх налаштуваннями, що дозволяє ефективно та безпечно взаємодіяти з API, існуючими серверами та сторонніми сервісами. Сама ж бібліотека Axios дозволяє взаємодіяти з REST API для отримання та відправки даних, реалізацію та обробку даних при процесі авторизації, оптимізацію більшості запитів та підтримку можливості перехоплення й модифікації запитів або відповідей, що значно розширює можливості та корисно для таких задач як управління авторизацією або логування запитів. Завдяки наявним можливостям централізованого керування заголовками та запитами, Axios ефективно використовується для реалізації авторизованих запитів, зокрема через токени JWT або API-ключі.

Для збереження даних, які циркулюють у створеній системі застосунку та призначені для працездатності функціоналу аутентифікації та відображення історії запитів, використовується система управління базами даних MySQL для збереження, обробки та управління даними. Вона створена на основі моделі клієнт-серверної архітектури та має підтримку усіх доступних SQL-команд, що забезпечує розширені можливості для створення складних запитів, маніпуляції даними, формування звітів і забезпечення зв’язків між таблицями. Крім цього, у MySQL присутні ефективні та надійні алгоритми обробки запитів та кешування результатів, а також система управління забезпечує високий рівень безпеки та систему контролю, підтримує різні механізми зберігання та має кросплатформність на різних операційних системах. Застосування MySQL у розробці web-застосунку допомагає надійно та ефективно зберігати дані

користувачів й історію запитів, обробляти велику кількість даних у разі потреби та забезпечувати конфіденційність й працездатність навіть при великих навантаженнях, надаючи можливість постійного використання й обміну інформацією між компонентами застосунку та базою даних.

3.2 Розробка повноцінного функціоналу і впровадження необхідних можливостей для агрегації і аналізу метеорологічних прогнозів

Web-застосунок, який має можливість отримування метеорологічних прогнозів, виконує потрібні вимоги для потенційного використання функціоналу системи користувачами, яким потрібна швидка доступність та комфортність використання через реалізований інтерфейс. При створенні web-застосунку з агрегації метеорологічних прогнозів з надання прогнозу погоди враховувалися основні визначені функціональні вимоги, а також створена архітектура й алгоритм взаємодії застосунку з користувачем. За допомогою використовуваних бібліотек й фреймворків, таких як Axios та Express, реалізована взаємодія застосунку зі створеною реляційною базою даних, що задовольняє визначеним потребам й надає працездатний механізм зберігання історії запитів окремих користувачів за допомогою можливостей авторизації й створення власного акаунта у системі. Реалізований інтерфейс забезпечує комфортну й швидку взаємодію користувача з web-застосунком, а функціонал розроблений таким чином, щоб забезпечити доволі зручний і одночасно функціональний інструмент для отримання інформації про стан погоди та метеорологічні показники за запитом користувача, надаючи можливості порівняння й агрегації для прогнозів з різних метеорологічних джерел. Наявність різноманітних джерел, отримання метеорологічної інформації у щоденному прогнозі або графіках погодних показників надають користувачу потрібне представлення про погодний стан у визначеному місці, а інтегрований перекладач забезпечує переклад введеного міста для коректної відповіді сервісу й більшого шансу отримати коректний прогноз без помилок. Наявність доступних кнопок для входу в існуючий акаунт

або реєстрації для нових користувачів забезпечує швидке використання й створення кабінету для збереження історії запитів для різних потреб, а функція завантаження історії у вигляді файлу різних форматах, що доволі зручно для перегляду, аналізу або редагування у наявних текстових редакторах. Розробка web-застосунку починається з налаштування оточення, визначення користувацької та серверної частини, ініціалізації додаткових компонентів та загальної структури web-застосунку. Створена структура компонентів розбита на різні частині і функціонал(рис. 3.2.), які відповідальні за повну і цілісну працездатність веб-проекту, який створюється за допомогою WebStorm й наявних інструментів.

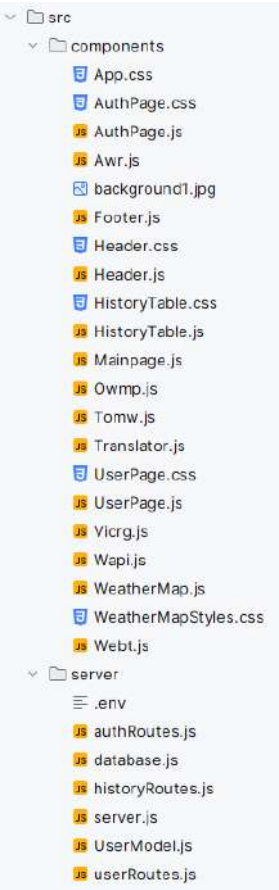


Рис. 3.2. Користувацька й серверна частина web-застосунку

(розроблено автором)

Розглядаючи структуру саме двох частин застосунку, можна розрізнити директорію components зі сторінками, модулями сервісів та файлів налаштувань

