Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)
Кафедра Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Вебдодаток для отримання рекомендацій щодо одягу на основі поточних погодних умов за допомогою штучного інтелекту
(тема)

Виконав:
здобувач четвертого року навчання,
групи ІТШІ-21-5

Артемій Васильченко
(власне ім'я, прізвище)

Спеціальність 122 Комп'ютерні науки

(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Штучний інтелект

(повна назва освітньої програми)
Керівник асистент Дмитро Малєєв
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ШІ _____ Олег ЗОЛОТУХІН
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Штучного інтелекту

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Штучний інтелект
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____» _____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Васильченку Артемію Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Вебдодаток для отримання рекомендацій щодо одягу на основі поточних погодних умов за допомогою штучного інтелекту

затверджена наказом університету від 19 травня 20 25 р. № 378Ст
2. Термін подання студентом роботи до екзаменаційної комісії 17 червня 20 25 р.
3. Вихідні дані до роботи офіційні документації до Firebase, Firestore, Google Cloud Functions, React.js, TypeScript, Mantine UI, RTK Query, OpenAI API, GeoDB Cities API, Vite, PWA Manifest та Service Worker.

4. Перелік питань, що потрібно опрацювати в роботі
1) Аналіз предметної галузі та постановка задачі
2) Проектування системи
3) Програмна реалізація

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	19.05.2025	виконано
2	Визначення проблеми	20.05.2025	виконано
3	Аналіз існуючих аналогів	21.05.2025	виконано
4	Розробка компонентів системи	25.05.2025	виконано
5	Реалізація вебдодатку	30.05.2025	виконано
6	Оформлення пояснювальної записки	02.06.2025	виконано
7	Захист	17.06.2025	виконано

Дата видачі завдання 19 травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ асистент Дмитро Малєєв
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 101 с., 23 рис., 1 дод., 23 джерела.

ПЕРСОНАЛІЗАЦІЯ, ПОГОДА, РЕКОМЕНДАЦІЇ, СЕРВІС, API, FIRESTORE, GPT, OPENAI, PWA, REACT.

Об’єктом дослідження є сучасні вебдодатки, що поєднують функціональність прогнозування погоди та персоналізованих рекомендацій для користувача.

Предметом дослідження виступають технології реалізації персоналізованих рекомендацій щодо вибору одягу на основі метеоданих, включаючи роботу з відкритими API та використання генеративних моделей штучного інтелекту.

Метою кваліфікаційної роботи є розробка та впровадження вебдодатку з підтримкою PWA-функціональності, який формує поради щодо одягу, адаптовані до локації користувача, погодних показників та типу активності.

Методи дослідження включають аналіз технічних рішень аналогічних систем, використання бібліотек React і Mantine, організацію архітектури клієнт-сервер за допомогою RTK Query та реалізацію серверної логіки на Google Cloud Functions з інтеграцією зовнішніх API.

Результатом роботи став повнофункціональний додаток, що поєднує сучасні хмарні сервіси, базу даних Firestore та модель GPT для генерації текстових порад, які можуть бути використані у практиці розробки погодних додатків нового покоління.

ABSTRACT

Bachelor’s thesis contains: 101 pp., 23 fig., 1 ann., 23 references.

API, FIRESTORE, GPT, OPENAI, PERSONALIZATION, PWA, REACT, RECOMMENDATIONS, SERVICE, WEATHER.

The object of the study is modern web applications that combine weather forecasting functionality with personalized user recommendations.

The subject of the study involves technologies for implementing personalized clothing recommendations based on meteorological data, including the use of open APIs and generative artificial intelligence models.

The aim of the qualification work is to develop and implement a web application with PWA support that provides clothing suggestions tailored to the user's location, weather conditions, and activity type.

The research methods include analysis of technical solutions in similar systems, usage of React and Mantine libraries, structuring client-server architecture via RTK Query, and implementing server-side logic with Google Cloud Functions integrated with external APIs.

As a result, a fully functional application was developed that integrates modern cloud services, the Firestore database, and the GPT model for generating textual suggestions, which can be applied in the development of next-generation weather applications.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AI – Artificial Intelligence – штучний інтелект;

API – Application Programming Interface – прикладний програмний інтерфейс;

CSS – Cascading Style Sheets – каскадні таблиці стилів;

GPT – Generative Pre-trained Transformer – генеративний попередньо навчений трансформер;

HTTP – Hypertext Transfer Protocol – протокол передачі гіпертексту;

JSON – JavaScript Object Notation – текстовий формат обміну даними;

PWA – Progressive Web Application – прогресивний вебдодаток;

REST – Representational State Transfer – передача репрезентативного стану;

SDK – Software Development Kit – комплект засобів розробки;

SQL – Structured Query Language – мова структурованих запитів;

UI – User Interface – інтерфейс користувача;

UX – User Experience – користувацький досвід.

ВСТУП

Сучасне інформаційне середовище характеризується швидким зростанням кількості вебсервісів, орієнтованих на щоденну підтримку користувача в прийнятті повсякденних рішень. Однією з найбільш поширених категорій таких сервісів є погодні додатки, що дозволяють мільйонам людей по всьому світу отримувати актуальні метеорологічні дані в реальному часі. Водночас більшість традиційних погодних сервісів зосереджені виключно на інформуванні – вони надають дані про температуру, вологість, вітер, опади та інші показники, однак не забезпечують інтерпретації або рекомендацій, які були б безпосередньо корисні у конкретному контексті користувача. Саме ця особливість формує проблему, яка й стала предметом дослідження в межах цієї кваліфікаційної роботи.

Проаналізувавши наявні рішення, можна зробити висновок, що на ринку існує потреба в системах, які здатні не лише повідомляти про погодні умови, а й пропонувати на їх основі персоналізовані поради, зокрема щодо вибору одягу. Людина щодня приймає рішення, що вдягнути перед виходом з дому, і дуже часто це рішення базується на інтуїтивному тлумаченні прогнозу. Особливо складними є ситуації у міжсезоння, за різких змін температури або при високій вологості, коли реальні відчуття значно відрізняються від сухих цифр у погодному додатку. У цих умовах стає очевидною необхідність у сервісі, який міг би не лише повідомити, що температура становить +7°C, а й дати практичну пораду: наприклад, вдягнути вітровку, шарф і взяти з собою парасольку.

Розвиток мовних моделей та відкритий доступ до потужних інструментів генерації тексту, таких як OpenAI API, створюють умови для принципово нового підходу до побудови погодних сервісів. Завдяки інтеграції метеоданих з мовною моделлю можна автоматизувати процес формування рекомендацій щодо одягу, враховуючи не лише погодні умови,

а й тип активності, час доби, локацію користувача та інші чинники. Такий підхід дозволяє реалізувати адаптивний інтерфейс, у якому користувач вводить коротке запитання на кшталт «що вдягти сьогодні на прогулянку в Києві?» і отримує повну, логічну та граматично правильну відповідь. Це значно підвищує практичну цінність сервісу та наближає його до категорії цифрових асистентів.

Актуальність цієї роботи визначається як об’єктивною потребою користувачів у зручному, швидкому та розумному інструменті для отримання погодних рекомендацій, так і відсутністю подібних рішень у вебсередовищі з використанням сучасних AI-платформ. Більшість існуючих сервісів або не надають таких порад взагалі, або роблять це у примітивній формі. Крім того, потреба в персоналізації цифрових інструментів зростає паралельно з підвищенням очікувань користувачів щодо зручності та адаптивності. Технології вже дозволяють реалізовувати подібні рішення, і саме тому створення такого вебдодатку є логічним кроком у розвитку інтерактивних погодних систем.

Метою цієї кваліфікаційної роботи є створення вебдодатку, який дозволяє користувачам отримувати персоналізовані текстові рекомендації щодо вибору одягу, ґрунтуючись на актуальній погоді, введеній локації та контексті використання. Для цього буде застосовано погодні API для отримання метеорологічних даних, а також мовну модель для генерації змістовних відповідей у природній мові. У майбутньому таку систему можна адаптувати для різних платформ, доповнити мобільною версією або інтегрувати з голосовими асистентами. Сфера застосування охоплює як індивідуальних користувачів, так і можливе використання у великих містах, освітніх закладах або туристичних інформаційних центрах. Результати цієї роботи продемонструють можливість поєднання класичних погодних сервісів з адаптивною логікою штучного інтелекту, що відповідає сучасним технологічним тенденціям.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Роль сучасних технологій у розвитку погодних сервісів

У сучасному цифровому середовищі погодні сервіси стали невід’ємною частиною повсякденного життя мільйонів користувачів. Вони інтегруються у браузери, віджети, смарт-пристрої та вебдодатки, забезпечуючи зручний і швидкий доступ до актуальної метеорологічної інформації. З розвитком технологій та поширенням доступу до Інтернету зростає попит на більш гнучкі та адаптивні форми подачі погодних даних, орієнтовані на конкретні потреби користувача.