стилів для різних частин інтерфейсу і сторінок. У директорії server же знаходяться файли, які відповідають за маршрутизацію, роботу сервера та взаємодію з базою даних для збережень необхідних даних. Це дозволяє швидко орієнтуватися в наявній структурі та створених компонентах, реалізовувати певні розширення та масштабувати застосунок чи додавати певний функціонал без ускладнень, враховуючи реалізовану структуру та відповідну працездатність за допомогою React, Node, Express та інших наявних застосованих доповнень.

```
try {
    const translatedCity :any | undefined = await translateText(city);
    console.log("Перекладений город:", translatedCity);
    let data;
    if (source === "OpenWeatherMap") {
        data = await Owmp.fetchWeather(translatedCity);
    } else if (source === "AccuWeather") {
        data = await Awr.fetchWeather(translatedCity);
    } else if (source === "Weatherbit") {
        data = await Webt.fetchWeather(translatedCity);
    } else if (source === "WeatherAPI") {
        data = await Wapi.fetchWeather(translatedCity);
    } else if (source === "VisualCrossing") {
        data = await Vicrg.fetchWeather(translatedCity);
    } else if (source === "Tomorrow") {
        data = await Tomw.fetchWeather(translatedCity);
    }

    setWeatherData(data);
    localStorage.setItem("weatherData", JSON.stringify(data));
    localStorage.setItem("city", translatedCity);
    localStorage.setItem("source", source);

    try {
        await axios.post( url: "http://localhost:5000/api/history/add", data: {
            city: translatedCity,
            source,
        }, config: { withCredentials: true });
    } catch (error) {
        console.error("Error saving history:", error);
    }
```

Рис. 3.3. Частина вибору джерела й збереження історії запитів

(розроблено автором)

У головній сторінці Mainpage.js на рисунку 3.3 реалізована важлива функціональна частина, яка відповідає за отримання даних про погоду з зовнішніх метеорологічних сервісів API, вибору відображення показників залежно від джерела та налаштування відображення графіків окремих показників. У цьому компоненті реалізований механізм операцій для виконання

асинхронних запитів, налаштованої змінної source та перевірки значення для коректної роботи й виконання асинхронного запита до вибраного зовнішнього API, що підтримує можливість отримування погодних даних від багатьох метеорологічних сервісів. Також у цьому блоці коду реалізовано за допомогою використання бібліотеки Axios, яка дозволяє за допомогою створеного запиту зберігати історії в базу даних, яка потім відображається у кабінеті користувача.

Рис. 3.4. Відповідний компонент для відображення графіків показників

(розроблено автором)

Також різні компоненти джерел мають свої власні налаштування та показники, які відрізняються один від одного та оптимізовані саме під конкретного постачальника погодних даних, враховуючи можливості надання інформації від джерела та термін часу, на який створюється потрібний прогноз. Це створює умови для формування метеорологічних прогнозів за достатньо швидкий час, а наявність додаткового елемента у коді під назвою WeatherChart дозволяє формувати графіки коливань та змін різних показників отриманих прогнозів (рис. 3.4.), залежно від прогнозу й кількості днів, забезпечуючи візуальне відображення для більш легкого та швидкого сприйняття погодних даних користувачами web-застосунку. За допомогою налаштованої обробки різних типів помилок у web-застосунку є наявна можливість отримувати повідомлення про можливі проблеми в застосунку чи недоступність метеорологічних сервісів, якщо його робота була порушена й частина функціонала виявилася недоступною для користувача застосунку.

Наступний важливий компонент під назвою UserPage.js є сторінкою користувача та відповідає за функціональність сторінки, відображення історії запитів та внутрішніх функцій, таких як видалення історії або її завантаження у вигляді файлу різних форматів.

```
function UserPage() {  Show usages
    const navigate :NavigateFunction  = useNavigate();
    const [user, setUser] = useState( initialState: null);
    const [history :any[] , setHistory] = useState( initialState: []);

    useEffect( effect: () :void  => {
        axios.get( url: "http://localhost:5000/api/users/me",  config: { withCredentials: true }) Promise<AxiosResponse<...>>
            .then((response :AxiosResponse<any> ) :void  => setUser(response.data)) Promise<void>
            .catch((error) :void  => {
                console.error("Помилка отримання користувача:", error);
                navigate("/auth");
            });
    }, deps: [navigate]);
```

Рис. 3.5. Елемент коду сторінки користувача

(розроблено автором)

Використаний функціонал дозволяє здійснювати навігацію на сторінці і відображати історії запитів, а за допомогою налаштованого запиту користувач може миттєво переходити на власну сторінку після аутентифікації, не стикаючись з проблемами доступу (рис. 3.5.). Обробка помилок дозволяє у разі чого висвітити необхідне повідомлення у разі проблеми, але налаштований механізм доволі надійний і має працездатну модель, яка базується на запитах, обробці даних й змінних стану. Також в цьому грає роль компонент стилів UserPage.css у якості доповнення до сторінки користувача, який відповідає за оформлення сторінки, а сами стилю таблиці історії, розміру й оформленню кнопок, а також їх розташування і розмірам, створюючи комунікативний й задовільний інтерфейс для взаємодії користувача з власним кабінетом web-застосунку. При цьому доступне очищення списку історії за допомогою запиту через axios до бази даних, що повністю очищує список історії для конкретного користувача web-застосунку. Ще створені декілька функціональних можливостей для трьох інших кнопок, які дозволяють завантажувати список

історії запитів у файлах різного формату для зручності користувача й подальшого використання.