Разом із цим еволюціонували і способи взаємодії користувачів з погодною інформацією. Якщо раніше достатньо було знати температуру та опади, сьогодні спостерігається попит на системи, що допомагають приймати рішення на основі отриманих даних. Особливо це актуально у контексті вибору одягу, планування активностей на відкритому повітрі, подорожей та занять спортом. У зв’язку з цим на перший план виходить не лише точність прогнозу, а й рівень інтерпретації, доступний користувачу.

Застосування штучного інтелекту дозволяє перейти від статичного інформування до персоналізованих порад. Інтелектуальні системи здатні враховувати більше факторів, зокрема тип активності, індивідуальні вподобання, геолокацію та інші змінні. Саме поява таких технологічних можливостей створює передумови для розробки інноваційних рішень, що трансформують традиційне уявлення про погодні сервіси.

1.1.1 Розвиток вебдодатків для відображення погодних умов

Еволюція вебдодатків для відображення погодних умов відбувалася поетапно, від простих текстових сторінок до комплексних інтерактивних платформ. На ранніх етапах розвитку користувачі мали доступ до базової

інформації у вигляді тексту або статичних піктограм, що відображали загальний прогноз. Ці сервіси мали мінімальну гнучкість і не враховували особливості взаємодії користувача з погодними даними.

Далі можна побачити узагальнений огляд (рисунок 1.1) етапів розвитку погодних вебдодатків, від базових текстових сайтів до інтеграції голосових асистентів і персоналізованих рекомендацій на основі штучного інтелекту. Сучасні тренди демонструють рух у бік глибшої взаємодії з користувачем, адаптації до його контексту і поведінки, що закладає основу для інтелектуальних сервісів нового покоління.

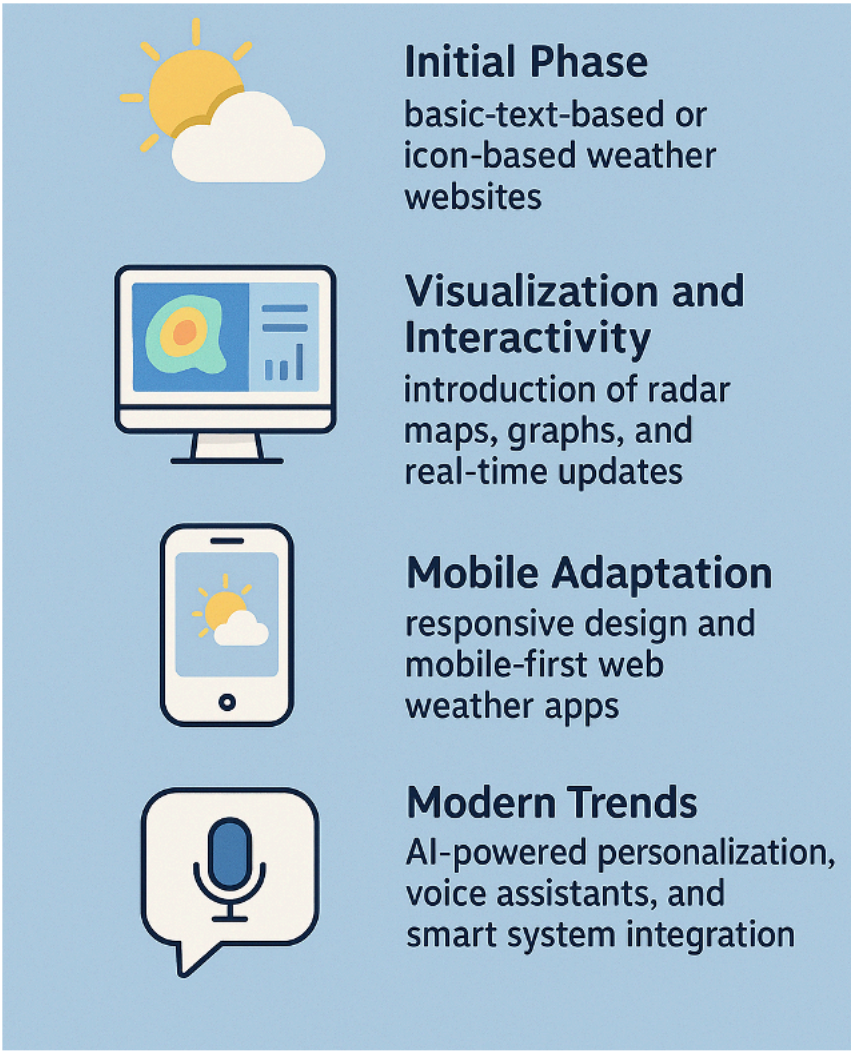

Рисунок 1.1 – Епохи розвитку погодних додатків

Наступним кроком стало впровадження засобів візуалізації та інтерактивності. Це включало відображення радарних мап, графіків, а також можливість отримання оновлень у режимі реального часу. Такі нововведення значно підвищили якість сприйняття даних і сприяли ширшому використанню вебдодатків у щоденній практиці. Візуалізація зробила погодні сервіси більш доступними і зрозумілими для широкої аудиторії, незалежно від технічної підготовки користувача.

Зі зростанням популярності мобільних пристроїв з'явилася потреба в адаптації вебдодатків під мобільні екрани. Реалізація адаптивної верстки та створення мобіле-орієнтованих погодних інтерфейсів дозволила користувачам отримувати прогноз у зручному форматі без прив'язки до стаціонарного браузера. Це також дало змогу впровадити нові сценарії використання, зокрема перегляд погоди «на ходу» або під час подорожей.

1.1.2 Функціональні можливості сучасних погодних сервісів

Сучасні погодні сервіси пропонують широкий спектр функціональних можливостей, які значно перевищують рівень базового інформування. Однією з основних складових є відображення поточних погодних умов, таких як температура, вологість, вітер, атмосферний тиск та опади. Ці дані оновлюються в режимі реального часу й дозволяють користувачу оперативно оцінити ситуацію у своєму регіоні.

Прогноз погоди є другою ключовою функцією, яка забезпечує користувача інформацією на декілька днів наперед. Більшість сервісів пропонують як денний, так і погодинний прогноз, що дозволяє краще планувати повсякденну активність. Особливо важливою є точність прогнозу в контексті короткострокових змін, наприклад, різкої зміни температури або ймовірності опадів у конкретний час доби.

Далі можна побачити узагальнений перелік основних функцій, що реалізуються в більшості популярних погодних додатків: поточна погода,

прогноз, погодні мапи та система сповіщень (рисунок 1.2). Сповіщення, як правило, використовуються для попередження про небезпечні погодні явища або для нагадування про очікувані зміни, що дозволяє користувачу оперативно реагувати на нові умови.