Компонент погодної мапи під назвою WeatherMap.js являє собою метеорологічну карту з можливістю вибору різних шарів для відображення стану погоди на карті або за допомогою позначення маркера. Використання бібліотеки Leaflet дозволяє відображати інтерактивну мапу с її властивостями, а також перемикати шари та дозволяти відстежувати погодні явища на розробленій мапі. Для користувача доступні 5 різних шарів – опади, вітер, тиск, відносна вологість й температура, перемикання яких реалізовано за допомогою наявного функціонала RadioControlButton, що дозволяє вибирати один шар у якості активного для перегляду стану погоди в потрібній місцевості на мапі. Вибір одного шара дозволяє також отримати текстову інформацію в якості параметрів, таких як визначена позиція при виборі точці на вибраній місцевості за допомогою маркера разом з координатами, показнику самого активного шару й короткого опису по розподіленим категоріям оцінки, від меншого до найбільшого. Це дозволяє отримувати необхідну інформацію в повному обсязі, а ще мати можливість використання додаткового вікна над маркером, якщо потрібно переглянути метеорологічні показники. У компоненті також реалізовано обробку помилок на рівні завантаження та роботи маркера, щоб у разі проблем показати повідомлення про наявні помилки. Компонент стилів під назвою WeatherMapStyles.js створює мінімалістичне, але достатнє оформлення інтерфейсу метеорологічної мапи для перегляду та використання інтерактивної карти користувачем web-застосунку, щоб отримати необхідну інформацію за короткий час без проблем та візуального перенавантаження. Це дозволяє реалізувати повноцінну метеорологічну інтерактивну мапу для перегляду стану погоди, температури та інших наявних показників у багатьох місцевостях планети, за наявністю доступних даних від постачальника у вигляді сервісу OpenWeatherMap.

Серверна частина представлена компонентами для забезпечення запуску та роботи самого сервера та з’єднання з наявною базою даних для збереження та

отримання необхідних даних за допомогою запита. Реалізовані елементи серверної частини забезпечують виконання створених запитів, налаштовану правильну маршрутизацію та безпечність обміну даних між застосунком й базою даних, використовуючи спеціалізовані токени JWT та хешування паролів для захисту конфіденційності користувачів web-застосунку. Компонент server.js відповідає за налаштування та запуск сервера, даючи можливість обміну інформацією та коректною маршрутизацією. Також реалізований важливий компонент database.js, який важливо потрібен для створення з'єднання з базою даних і надання такої можливості усім іншим компонентам, забезпечуючи повну функціональність та операції обміну й збереження інформації у базі даних. Для налаштування маршрутизації двох частин web-застосунку та можливості надійного збереження історії запитів використовується компонент historyRoutes.js, який безпосередньо грає важливу роль у механізмі збереження історії запитів користувача під час використання їм web-застосунку, зберігаючи усі запити від наявних джерел до окремої таблиці history в створеній базі даних.

```
const UserModel : {login: UserModel.login, register: ...}  = {
    register: async (username, password, callback) : Promise<void>  => {
        const hashedPassword = await bcrypt.hash(password,  salt: 10);
        const query : string  = "INSERT INTO users (username, password) VALUES (?, ?)";
        db.query(query, [username, hashedPassword], callback);
    },
```

Рис. 3.6. Елемент коду для хешування пароля

(розроблено автором)

Ще один компонент у серверній частині під назвою authRoutes забезпечує правильну маршрутизацію запитів, обробку операцій аутентифікації користувачів та додаткових дій з токенами, забезпечуючи надійний механізм входу до власного акаунта. UserModel.js використовується для валідації введених даних, хешування створених користувачами паролів (рис. 3.6.), забезпечуючи високий рівень безпеки за допомогою перевірки даних автентифікації при вході користувачів у систему web-застосунку, обробляючи

введені дані та вже збережену інформацію у базі даних. Використаний фреймворк Express дозволяє забезпечувати з'єднання між компонентами застосунку й налаштованої маршрутизації, забезпечуючи роботу реалізованих алгоритмів й коректної взаємодії між компонентами web-застосунку й базою даних, шляхом передачі запитів до відповідних модулів й надання доступу для обміну даних між ними. Взаємопов'язаний компонент userRoutes.js за налаштування запитів для операцій з користувачами, перевіряє наявні токени й оновлює інформацію про активних аутентифікованих користувачів у системі web-застосунку, що підтримує працездатність й коректність роботи застосунку й власних кабінетів користувачів. Це дозволяє забезпечувати роботу компонентів системи авторизації та реєстрації, дозволяючи зареєстрованим користувачам входити до створених власних кабінетів у застосунку. Реалізована серверна частина web-застосунку дозволяє забезпечувати стабільне та безпечне використання застосунку й створення власних акаунтів для збереження історії запитів, а розроблений додатковий функціонал дозволяє отримувати більш можливих даних та функцій по їх обробці або використанню, забезпечуючи доволі функціональну систему усіма необхідними можливостями при виконанні визначених вимог. Усі компоненти реалізовані з розподілом відповідних функцій окремих алгоритмів, дозволяючи підвищити відмовостійкість системи та безпеку у разі проблем з окремим елементом серверної частини або несправністю алгоритмів, що дозволяє за допомоги розробленого механізму виявлення та відстежування помилок знаходити та виправляти помилки, що виникають під час використання web-застосунку. Разом усі компоненти створюють цілісну взаємопов'язану архітектуру компонентів для працездатності web-застосунку, надаючи можливість користувачу отримувати метеорологічні прогнози та графіки змін показників погоди, маючи можливість користуватися додатковим функціоналом створеного застосунку у вигляді створення власного акаунта та перегляду історії запитів, забезпечуючи усі визначені вимоги та можливості системи.