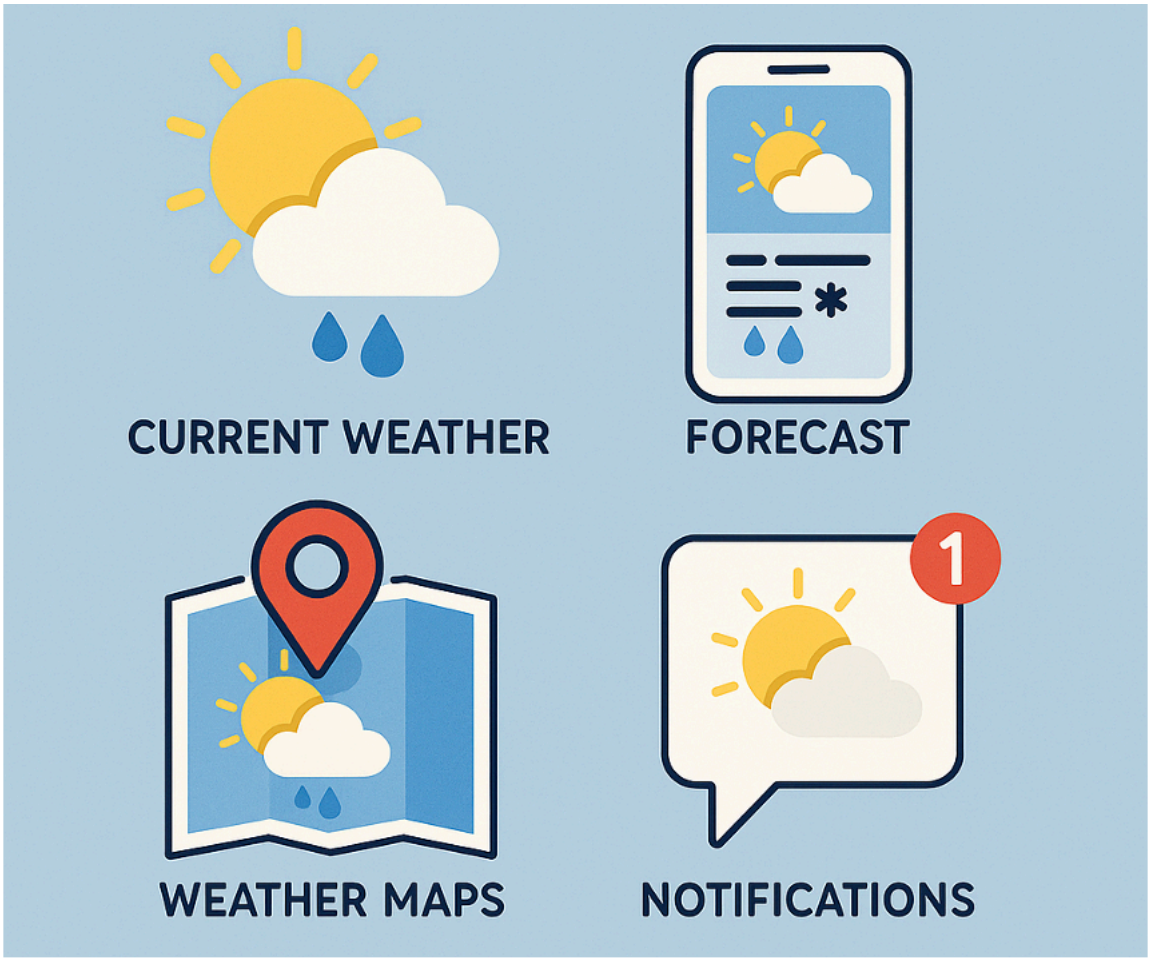

Рисунок 1.2 – Основні можливості погодних сервісів

Візуалізація через погодні мапи є ще одним критично важливим елементом сучасних рішень. За допомогою інтерактивних мап користувачі можуть оцінити атмосферну ситуацію не лише в конкретній точці, а й у регіональному або глобальному масштабі. Це особливо корисно для подорожей, пересування містом або аналізу погодних фронтів і зон опадів, щоб знати погоду.

1.1.3 Вплив розвитку штучного інтелекту на індустрію

Стрімкий розвиток штучного інтелекту вплинув на багато сфер, зокрема і на галузь погодних сервісів. Раніше основна увага приділялася точності прогнозування на основі метеомоделей, тоді як сьогодні інтерес зміщується до інтерпретації даних та їх адаптації під індивідуальні запити користувачів. ШІ дозволяє переходити від загального прогнозу до гнучких і контекстно-залежних рішень.

Одним із ключових напрямів застосування штучного інтелекту є побудова персоналізованих моделей взаємодії. На основі попередніх дій, геолокації, стилю життя та інтересів користувача система може не лише надавати точні погодні дані, а й формувати рекомендації щодо одягу, планування маршруту чи попереджень про несприятливі умови. Такий підхід змінює логіку побудови погодного інтерфейсу з реактивної на проактивну.

Також набувають поширення голосові асистенти та чатботи, які використовують штучний інтелект для діалогу з користувачем. Вони дають змогу отримати рекомендації у природній мові, наприклад: «що краще вдягти для ранкової прогулянки за 12 градусів і вітру 5 м/с?». Такі рішення забезпечують не лише зручність, а й підвищення довіри до сервісу завдяки інтуїтивній взаємодії.

1.1.4 Мотивація до створення нового інструменту

Аналіз сучасних погодних сервісів виявляє явний розрив між доступною інформацією та реальними потребами користувача. Більшість систем надають точні метеорологічні дані, але залишають на користувача обов’язок самостійно їх інтерпретувати. У контексті швидкого темпу життя та великої кількості щоденних рішень це створює додаткове когнітивне навантаження, яке могло б бути автоматизовано.

З іншого боку, поява потужних мовних моделей та відкритих API створила передумови для нового типу погодного сервісу, який не лише повідомляє, але й радить. Використання штучного інтелекту дає змогу перетворити сухі числові показники на змістовні поради, що враховують контекст активності, стиль життя, звички та географічні особливості. Це дозволяє підвищити практичну цінність сервісу без збільшення обсягу вхідних даних.

Таким чином, мотивацією для створення нового інструменту стала необхідність закрити прогалину між прогнозом і дією. Користувачі прагнуть отримувати не просто інформацію про погоду, а конкретні підказки у відповідь на повсякденне запитання: «що мені робити з цією інформацією?». Саме тому розробка системи, яка перетворює прогноз на персоналізовану рекомендацію, є актуальним і логічним кроком у розвитку інтелектуальних погодних сервісів.

1.2 Ринок додатків для вибору одягу на основі погодних умов

Погодні додатки стабільно утримують високі позиції серед найпопулярніших цифрових сервісів у світі. Щодня мільйони користувачів звертаються до них із метою дізнатись про погодні умови у своєму місті чи будь-якому іншому регіоні. У зв’язку з цим сформувався стійкий і різноманітний ринок, представлений як глобальними гравцями, так і нішевими рішеннями, орієнтованими на певні сценарії використання.

Значна частина таких додатків обмежується передачею фактичних даних про погоду, однак деякі з них пропонують додаткову функціональність, зокрема прості рекомендації. Усе ж більшість доступних інструментів не виходять за межі шаблонного підходу до подачі інформації, часто ігноруючи потребу в адаптації прогнозу під конкретну ситуацію користувача. Це створює простір для нових рішень, здатних підвищити цінність погодного сервісу завдяки персоналізації.

У межах даного підрозділу буде проаналізовано існуючі категорії погодних додатків, включно з тими, що поєднують прогнозування з порадами щодо вибору одягу. Також буде розглянуто функціональні особливості конкурентних систем, порівняно їх сильні та слабкі сторони, а також визначено, які проблеми залишаються невирішеними. Це дозволить сформувати обґрунтовані висновки щодо доцільності впровадження нового рішення.

1.2.1 Традиційні сервіси прогнозу погоди

Традиційні погодні сервіси зазвичай орієнтуються на інформативну подачу даних без акценту на адаптацію до особистих потреб користувача. Основний інтерфейс таких додатків зосереджений на ключових метеорологічних параметрах: температура, опади, вологість, вітер, атмосферний тиск. Окрім поточної ситуації, зазвичай доступний короткотерміновий та довготерміновий прогноз, поданий у табличному або графічному вигляді.

Зручність користування часто досягається завдяки привабливому візуальному оформленню, однак це не завжди супроводжується інтелектуальною інтерпретацією даних. Користувачу потрібно самостійно оцінити зміст прогнозу та приймати рішення на його основі. При цьому відсутня будь-яка система рекомендацій, що могла б допомогти сформувати висновок з урахуванням контексту, типу активності або часу доби.

Додаткові можливості традиційних сервісів можуть включати інформацію про якість повітря, індекс ультрафіолетового випромінювання або час сходу і заходу сонця. Ці функції, хоч і корисні, подаються як окремі блоки даних і не інтегруються в загальну логіку допомоги користувачу у прийнятті рішень. У результаті взаємодія з додатком зводиться до механічного споживання інформації без підтримки або підказок. Інтерфейс

традиційного погодного сервісу (рисунок 1.3), який поєднує всі вказані риси, можна побачити на прикладі зображення нижче.

Рисунок 1.3 – Інтерфейс традиційного погодного сервісу

Незважаючи на візуальну привабливість і структурованість, така модель не враховує змінність контексту використання і не забезпечує адаптивність, що обмежує її ефективність у повсякденному застосуванні.

1.2.2 Погодні додатки з порадами щодо одягу

У межах ринку погодних сервісів існує окрема, менш чисельна категорія додатків, які поєднують прогноз погоди з порадами щодо вибору одягу. Такі рішення намагаються перейти від простої трансляції даних до прикладної інтерпретації, пропонуючи користувачеві варіанти одягу,

адаптовані до поточних або прогнозованих погодних умов. Хоча більшість подібних продуктів реалізовано у вигляді мобільних додатків, сама ідея поєднання метеорологічної інформації з порадами щодо стилю є надзвичайно актуальною і для вебсередовища.