3.3 Тестування веб-застосунку і перевірка роботи застосунку для агрегації і аналізу метеорологічних прогнозів

Останні кроки перед закінченням розробки застосунку та завершенням роботи є тестування розробленого програмного продукту та перевірка роботи функцій й відповідності визначеним вимогам розробки. Необхідність забезпечення для користувачів доступності усіх розроблених функцій й можливостей потребує повної перевірки та тестування усіх наявних для користувача алгоритмів, щоб забезпечити безперебійну і коректну роботу усіх елементів, починаючи з інтерфейсу і закінчуючи операціями збереження даних та збереження історії запитів. Ще розроблений web-застосунок повинен відповідати визначеним функціональним і нефункціональним вимогам, включаючи простоту і зручність інтерфейсу, швидкого отримання прогнозів погоди у вигляді структурованих даних та графіків показників, а також доступну та надійну авторизацію та реєстрацію у системі застосунку для створення власного акаунта та збереження історії запитів для подальшого її перегляду чи завантаження у зручному форматі. Крім зазначених дій, потрібно перевірити правильність відображень повідомлень про помилки, які доволі корисні як для розробника, так і для користувача у процесі тестування та взаємодія з розробленим web-застосунком, щоб при необхідності розуміти корінь проблеми й при можливості за допомогою доступних можливостей виправити їх. Тестування web-застосунку та користувацької частини повинно забезпечити кросбраузерність, яка передбачає сумісність з популярними веб-браузерами, такими як Microsoft Edge, Google Chrome та Mozilla Firefox, при використанні яких має бути забезпечена повноцінна функціональність додатка і робота алгоритмів та взаємодією с базою даних, не отримуючи проблем або неправильного відображення отриманих даних зі сторонніх сервісів. Також повинно перевірити адаптивність дизайну, який повинен автоматично підстроюватися під розширення екрана, що має забезпечити коректну роботу додатка на абсолютно різних пристроїв потенційних користувачів – від

комп’ютера й планшета до смартфона різних розмірів.

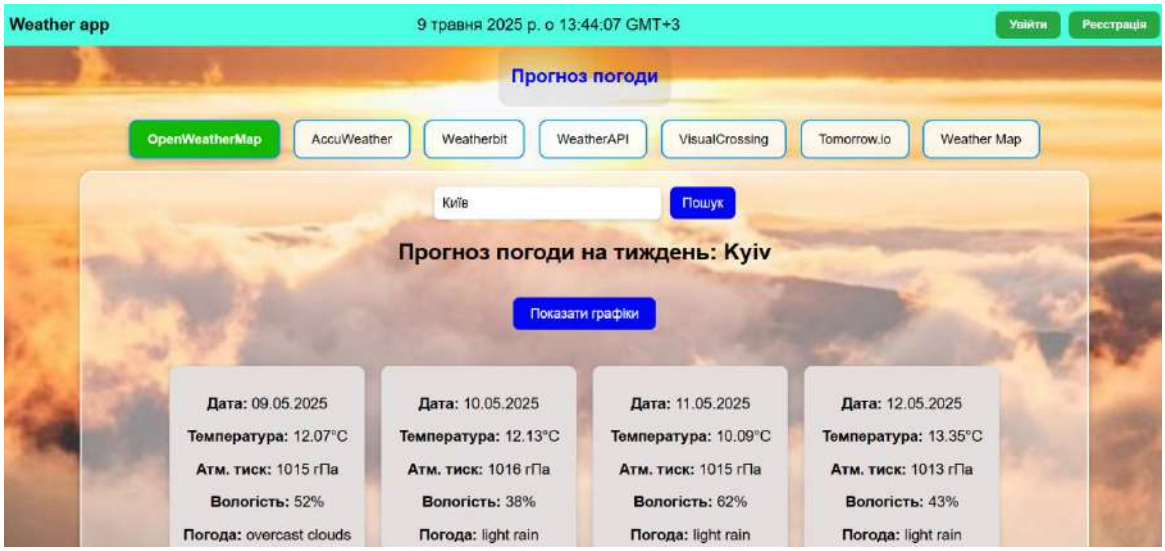


Рис. 3.6. Отримання прогнозу від OpenWeatherMap

(розроблено автором)

Тестування функціоналу web-застосунку для агрегації і аналізу метеорологічних прогнозів починається з основних й загальних функцій, таких як отримання прогнозу погоди з вибраного користувачем джерела (рис. 3.6.). Починаючи з першого джерела як визначеного, вводимо потрібне місто для пошуку прогнозу і натискаючи необхідну для відправки запиту кнопку, отримуємо структурований по днях результат для вибраного місця, з усіма доступними показниками й можливість переглянути по деяких з них автоматично створені графіки. Усі наявні на головній сторінці web-застосунку вибрані метеорологічні джерела повністю надають можливості отримання метеорологічних прогнозів у повному обсязі, маючи додаткову функцію у вигляді формування графіків погодних показників на увесь час наданого прогнозу. Також була перевірена можливість повідомлень про помилки, коли при неправильному вводі або пустому полі у пошуку реалізовані повідомлення повідомляють користувача web-застосунку про необхідність коректного вводу або інших помилок, які сталися вже через проблеми порушення з’єднання з сервером чи джерелами, тому створена система уловлювання помилок дозволяє

висвітлювати проблеми під час користування web-застосунком. Налаштований перекладач також має обробку помилок, але при тестуванні повністю перекладає введені назви на англійську мову автоматично, хоча й має деякі труднощі з перекладом довгих або складних назв.

Також важливою умовою для роботи web-застосунку є реалізована кросбраузерність, яка має полягати в коректній та повноцінній роботі усієї системи та розробленого функціоналу застосунку у найбільш поширених браузерах, таких як Google Chrome, Microsoft Edge та Mozilla Firefox, що є найбільш популярними серед користувачів на платформі операційної системи Windows.