Одним із найвідоміших прикладів є додаток Whatoweather – вебдодаток, який дозволяє користувачу обрати місто та тип активності, наприклад, прогулянка, велопоїздканн, після чого генерує загальні поради щодо одягу. Інтерфейс побудований доволі просто: користувач бачить температуру, умовні позначки погоди, а також текстову рекомендацію на кшталт «одягніть легку куртку». Однак такі рекомендації сформульовані за шаблоном і не адаптуються до індивідуальних переваг чи точних метеоумов у заданий момент. Штучний інтелект у класичному розумінні в роботі цього сервісу не використовується, що обмежує гнучкість підходу.

Ще одним прикладом є DressCast (раніше відомий як Cladwell) – мобільний додаток, який поєднує прогноз погоди з порадами щодо гардеробу користувача. Його функціональність базується на інтеграції з особистим гардеробом, тобто користувач вводить дані про свої речі, і система формує пропозиції залежно від погоди. Проте цей сервіс не є вебдодатком і не підтримує генерацію рекомендацій на основі відкритого природного запиту. Крім того, відсутня підтримка голосового або текстового діалогу з користувачем, що обмежує рівень персоналізації.

Інші сервіси, як-от Google Assistant або Amazon Alexa, мають голосові навички, що дозволяють отримувати погодні підказки, однак вони не спеціалізуються на одязі. У більшості випадків відповідь на запитання «що одягти сьогодні» зводиться до формального повторення прогнозу або короткої фрази, яка не враховує складних контекстних факторів, таких як тип події, місце перебування, особисті уподобання тощо. Це підкреслює той факт, що на ринку відсутні повноцінні вебрішення, які поєднували б сучасні можливості генеративного штучного інтелекту з погодними даними для створення дійсно персоналізованих порад.

Таким чином, хоча певні приклади погодних додатків з рекомендаціями щодо одягу справді існують, вони здебільшого реалізовані як мобільні продукти і мають обмежену інтелектуальну гнучкість. Вебдодатки в цій сфері поки що залишаються недорозвиненими, що відкриває широкі можливості для створення інноваційного рішення. Впровадження системи, яка поєднує погодні API з сучасними мовними моделями для генерації рекомендацій в реальному часі, є перспективним напрямом, який здатен зайняти нішу на ринку.

1.2.3 Порівняння ключових характеристик існуючих рішень

Порівняння існуючих рішень у сфері погодних додатків дає змогу виокремити декілька ключових характеристик, які впливають на ефективність та зручність користування. Насамперед це точність і деталізація прогнозу, якість візуалізації, швидкість оновлення даних, а також наявність додаткового функціоналу. У той час як більшість традиційних сервісів зосереджені на передаванні метеорологічної інформації, новіші рішення намагаються інтегрувати погодні дані в повсякденний досвід користувача.

Інтерфейс залишається одним із визначальних факторів при оцінці сервісу. Сучасні додатки мають привабливий дизайн, адаптивність до різних пристроїв і зрозумілу структуру. Водночас лише деякі з них мають дійсно інтуїтивну подачу даних, що дозволяє миттєво сприймати погодну ситуацію без глибокого аналізу числових показників. Перевагу отримують ті сервіси, які подають інформацію у вигляді піктограм, інфографіки або коротких текстових описів.

Функціональність також суттєво варіюється між додатками. Частина з них пропонує стандартний набір можливостей: температура, вітер, опади, індекс ультрафіолету. Інші додають нові модулі, як показники якості повітря, сповіщення про небезпечні погодні явища або розширений

погодинний прогноз. Ще менше сервісів інтегрує механізми рекомендацій, особливо щодо одягу чи поведінки користувача у певних умовах.

Загалом аналіз показує, що хоча функціональні й візуальні відмінності між сервісами значні, на ринку все ще бракує комплексних рішень, які б поєднували високоточний прогноз з персоналізованими рекомендаціями. Це свідчить про наявність потенціалу для створення нових підходів, які будуть орієнтовані не лише на передачу інформації, а й на активну допомогу у прийнятті повсякденних рішень.

1.2.4 Визначення слабких місць та проблем конкурентів

Незважаючи на наявність великої кількості погодних додатків, більшість із них мають спільні обмеження, які знижують практичну цінність сервісу. Основною проблемою є відсутність гнучкої персоналізації та неспроможність адаптуватися до конкретного контексту використання. Користувачі отримують однакові поради незалежно від типу активності, стилю життя або індивідуальних вподобань. Це особливо помітно у випадках, коли мова йде про рекомендації щодо одягу або планування активностей на відкритому повітрі.

Ще одним недоліком конкурентних рішень є використання шаблонного підходу до формування рекомендацій. У багатьох додатках поради щодо одягу не залежать від додаткових факторів, як-от вологість, вітер, реальне відчуття температури чи час доби. Відсутність штучного інтелекту або мовних моделей у процесі генерації рекомендацій обмежує гнучкість і якість взаємодії. Це створює потребу в нових рішеннях, які здатні інтегрувати дані, контекст і користувацькі сценарії в єдиний адаптивний інтерфейс. Тому й з’явилась ідея реалізації даного вебдодатку в рамках виконання кваліфікаційної роботи.

1.3 Огляд потреб користувачів

Розробка інноваційного вебдодатку потребує глибокого розуміння очікувань, мотивацій і поведінки кінцевих користувачів. У випадку погодного сервісу з рекомендаціями щодо одягу важливо враховувати не лише потребу в точному прогнозі, а й бажання отримати зрозумілу і прикладну пораду. Саме поведінкові особливості аудиторії визначають, яким чином має бути організована взаємодія із системою.

Користувачі орієнтуються не лише на температуру чи наявність опадів, а й на власні плани, стиль життя та індивідуальне сприйняття комфорту. У багатьох випадках навіть незначні зміни погодних умов впливають на рішення, що стосуються вибору одягу, маршруту або засобу пересування. Тому система повинна бути не просто інформативною, а контекстно чутливою до ситуацій, у яких знаходиться користувач.

Визначення потреб аудиторії також дозволяє виділити ключові проблеми, з якими стикаються користувачі в процесі використання існуючих рішень. Недостатня деталізація, відсутність адаптації до конкретного запиту та шаблонний підхід часто знижують довіру до сервісу. У цьому розділі буде розглянуто структуру цільової аудиторії, типові сценарії використання та основні очікування користувачів, що дозволить обґрунтувати підхід до побудови функціоналу майбутнього додатку.

1.3.1 Визначення цільової аудиторії

Цільова аудиторія погодного вебдодатку з рекомендаціями щодо одягу охоплює широкий спектр користувачів, однак основною групою є активні мешканці міст, які щодня приймають рішення щодо одягу перед виходом на вулицю. Це можуть бути студенти, офісні працівники, фрілансери, кур'єри, спортсмени, туристи та всі, хто регулярно переміщується пішки або громадським транспортом. Для цієї категорії

важливим є швидкий доступ до актуальної інформації, зручно поданої у прикладному форматі.

Окрему категорію становлять користувачі, які ведуть активний спосіб життя на відкритому повітрі, зокрема велосипедисти, бігуни, власники собак, а також ті, хто займається ранковими або вечірніми прогулянками. Для них погодні умови безпосередньо впливають на комфорт і безпеку, тому рекомендації щодо одягу мають враховувати не лише температуру, а й вологість, вітер та час доби. Такі користувачі схильні шукати прості, але точні підказки, які не потребують глибокого аналізу даних.

Також серед потенційних користувачів є особи, які цінують естетичний вигляд і бажають відповідати погоді, не втрачаючи стилю. У цьому випадку сервіс може бути корисним не лише з точки зору практичності, а й як елемент стилістичної підтримки. Незалежно від мотивації, спільною рисою усіх представників цільової аудиторії є бажання зекономити час на аналізі прогнозу і зменшити невизначеність при виборі одягу.

1.3.2 Аналіз потреб користувачів при виборі одягу за погодою

Потреба у рекомендаціях щодо одягу виникає в користувачів щодня, особливо у міжсезоння, коли погодні умови змінюються швидко й непередбачувано. Багато людей не впевнені, як краще одягтись при нестійкій погоді, щоб не змерзнути, не перегрітись і водночас виглядати доречно. Традиційні погодні сервіси надають лише температуру і ймовірність опадів, що змушує користувача самостійно інтерпретувати ці дані й приймати рішення інтуїтивно.