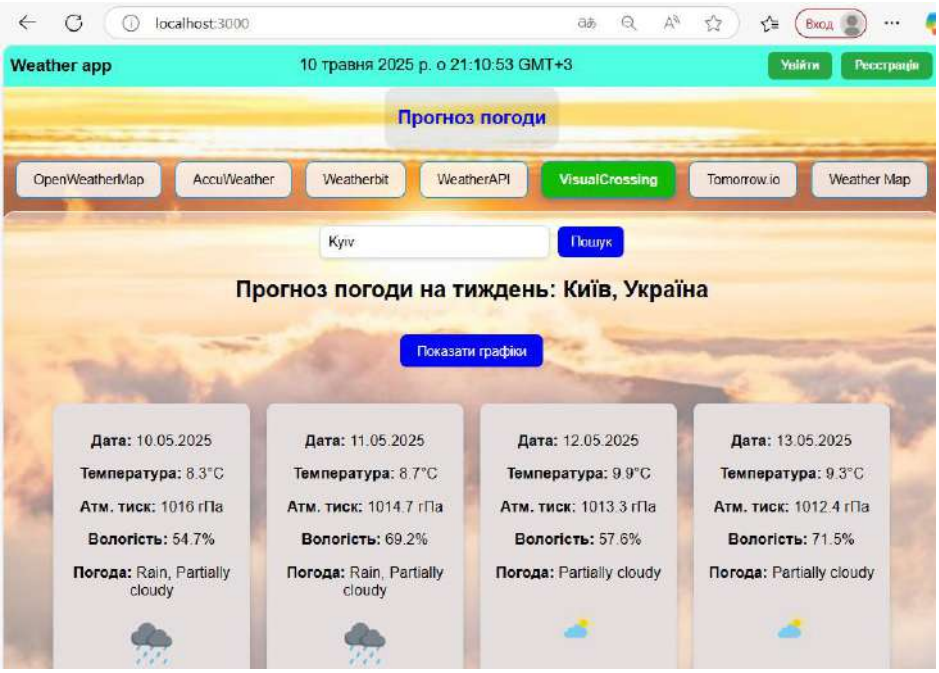


Рис. 3.7. Робота застосунку у браузері Microsoft Edge

(розроблено автором)

Оскільки при розробці й поточному тестуванні на різних етапах використовувався Google Chrome для тестування та визначення помилок, перевіряємо web-застосунок на працездатність в Microsoft Edge та Mozilla Firefox (рис. 3.7.), отримуючи повну працездатність та коректність відображення елементів інтерфейсу під час тестування сторінок та запитів прогнозу погоди у

вибраних для тестування браузерів.

Враховуючи наявність системи авторизації та створення власного кабінету у ролі акаунта, який дозволяє отримувати доступ до функціоналу зберігання історії запитів та її перегляду, що дозволяє дізнаватися про найбільш часті або недавні запити та при необхідності завантажувати їх в зручному форматі. Необхідність перевірки функціоналу акаунта та усіх кнопок для правильності роботи алгоритмів та маршрутизації визначається важливістю можливості та необхідності забезпечення швидкого та коректного доступу користувача до створеного акаунта або форми реєстрації у web-застосунку.

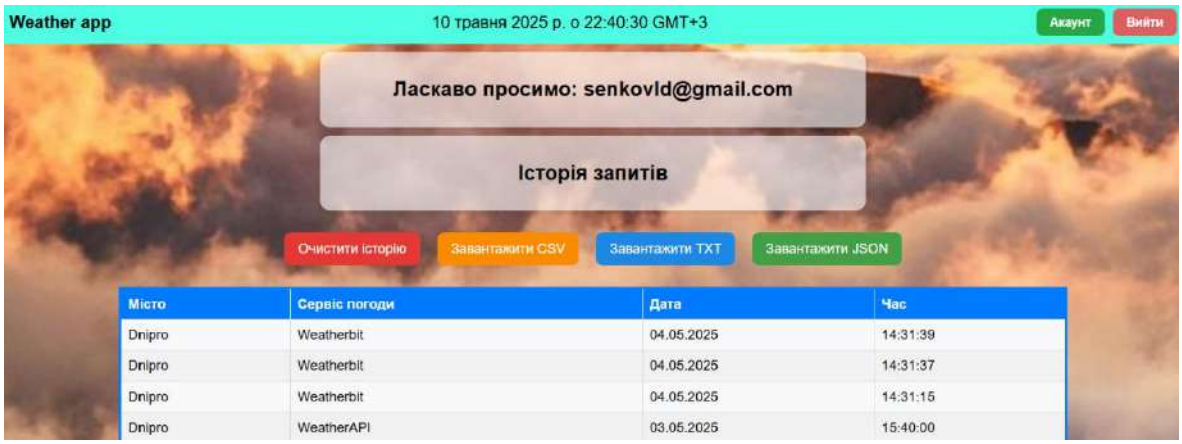


Рис. 3.8. Кабінет користувача зі списком історії запитів

(розроблено автором)

Як можна бачити, система входу до акаунта у вигляді авторизації працює коректно й дозволяє користувачу увійти до власного створеного кабінету у системі web-застосунку, де відображається історія запитів й додатковий функціонал у вигляді чотирьох кнопок, які мають різне призначення для визначених дій (рис. 3.8.). Правильне відображення елементів інтерфейсу і функціоналу кнопок завдяки налаштованим компонентам дозволяє швидко та комфортно користуватися інтерфейсом сторінок, з можливістю переходити між головною сторінкою з метеорологічними прогнозами на власну сторінку кабінету з історією запитів. Це є одною з важливих задач та визначених вимог при реалізації web-застосунку, який є багатофункціональним та має можливість

для вдосконалення або розширення існуючого функціоналу в більшу сторону при можливості та нових потребах користувачів, що може забезпечувати подальший розвиток застосунку і його вдосконалення.