Користувачі часто скаржаться на те, що не знають, як зіставити прогноз із реальною потребою. Наприклад, температура +8°C може бути сприйнята по-різному залежно від вологості, вітру, сонячного світла або тривалості перебування на вулиці. У результаті люди орієнтуються на

власні відчуття або поради знайомих, що не завжди дає бажаний результат. Відсутність адаптивних рекомендацій призводить до помилок у виборі одягу, дискомфорту і, в деяких випадках, до проблем зі здоров’ям.

Крім того, для багатьох користувачів важливо, щоб поради були простими, конкретними та не вимагали аналізу кількох джерел. Очікується, що сервіс зможе дати однозначну відповідь у форматі «вдягни це», з урахуванням не лише температури, а й умов використання – наприклад, прогулянки, поїздки на велосипеді або виходу на роботу. Саме ця потреба у зрозумілих, швидких і релевантних порадах створює основу для реалізації інтелектуального інтерфейсу.

1.3.3 Поведінкові патерни та очікування користувачів

Поведінкові патерни користувачів у сфері погодних сервісів демонструють прагнення до зручності, швидкості та мінімізації розумових зусиль при щоденному прийнятті рішень. Більшість людей звертаються до прогнозу погоди не для глибокого аналізу, а для отримання конкретної відповіді на просте питання: «що одягнути сьогодні». Якщо сервіс не може швидко надати релевантну інформацію, користувачі схильні відмовлятися від його використання або шукати інші альтернативи.

Важливою особливістю поведінки є прагнення до мінімалізму в інтерфейсі та інформативності у виводах. Користувач не хоче витрачати час на перегляд кількох сторінок або аналіз іконок. Очікується, що вся необхідна інформація буде доступна за кілька секунд, бажано на одному екрані, у вигляді короткого тексту або візуального блока. Це пояснює популярність сервісів з лаконічним дизайном і зрозумілими порадами.

Користувачі також схильні до формування повторюваних сценаріїв взаємодії. Наприклад, щоденний перегляд прогнозу в один і той самий час або вибір певного режиму активності, як-от «дорога на роботу» чи «прогулянка з дитиною». Виявлення таких патернів дозволяє системі

адаптуватися до користувача і пропонувати персоналізовані поради на основі попереднього досвіду. Це створює ефект довіри і підвищує частоту використання сервісу.

Очікування користувачів у цій сфері можна охарактеризувати як потребу в розумному асистенті, який не просто повідомляє дані, а допомагає ухвалювати рішення. Особливо цінуються поради, які враховують деякий контекст – місцезнаходження, час доби, стиль життя або навіть настрій. У перспективі це відкриває шлях до створення сервісів нового покоління, які поєднують прогноз погоди, поведінкову аналітику та адаптивну комунікацію.

1.4 Функціональність додатку

Після аналізу предметної галузі, ринку конкурентів та потреб користувачів можна сформулювати загальне бачення функціональності майбутнього вебдодатку. Його головне призначення полягає у наданні користувачам персоналізованих рекомендацій щодо одягу на основі поточних або прогнозованих погодних умов. Важливо, щоб взаємодія з системою була простою, швидкою та інтуїтивною, без перевантаження зайвими деталями.

Функціональна частина системи охоплює модулі для пошуку та додавання міст, отримання актуальної погодної інформації, генерації персоналізованих рекомендацій щодо одягу, а також збереження та перегляду майбутніх поїздок. Ключовою особливістю стане використання зовнішніх API для збору метеоінформації, а також інтеграція з мовною моделлю, здатною генерувати рекомендації у зрозумілому й адаптованому до контексту вигляді. Такий підхід дозволить не лише автоматизувати процес прийняття рішень, а й зробити систему універсальною для різних типів користувачів.

Окрему увагу буде приділено технічним аспектам реалізації, зокрема вибору стеку технологій та організації архітектури додатку. Важливо також врахувати вимоги до зручності інтерфейсу та адаптивності дизайну для різних пристроїв. У межах підрозділів цього пункту буде розглянуто основні компоненти системи, програмні засоби, що використовуються для розробки, та ключові принципи, на яких базується побудова користувацького інтерфейсу.

1.4.1 Основні функції системи

Функціональність вебдодатку зосереджена навколо чотирьох основних компонентів, що разом формують повний користувацький сценарій, починаючи від додавання міст для моніторингу погоди, отримання актуальних метеоданих і генерації персоналізованих рекомендацій щодо одягу, і завершуючи можливістю створення подорожей із прогнозом та порадами на майбутні дати, забезпечуючи таким чином комплексну підтримку користувача на щодень і під час планування.

Першим є модуль пошуку та додавання міст, який дозволяє користувачеві швидко знаходити необхідне місто та додавати його до персонального списку. Додані міста відображаються у вигляді окремих карток на головній сторінці з короткою інформацією.

Другий компонент – модуль перегляду детальної погодної інформації. При відкритті сторінки міста система надсилає запит до зовнішнього погодного API, отримуючи актуальні дані про температуру, вологість, швидкість вітру, видимість та прогноз на кілька днів. Це забезпечує надійну основу для подальшої генерації порад.

Третім ключовим компонентом є інтеграція з OpenAI API. На основі погодних умов та обраного типу активності (прогулянка, біг, їзда на велосипеді) система формує рекомендації щодо одягу. Поради

відображаються у зручному вигляді акордеонів із поясненнями до кожного елементу.

Четвертий компонент – модуль планування поїздок. Користувач має змогу створити подорож, вибравши місто та дати. Система генерує прогноз на відповідний період і рекомендації щодо одягу для подорожі. Усі поїздки зберігаються у відповідному розділі для подальшого перегляду.

1.4.2 Технічний стек розробки

Для реалізації вебдодатку буде використано сучасний стек технологій, який забезпечує високу продуктивність, масштабованість і зручність розробки. На стороні клієнта застосовується бібліотека React у поєднанні з TypeScript, що дозволяє створити динамічний, типобезпечний інтерфейс. Інтерфейсні компоненти оформлюються за допомогою UI-бібліотеки Mantine, яка підтримує адаптивність і темну тему.

Для взаємодії з погодним API та мовною моделлю використовується внутрішній серверний шар, що здійснює запити до відповідних сервісів. Отримані дані обробляються, формуються у форматі, зручному для генерації промптів, і передаються в OpenAI API.

1.4.3 Вимоги до UX/UI

Інтерфейс вебдодатку має бути інтуїтивним, мінімалістичним та адаптивним до різних розмірів екранів. Основна увага приділяється зручності отримання відповіді без зайвих дій з боку користувача. Використання простих форм введення, автоматичного визначення локації та стислого текстового блоку з рекомендацією дозволяє досягти високої ефективності взаємодії. Дизайн має підтримувати як світлу, так і темну тему, забезпечуючи комфортне використання в різних умовах освітлення.

1.5 Постановка задачі

У результаті аналізу предметної галузі та визначення потреб користувачів сформовано чітке бачення проблеми, яку має вирішувати запропонований вебдодаток. Сучасні погодні сервіси не надають персоналізованих порад у зручному форматі, а існуючі рекомендаційні системи або побудовані на шаблонах, або орієнтовані виключно на мобільні пристрої. Це створює прогалину на ринку, яку доцільно заповнити за допомогою інтелектуального вебрішення.

Основною метою є розробка інноваційного вебдодатку, що дозволяє користувачу отримувати персоналізовані текстові рекомендації щодо одягу на основі актуальних погодних умов у заданій локації. Для досягнення цієї мети необхідно реалізувати функціональну зв’язку між введенням даних, збором прогнозу, генерацією тексту та виведенням результату.

Однією з ключових вимог до системи є мінімізація кількості дій, які має виконати користувач для отримання рекомендації. Очікується, що система зможе автоматично визначати місцезнаходження, збирати погодні дані та формувати відповідь без необхідності самостійного аналізу числових значень. Таким чином, користувач отримує готове рішення замість набору показників.

Ще одним важливим завданням є інтеграція з сучасною мовною моделлю, здатною адаптувати вихідну інформацію до контексту. Важливо, щоб результат був не лише точним, а й природним у формулюванні, легким для сприйняття і корисним у конкретній ситуації. Це дозволяє покращити користувацький досвід і зробити сервіс по-справжньому інтелектуальним.

У сукупності, ці завдання формують основу для побудови вебдодатку нового типу – системи, що поєднує погодні дані, штучний інтелект і гнучкий інтерфейс з метою надання прикладної допомоги користувачеві.