Рис. 3.9. Перевірка адаптивного дизайну

(розроблено автором)

Ще одною необхідністю та важливим моментом при тестуванні є правильна робота застосунку при використанні на мобільних пристроях, так званого адаптивного дизайну, який дозволяє переглядати застосунок з таким же комфортом та функціоналом, як і на персональних комп’ютерах. Як можна бачити на рисунку 3.9, тестування web-застосунку на емуляції мобільного

пристрою типу смартфона Iphone 12 з розширенням смартфона пройшло успішно, інтерфейс повністю відображається без помилок, а кнопки та елементи зберегли розроблений функціонал, що вказує на повну працездатність web-застосунку на мобільних пристроях типу смартфона або планшета. Ця можливість не тільки дозволяє уникати помилок інтерфейсу при зміні розширення екрана, але й створювати гарний та комфортний користувацький інтерфейс для використання на мобільних пристроях.

Враховуючи повне тестування web-застосунку й усього доступного функціонала з інтерфейсом та адаптивним дизайном, можна зробити висновки, що розроблений застосунок демонструє повну працездатність, коректність роботи створених алгоритмів та активну обробку помилок, яка дозволяє отримувати повідомлення у разі незвичайної поведінки системи або недоступності деякого функціоналу в момент використання. Це дозволяє визначити, що web-застосунок відповідає усім необхідним вимогам та придатний для використання звичайними користувачами, які при наявності пристрою з підключеним інтернетом та браузером можуть повноцінно користуватися розробленим web-застосунком для агрегації й аналізу метеорологічних прогнозів, який має багато доступних функцій та створених можливостей.

Висновки до розділу 3

В третьому розділі було розглянуто інструменти та додатки реалізації системи й виконання вимог розробки, зроблено їх огляд та порівняння можливостей, аналіз середи розробки у вигляді IDE Webstorm від компанії JetBrains, його переваги та додаткові інструменти для реалізації розробки web-застосунку, що дозволяє створити структуру та необхідні компоненти системи згідно з визначеними вимогами реалізації програмного продукту. Використання мови програмування javascript разом з React та Node.js дозволяє швидко та повноцінно реалізовувати потрібні алгоритми обробки даних та механізми

взаємодії, створюючи повноцінну систему із користувацької та серверної частини. Фреймворк Express та бібліотека Axios дозволяє створювати маршрутизації та налаштування взаємодії зі сторонніми API, що доволі важливо при розробці серверної частини застосунку. Розроблена реляційна база даних MySQL дозволяє надійно та ефективно зберігати необхідні дані, що можуть бути оброблені, отримані або видаленні при більшості доступних запитів, підвищуючи зручність та безпечність обміну інформації.

При повноцінній розробці функціоналу web-застосунку агрегації метеорологічних прогнозів та надання прогнозу погоди було створено користувацьку і серверну частину, налаштована надійна взаємодія компонентів через маршрутизацію та створену логіку запитів, а також необхідні запити до реляційної бази даних та до потрібних API метеорологічних сервісів, які надають необхідні дані для формування прогнозу у web-застосунку. Було реалізовано головну сторінку з механізмом обробки, модулі окремих джерел для обробки погодних даних та компоненти для сторінки входу до акаунта й відображення кабінету з історією запитів. Для роботи серверної частини були реалізовані компоненти для запуску сервера, налаштування маршрутизації між створеними модулями, можливістю авторизації та реєстрації в акаунті й збереження історії запитів у базі даних. Це дозволило реалізувати повноцінний працездатний web-застосунок, який відповідає раніше визначеним вимогам та потребам для користувачів, маючи достатній функціонал і можливості користування.

У останньому етапі у ході тестування web-застосунку було перевірено роботу загального функціоналу, механізмів обробки помилок при проблемах та маршрутизацію між сторінками застосунку. Перевірка показала повну працездатність web-застосунку, отримання прогнозів погоди для користувачів, коректну авторизацію та використання акаунта для перегляду історії й функцій завантаження у різних файлових форматах. Також було проведено тестування застосунку на емуляції мобільних пристроях і роботу адаптивного дизайну, яка передбачає комфортну і коректну роботу сторінок й алгоритмів на мобільних пристроях різних типів, таких як смартфон або планшет.

ВИСНОВКИ

Під час проведеного дослідження було проаналізовано процеси отримання метеорологічних даних за допомогою запитів до сторонніх API, способи формування прогнозів та обробки даних, можливості агрегації та візуалізації отриманих даних для надання більшої функціональності web-застосунку для користувачів. Використання необхідних інструментів дозволяє реалізувати потрібні можливості у вигляді запитів, які адресовані до визначених сторонніх API, що дозволяє отримувати велику кількість погодних даних з різних джерел. Застосовані технології допомагають створювати надійні механізми взаємозв'язку між компонентами та налаштованої маршрутизації, що створює взаємопов'язану структуру застосунку.

У процесі виконання кваліфікаційної роботи були визначені необхідні функціональні вимоги, розроблена архітектура застосунку, потреби користувачів та необхідний функціонал для використання web-застосунку користувачами. Розроблений web-застосунок базується на важливих технологіях Node.js, React, та інших доповнень у вигляді Axios та Express з використанням реляційної бази даних MySQL. Основною функцією застосунку є отримання погодних даних та формування прогнозів за запитами користувачів, які мають надавати достатню кількість інформації на визначений доступний час. Реалізована інтерактивна карта дозволяє користувачам застосунку отримувати дані про стан погоди з вибраної місцевості за допомогою маркера, який можна рухати. Також на карті доступні різні шари погодних показників, такі як температура, тиск та опади. Доступна візуальна частина у кожному джерелі дозволяє перемикати режим перегляду отриманого прогнозу погоди на автоматично створені графіки загальних погодних показників, які показують коливання та іншу необхідну інформацію користувачам web-застосунку, що дозволяє більш детально та якісно отримати уявлення про стан погоди та умов у визначеному місці прогнозу.