2 ПРОЕКТУВАННЯ СИСТЕМИ

2.1 Формалізація функціональних та нефункціональних вимог

Проектування системи неможливе без чіткого формулювання вимог до її функціонування, продуктивності, інтерфейсу та архітектурної побудови. На даному етапі важливо виявити основні потреби користувача, очікувану поведінку додатку, перелік доступних функцій і технічні обмеження, які можуть виникнути при реалізації. Це дозволяє закласти міцний фундамент для всієї подальшої розробки та забезпечити узгодженість між задумом і кінцевим результатом.

Формалізація вимог охоплює функціональні й нефункціональні характеристики, які мають бути реалізовані у вебдодатку. Функціональні вимоги описують, що саме має робити система: пошук міст, перегляд прогнозу погоди, генерація рекомендацій щодо одягу, створення подорожей тощо. Натомість нефункціональні вимоги стосуються продуктивності, стабільності, безпеки та зручності взаємодії користувача з системою.

Особливу увагу приділено вимогам до реалізації додатку як PWA, оскільки це дозволяє користувачам працювати з додатком як з нативною програмою, навіть за нестабільного інтернет-з’єднання. Такий підхід підвищує доступність і зручність користування, особливо для мобільних пристроїв, не потребуючи встановлення через традиційні маркетплейси.

У результаті цього етапу буде сформовано перелік конкретних вимог, який стане основою для побудови архітектури системи, вибору технологічного стеку, моделювання бази даних тощо

2.1.1 Функціональні вимоги до вебдодатку

Функціональні вимоги до вебдодатку визначають набір дій, які має мати можливість виконувати користувач, а також основні сценарії взаємодії

з інтерфейсом. Система повинна забезпечити користувача інструментами для пошуку міст, додавання їх до списку, перегляду прогнозу погоди та отримання рекомендацій щодо одягу. Ці дії є базовими для реалізації головної мети додатку – надання корисної інформації у зручному вигляді.

Користувач повинен мати змогу створити персоналізований список міст. Для кожного міста має бути реалізована можливість перегляду погодних даних, включаючи температуру, відчутну температуру, вологість, швидкість вітру, атмосферний тиск, видимість та прогноз на кілька днів. Важливим є також забезпечення інтерактивного переходу до сторінки з порадами щодо одягу, що генеруються відповідно до поточних умов і типу активності.

Однією з ключових функцій є модуль створення поїздок. Користувач повинен мати можливість вибрати місто, вказати назву поїздки, дати початку та завершення, після чого отримати прогноз погоди на обраний період та рекомендації, що саме взяти з собою. Кожна створена поїздка зберігається в особистому кабінеті та доступна для подальшого перегляду.

Система має також підтримувати функції авторизації, видалення міст і поїздок, перегляду профілю, перемикання між сторінками. Усі функції мають бути реалізовані у межах єдиного інтерактивного інтерфейсу з адаптацією під різні пристрої. Реалізація вищезазначених функцій забезпечує цілісну логіку використання додатку та дозволяє користувачеві повністю взаємодіяти з сервісом без потреби звернення до сторонніх джерел.

2.1.2 Нефункціональні вимоги до продуктивності та надійності

Нефункціональні вимоги визначають якісні характеристики вебдодатку, які впливають на зручність користування, ефективність роботи, стабільність та загальний користувацький досвід. У випадку розробки погодного сервісу з рекомендаціями щодо одягу особливо важливо

забезпечити швидке завантаження інтерфейсу, коректну роботу зовнішніх API та адаптацію під різні типи пристроїв.

Система повинна забезпечувати високу продуктивність незалежно від кількості запитів користувачів або обсягу отримуваних погодних даних. Ключовим аспектом є мінімізація часу очікування після взаємодії з елементами інтерфейсу, зокрема при завантаженні нових міст, формуванні поїздок або генерації порад. Важливо досягти відгуку системи протягом декількох секунд навіть при повільному інтернет-з’єднанні.

До вимог надійності належить обробка можливих помилок: недоступність зовнішніх API, помилкові запити, нестабільне з’єднання. Система має коректно реагувати на подібні ситуації – виводити відповідні повідомлення та не допускати аварійного завершення роботи. Важливою вимогою є також забезпечення збереження користувацьких даних та стану інтерфейсу у випадках оновлення сторінки чи тимчасової втрати зв’язку.

Система повинна масштабуватися без втрати якості при зростанні кількості користувачів. Це стосується як обробки одночасних запитів, так і загальної архітектури даних. Застосування хмарного середовища з розподіленим зберіганням дозволяє підвищити надійність і гарантувати доступ до сервісу без простоїв. Усі ці аспекти разом формують технічну стійкість та якісне функціонування вебдодатку в реальних умовах використання.

2.1.3 Вимоги до PWA

Однією з важливих нефункціональних вимог до сучасного вебдодатку є підтримка принципів прогресивного вебдодатку. Формат PWA дозволяє створити додаток, який поєднує переваги звичайного вебсайту та нативного мобільного додатку. Це забезпечує зручність для користувача, можливість автономної роботи та швидкий доступ до інтерфейсу без встановлення через магазини додатків.

Завдяки використанню PWA-додаток може працювати навіть за умов нестабільного або відсутнього інтернет-з’єднання. Це досягається шляхом кешування основних ресурсів і даних, які зберігаються у браузері користувача. Таким чином, при повторному відкритті додатку користувач отримує доступ до останньої завантаженої інформації та частково функціонального інтерфейсу, що підвищує загальну надійність системи.

Серед інших переваг PWA можна відзначити автоматичне оновлення додатку, підтримку push-сповіщень, швидкий запуск з головного екрана пристрою та адаптивний інтерфейс. Вебдодаток має вигляд і поведінку, подібні до мобільного додатку, що позитивно впливає на користувацький досвід. Особливо це актуально для додатку, який часто використовується у мобільному середовищі.

Підтримка стандартів PWA також дає змогу легко масштабувати систему під різні операційні системи без необхідності створення окремих додатків для Android чи iOS. Таким чином, реалізація вебдодатку з підтримкою PWA не лише підвищує його технічну гнучкість, а й робить його доступним для ширшого кола користувачів.

2.2 Моделювання сценаріїв взаємодії користувача з системою

Проектування функціональної моделі взаємодії користувача з системою є ключовим етапом у створенні інтуїтивно зрозумілого та ефективного вебдодатку. Вона дозволяє на етапі аналізу визначити основні логічні етапи використання інтерфейсу, встановити зв’язки між діями користувача та відповідними реакціями системи. Модель має забезпечити повне охоплення усіх очікуваних сценаріїв взаємодії користувача з інтерфейсом для досягнення основної мети вебдодатку – надання актуальної погодної інформації та персоналізованих рекомендацій щодо вибору одягу для прогулянок чи пробіжки.

У межах цього додатку користувач має пройти послідовність дій, яка починається з авторизації, додавання потрібного міста, перегляду метеоінформації та завершення у вигляді отриманих порад, адаптованих до погодних умов і типу активності. Усі ці етапи мають бути пов’язані з мінімальною кількістю переходів та забезпечувати простоту використання, особливо на мобільних пристроях.

Функціональна модель передбачає реалізацію кількох основних маршрутів взаємодії: перевірка погоди в місті, перегляд рекомендацій, створення поїздки з подальшим прогнозом і збереженням. Для кожного з маршрутів визначається початкова точка, набір дій користувача та кінцевий результат. Це дозволяє оптимізувати інтерфейс і уникнути перевантаження зайвими функціями.

Особливу увагу приділено моделі отримання порад щодо одягу. У цьому випадку функціональна логіка включає збір погодних параметрів через зовнішній API, передачу їх у запит до OpenAI API, обробку результату та його виведення у форматі акордеонів за категоріями активності. Важливо, щоб користувач отримував вичерпну, але структуровану відповідь без потреби додаткового аналізу.

Модель передбачає підтримку сценаріїв редагування списку міст, повторного перегляду рекомендацій, а також можливість перегляду вже збережених поїздок. Усі взаємодії мають бути реактивними, тобто миттєво реагувати на дії користувача без повного перезавантаження сторінки. Це дозволяє підвищити зручність взаємодії й забезпечити сучасний рівень користувацького досвіду.

Загалом функціональна модель вебдодатку спрямована на забезпечення швидкого доступу до цільової інформації, мінімізацію часу на навігацію та автоматизацію процесу прийняття рішень щодо підбору одягу. Завдяки правильному моделюванню сценаріїв вдається досягти взаємної відповідності між потребами користувача та функціоналом системи.