При реалізації системи були розглянути доступні методи та необхідні

рішення для створення багатофункціонального web-застосунку, який має систему реєстрації та можливість перегляду історії запитів, а також її завантаження у різних форматах. Створено клієнт-серверну модель на основі компонентної архітектури програмного продукту з можливостями покращення та налаштування надійної маршрутизації зі створеними сторінками для швидкого й комфортного користування. Це дозволяє не тільки отримувати поширені прогнози з більшості джерел, але й відстежувати свою активність у системі та користування web-застосунком. Розроблений застосунок покращує швидкість отримання прогнозів та можливість порівняння різних доступних джерел, даючи користувачу можливість вибирати сформовані прогнози або користуватися інтерактивною картою для перегляду стану погоди, отримання актуальних даних і додаткових можливостей у вигляді візуального представлення деяких метеорологічних показників.

При можливостях продовжування досліджень та розширення функціоналу реалізований програмний продукт може бути значно поліпшений за допомогою реалізації більшої кількості інтерактивних та візуальних елементів, інтеграції нових систем і додаткових API, а також використання нових механізмів для збільшення точності та ефективності прогнозів, розробки багатомовного інтерфейсу. Можливе вдосконалення може полягати як у налагодження зв’язку с користувачем у вигляді спливаючих повідомлень, активного статусу у окремому вікні або сповіщень про зміни стану погоди через електронну пошту або інші подібні сервіси. Це дозволить провести аналіз нових методів прогнозування, потенційного використання штучного інтелекту для обробки даних або формування прогнозів, що дає можливості значно покращити існуючий функціонал та поширити можливості web-застосунку для користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Босько В. В., Константинова Л. В., Марченко К. М., Улічев О. С. Web-програмування. Частина 1 (frontend) [Електронний ресурс]. – Кривий Ріг, 2022. – Режим доступу: https://dspace.kntu.kr.ua/server/api/core/bitstreams/2bfb5a3c-54d9-4283-89c1-5e190e8aa151/content.
2. Ries, J. E. A client-server architecture for modern times. Columbia: University of Missouri, 2024. – [Електронний ресурс]. – Режим доступу: https://mospace.umsystem.edu/xmlui/handle/10355/104732
3. Cervantes H., Kazman R. Designing software architectures: a practical approach. – Addison-Wesley Professional, 2024. – [Електронний ресурс]. – Режим доступу: https://books.google.com/books?hl=ru&lr=&id=Qjb_EAAAQBAJ&oi=fnd&pg=PT13&dq=Architectural+approaches+for+web+applications&ots=n7gJpaQybn&sig=kC7uOrON9JA8SMfbNkaMtBGdQvA.
4. Погромська Г. С., Махровська Н. А. Бази даних: проектування та реалізація. – Місто: Миколаїв, 2019. – [Електронний ресурс]. – Режим доступу: https://files.znu.edu.ua/files/Bibliobooks/Inshi78/0058251.pdf
5. Ranjan A., Sinha A., Battewad R. JavaScript for modern web development: building a web application using HTML, CSS, and JavaScript. – BPB Publications, 2020. – [Електронний ресурс]. – Режим доступу: https://books.google.com/books?hl=ru&lr=&id=b2bdDwAAQBAJ&oi=fnd&pg=PT24&dq=developing+web+applications+javascript&ots=6gfz_zDlL_&sig=JUe1UCVAwAncLHOipnzi8fJHh1Y
6. Haverbeke M. Eloquent JavaScript. – 4th ed. – No Starch Press, 2024. – 660 с. – [Електронний ресурс]. – Режим доступу: https://eloquentjavascript.net/Eloquent_JavaScript_small.pdf
7. Chinnathambi K. Learning React: A Hands-On Guide to Building Web Applications Using React and Redux. – Addison-Wesley, 2018. – 304 с.
8. Rappl F. Modern Frontend Development with Node.js: A compendium for modern JavaScript web development within the Node.js ecosystem. – Packt

Publishing Ltd, 2022. – 191 с.

9. Halliday P. How To Use Axios with React. – 2021.
10. Frain B. Responsive Web Design with HTML5 and CSS: Build future-proof responsive websites using the latest HTML5 and CSS techniques. – 4th ed. – Packt Publishing Ltd, 2022. – 498 с. – [Електронний ресурс]. – Режим доступу: https://books.google.com.ua/books/about/Responsive_Web_Design_with_HTML5_and_CSS.html?id=TkyJEAAAQBAJ&redir_esc=y
11. Attardi J. Master the Key Concepts of CSS for Modern Web Development. – Apress, 2020. – 289 с.
12. Rockoff L. The Language of SQL. – Addison-Wesley Professional, 2021. – 272 с. – [Електронний ресурс]. – Режим доступу: https://books.google.com.ua/books/about/The_Language_of_SQL.html?id=0NnPEAAAQBAJ&redir_esc=y
13. Grippa V. M., Kuzmichev S. Learning MySQL. – O'Reilly Media, Inc., 2021. – 632 с. – [Електронний ресурс]. – Режим доступу: https://books.google.com.ua/books/about/Learning_MySQL.html?id=lDhCEAAAQBAJ&redir_esc=y
14. Casciaro M., Mammino L. Node.js Design Patterns: Master Best Practices to Build Modular and Scalable Server-Side Web Applications. – 2nd ed. – Packt Publishing, 2016. – 526 с.
15. Mardan A. Pro Express.js: Master Express.js: The Node.js Framework for Your Web Development. – Apress, 2014. – 372 с. – [Електронний ресурс]. – Режим доступу: https://link.springer.com/book/10.1007/978-1-4842-0037-7
16. Párraga R. J. C. та ін. Desarrollo de Aplicaciones Web Utilizando Vue.js, Axios y PHP: Web Application Development Using Vue.js, Axios and PHP // Revista Científica Multidisciplinar G-nerando. – 2023. – Т. 4. – [Електронний ресурс]. – Режим доступу: https://revista.gnerando.org/revista/index.php/RCMG/article/view/94
17. Hahn E. Express in Action: Writing, Building, and Testing Node.js Applications. – Manning Publications, 2016. – 256 с.