2.2.1 Перевірка погоди в місті та отримання поради щодо одягу

Сценарій взаємодії користувача з системою при перевірці погоди в місті та отриманні поради щодо одягу є одним із найбільш типових випадків використання вебдодатку. У цьому процесі задіяні основні компоненти функціональної моделі, включаючи інтерфейс користувача, погодний API, систему генерації порад та мовну модель. Усі ці елементи взаємодіють у межах єдиного логічного ланцюга, що забезпечує швидке отримання персоналізованої інформації для кінцевого користувача.

Далі можна побачити діаграму послідовності (рисунок 2.1), яка демонструє етапи взаємодії між користувачем та системою у сценарії перевірки погоди в місті. Користувач обирає місто зі списку або додає нове, після чого інтерфейс додатку надсилає запит до погодного API. Отримані метеодані передаються до системи рекомендацій, яка через інтеграцію з OpenAI API генерує текстові поради щодо одягу, адаптовані до типу активності.

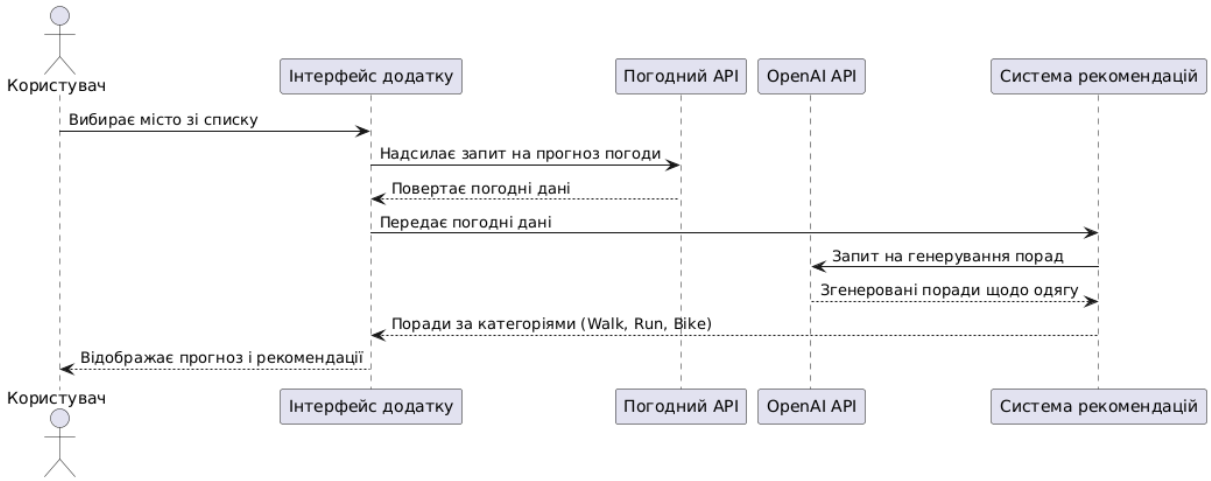

Рисунок 2.1 – Перший типовий сценарій використання додатку

На етапі отримання даних з погодного API система збирає базову інформацію про температуру повітря, вологість, швидкість вітру,

атмосферний тиск та інші ключові параметри. Ці дані є основою для подальшого аналізу і використовуються як візуально на інтерфейсі додатку, так і як вхідні значення у мовну модель. Важливо, щоб відповідь API була отримана швидко та без помилок, адже від цього залежить загальний користувацький досвід.

Система рекомендацій отримує структуровані погодні дані та передає їх до OpenAI API. Мовна модель аналізує числові параметри та текстові описові характеристики й формує відповідну пораду для користувача. Рекомендації згруповані за категоріями Walk, Run і Bike, що дозволяє користувачу адаптувати одяг відповідно до свого плану активності. Вивід рекомендацій відбувається у вигляді акордеонів, що відкриваються для детального перегляду кожної поради.

Інтерфейс додатку відображає результат у зручному та інтуїтивно зрозумілому вигляді. Користувач бачить загальний прогноз на найближчі години та дні, а також отримує персоналізовану текстову відповідь на запитання про те, що одягти за поточних погодних умов. Таке поєднання точності метеоінформації та гнучкості генеративної моделі створює відчуття індивідуального підходу.

У підсумку можна сказати, що перевірка погоди та отримання поради щодо одягу є сценарієм, який демонструє сильні сторони системи. Він поєднує в собі інтеграцію з зовнішніми джерелами, автоматизацію на основі штучного інтелекту та зрозумілий інтерфейс для користувача. Усі компоненти системи працюють узгоджено, забезпечуючи ефективність та зручність взаємодії.

2.2.2 Планування поїздки та підбір одягу на визначені дати

Сценарій планування поїздки з подальшим підбором одягу охоплює розширену взаємодію користувача з вебдодатком. У цьому випадку користувач не лише переглядає поточну погоду, а формує подію в

майбутньому, для якої необхідно отримати прогноз погоди та відповідні рекомендації. Такий підхід передбачає роботу з діапазоном дат та потребує більш складної обробки даних на стороні системи.

Далі можна побачити діаграму послідовності (рисунок 2.2), яка відображає логіку дій при плануванні поїздки. Користувач переходить до розділу «Мої поїздки», де обирає місто та зазначає початкову й кінцеву дати запланованої подорожі. Інтерфейс додатку фіксує введену інформацію та зберігає поїздку до бази даних. Після цього відправляється запит до погодного API з урахуванням вказаного діапазону дат.

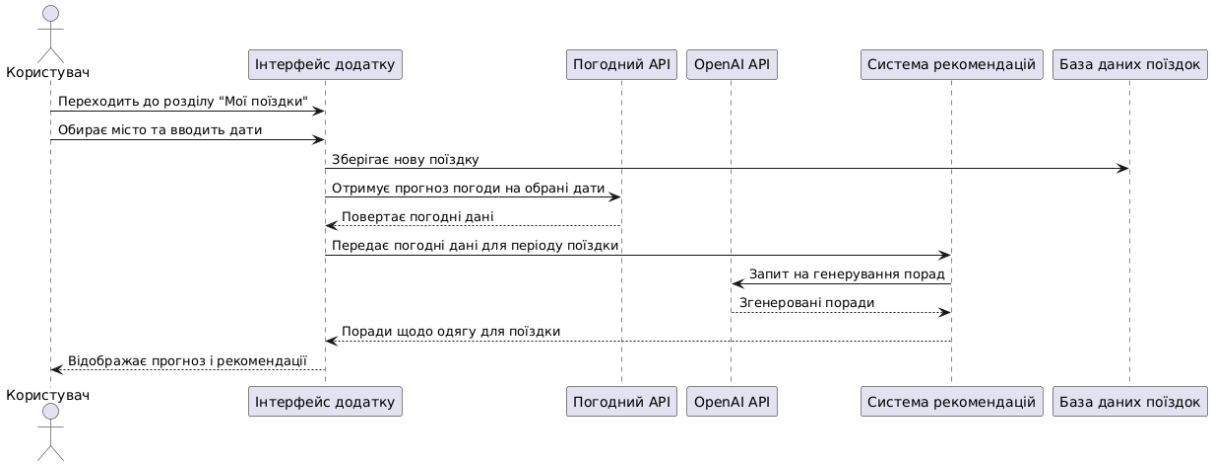

Рисунок 2.2 – Другий типовий сценарій використання додатку

Погодний API повертає прогноз на кілька днів наперед, охоплюючи весь період поїздки. Отримані погодні дані передаються до системи рекомендацій, яка, своєю чергою, формує відповідну структуру даних для кожного дня. У такий спосіб забезпечується контекстна обробка нашої інформації – кожен день аналізується окремо з урахуванням його температурних та кліматичних характеристик.

Система рекомендацій на основі цих даних ініціює запит до OpenAI API. Мовна модель генерує добірку текстових порад, де вказано рекомендований одяг на кожен день подорожі. Це дозволяє користувачеві

отримати максимально адаптовану інформацію щодо речей, які варто взяти з собою в дорогу. У результаті створюється структурований список порад, що відповідає як погодним умовам, так і потребам у зручності.

Усі поради групуються в межах однієї поїздки та зберігаються в базі даних. Користувач може повернутися до збереженої події в будь-який момент та переглянути як погодні умови, так і рекомендації. Це забезпечує зручність у плануванні та мінімізує ризик неправильного вибору одягу в умовах змінного клімату.

Таким чином, функціональність планування поїздок у поєднанні з персоналізованими порадами щодо одягу забезпечує додаткову цінність додатку. Вона розширює його можливості за межі щоденного використання, перетворюючи систему на корисний інструмент підготовки до подорожей. Завдяки цьому користувач отримує не лише інформацію, а й готові рішення, що підвищують рівень комфорту.