18. Nichter D. Efficient MySQL Performance. – O'Reilly Media, Inc., 2021. – [Електронний ресурс]. – Режим доступу: https://books.google.com/books?hl=ru&lr=&id=CzZTEAAAQBAJ&oi=fnd&pg=PR2&dq=MySQL&ots=Eh2E3rUg_v&sig=FKLll__OyOFIVCZJDvRpuEWPHkU
19. Schönig H. J. Mastering PostgreSQL 15: Advanced techniques to build and manage scalable, reliable, and fault-tolerant database applications. – Packt Publishing Ltd, 2023. – [Електронний ресурс]. – Режим доступу:
20. https://books.google.com/books?hl=ru&lr=&id=ZBOrEAAAQBAJ&oi=fnd&pg=PP1&dq=PostgreSQL&ots=r1vl2K9cFJ&sig=GK2qOqvVqnl2SnKGLqtMFXxGPHc
21. Kuhn D., Kyte T. Expert Oracle Database Architecture: Techniques and Solutions for High Performance and Productivity. – Apress, 2022. – 1123 с. – [Електронний ресурс]. – Режим доступу:
https://link.springer.com/book/10.1007/978-1-4842-7499-6
22. Price I., Sanchez-Gonzalez A., Alet F. та ін. Probabilistic weather forecasting with machine learning - Nature. – 2024.
23. Lauret A. The Design of Web APIs. – Manning Publications, 2019. – 400 с. – [Електронний ресурс]. – Режим доступу:
https://www.manning.com/books/the-design-of-web-apis
24. Agarwal A. B., Rajesh R., Arul N. Spatially-resolved hyperlocal weather prediction and anomaly detection using IoT sensor networks and machine learning techniques. – BITS Pilani, K.K. Birla Goa Campus, India, 2023.
25. Chadha S., Ahmad S. K., Sharma D., Singh M., Srivastavaline A. Development of Weather Forecast Application Using API - Proceedings of ICCSC, IEEE, 2024. – [Електронний ресурс]. – Режим доступу:
https://www.researchgate.net/publication/388067445_Development_of_Weather_Forecast_Application_Using_API
26. Meshram S. U. Evolution of modern web services–REST API with its architecture and design - International Journal of Research in Engineering, Science and Management. – 2021.

27. Shukla A. Modern JavaScript frameworks and JavaScript's future as a full-stack programming language - Journal of Artificial Intelligence & Cloud Computing. – 2023.
28. Nugraha A. F., Kabetta H., Buana I. K. S., Hadiprakoso R. B. Performance and security comparison of JSON Web Tokens (JWT) and Platform Agnostic Security Tokens (PASETO) on RESTful APIs - 2023 IEEE International Conference on Cryptography, Informatics, and Cybersecurity (ICoCICs). – IEEE, 2023.
29. Ramadasa I., Liyanage L., Asanka D., Dilanka T. Analysis of the effectiveness of using Google Translations API for NLP of Sinhalese. – University of Kelaniya, 2022.
30. Chug S., Nath S. Recent advances in social weather, Common Alert Protocol and dissemination services through APIs in India Meteorological Department - Mausam. – 2024.
31. Munro M. C. Learn FileMaker Pro 2024: The Comprehensive Guide to Building Custom Databases. – Springer Nature, 2024. – 1024 c.
32. Karlsson O. A Performance comparison Between ASP.NET Core and Express.js for Creating Web APIs. – 2021.
33. Syed B. Beginning Node.js. – Apress, 2014. – 308 c.
34. Mardan A. Express.js Deep API Reference. – Apress, 2014. – 160 c.
35. Walmart Product API Data Extraction: A Comprehensive Guide for Businesses [Електронний ресурс]. – Режим доступу: https://www.retailgators.com/walmart-product-api-data-extraction.php

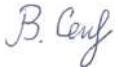
ЗГОДА здобувача вищої освіти
Державного університету економіки і технологій про перевірку
кваліфікаційної роботи на прояви академічного плагіату
та розміщення в Репозитарії Університету

Я, Сенько Владислав Михайлович, підтримую політику Державного університету економіки і технологій з академічної доброчесності і відкритого доступу.

Засвідчую, що кваліфікаційна бакалаврська робота «Розробка web-застосунку для агрегації і аналізу метеорологічних прогнозів» виконана самостійно та не містить академічного плагіату. Я не надавав і не одержува недозволену допомогу під час підготовки цієї роботи. Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Із чинним Положенням про запобігання та виявлення академічного плагіату в роботах здобувачів вищої освіти Державного університету економіки і технологій ознайомлений. Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі порушення норм академічної доброчесності робота не допускається до захисту або оцінюється незадовільно.

Також я поінформований, що відповідно до «Положення про Репозитарій (електронну базу даних) Державного університету економіки і технологій» зазначена робота буде розміщена в Електронному архіві Університету (Репозитарії ДУЕТ). З умовами такого розміщення ознайомлений.

10.06.25  В.М. Сенько