2.3 Побудова архітектури вебдодатку

Архітектура вебдодатку побудована на принципах клієнт-серверної моделі, що дозволяє забезпечити чітке розділення відповідальностей між компонентами системи. Клієнтська частина відповідає за взаємодію з користувачем, збір вхідних даних та ініціацію запитів. Серверна частина виконує обробку запитів, логіку формування порад та взаємодію з зовнішніми сервісами.

Нижче представлено (рисунок 2.3) загальну архітектурну схему системи, яка демонструє структуру компонентів та напрямки обміну даними між ними. Користувач взаємодіє з інтерфейсом додатку, який передає запити через фронтенд-сервіс до бекенду. Серверна логіка обробляє запити та забезпечує зв’язок із зовнішніми джерелами інформації, такими як погодний API та OpenAI API.

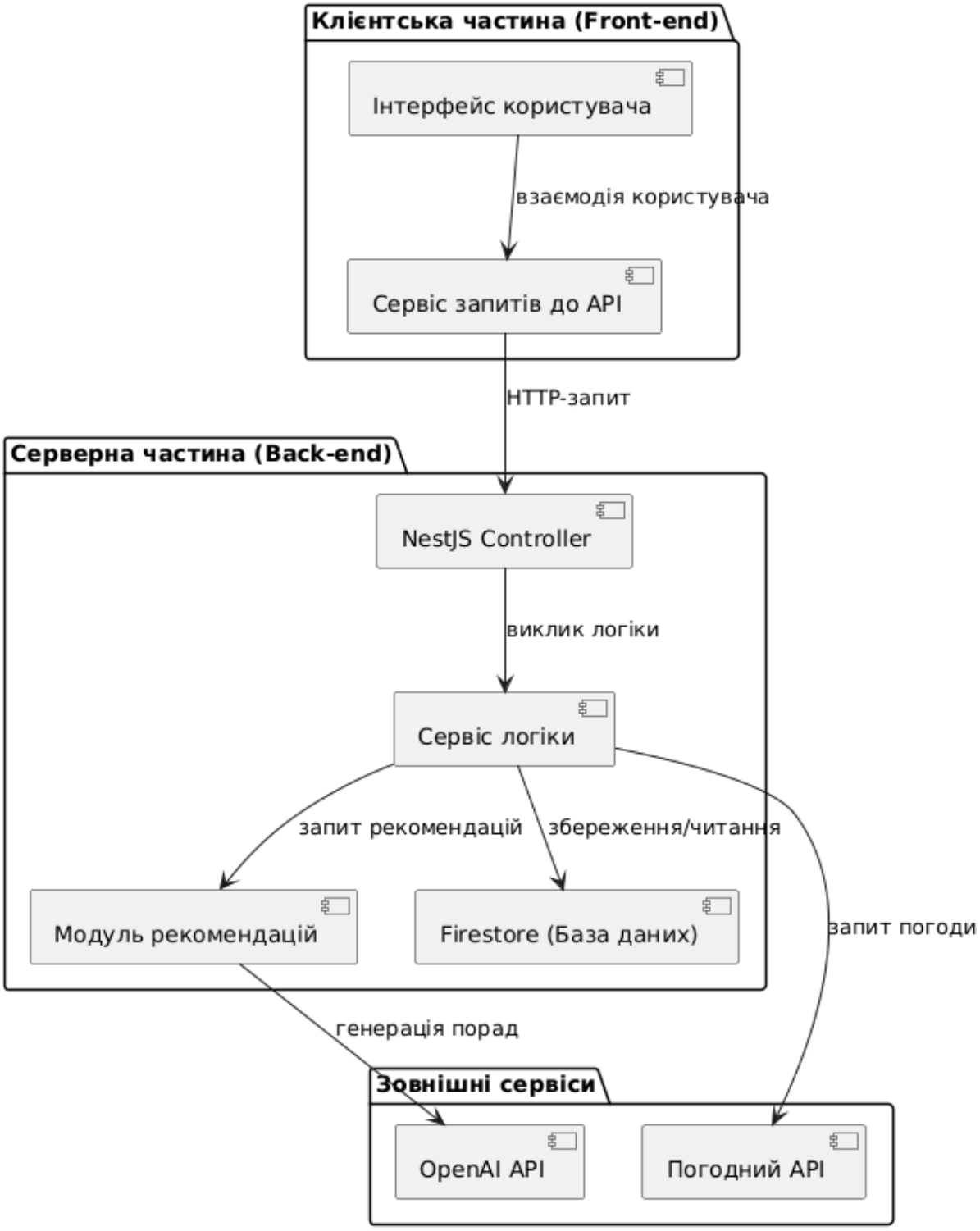

Рисунок 2.3 – Архітектурна схема системи

Клієнтська частина реалізована з урахуванням принципів зручності та швидкодії. Вона містить модулі для введення запитів користувача, зокрема вибір міста або дати поїздки, а також відображення результатів у зручному

форматі. Зв’язок між клієнтом і сервером реалізовано за допомогою актуальних HTTP-запитів до API.

Серверна частина реалізована на базі NestJS, що забезпечує модульність та масштабованість системи. Контролери приймають запити від фронтенду та делегують обробку до відповідних сервісів. Сервіси логіки здійснюють підготовку запитів до зовнішніх API, отримання результатів та збереження рекомендацій до бази даних.

У складі серверної частини також передбачено модуль рекомендацій, який формує запит до OpenAI API на основі зібраних метеоданих. Після генерації відповідь з порадами надсилається назад на клієнт для відображення. База даних Firestore використовується для зберігання історії запитів, даних про міста та інших службових відомостей.

Зовнішні сервіси, такі як погодний API та OpenAI API, є ключовими джерелами даних і рекомендацій. Вони забезпечують систему актуальною інформацією, а також дають змогу адаптувати відповіді до конкретного контексту. Інтеграція з цими сервісами є невід’ємною частиною загальної архітектури.

2.4 Концепція NoSQL у проектуванні бази даних

Зберігання даних у розроблюваній системі реалізовано на основі концепції NoSQL, що дає змогу зручно оперувати гнучкими та неструктурованими структурами. Вибір такого підходу зумовлений потребою у високій швидкості читання і запису, масштабованості та можливості роботи з динамічними структурами об'єктів, які формуються під час інтеграції з API.

У системі використовується база даних Firestore від Google, яка є частиною хмарної платформи Firebase. Цей інструмент дозволяє зберігати документи у вигляді колекцій з вкладеними об'єктами. Така модель особливо зручна для зберігання даних про міста, погодні умови, результати

запитів користувача та згенеровані рекомендації, які мають різну структуру залежно від типу активності.

Для кожного запиту створюється окремий документ, у якому міститься інформація про параметри запиту (місто, дата, активність), отримані метеоумови та сформовані рекомендації. Це дає змогу легко організовувати історію використання системи, а також забезпечити можливість повторного перегляду порад без повторного звернення до зовнішніх API.

Перевагою Firestore є підтримка офлайн-режиму та синхронізації в реальному часі, що може бути корисним у подальших розширеннях додатку, наприклад, при створенні мобільної версії. Крім того, Firestore має вбудовані механізми безпеки на основі правил доступу, які дозволяють налаштовувати авторизований доступ до даних.

Структура колекцій у Firestore розділяється на кілька основних категорій: користувачі, міста, історія запитів та збережені рекомендації. Така організація дозволяє легко масштабувати систему і реалізовувати нові функції, зокрема персоналізовану аналітику або сегментацію користувачів.

Firestore також ефективно взаємодіє з іншими сервісами Google, що спрощує розгортання додатку в екосистемі Google Cloud. Завдяки цьому забезпечується стабільна робота, мінімальна затримка в обробці запитів і висока надійність зберігання критичних даних системи.

2.5 Інтеграція з зовнішніми API

Інтеграція з зовнішніми API є ключовим елементом функціонування вебдодатку, оскільки саме завдяки цим сервісам система отримує актуальні погодні дані та генерує адаптовані рекомендації. Для цього передбачено взаємодію з двома основними джерелами – метео API та мовною моделлю через OpenAI API. Вся взаємодія реалізована у фоновому режимі через серверну частину додатку.

Використання зовнішніх API дозволяє знизити навантаження на власну інфраструктуру, зберігаючи при цьому високу якість та точність обробки запитів. Комунікація з сервісами здійснюється на основі REST-запитів з обробкою відповіді у форматі JSON, що дозволяє гнучко витягувати необхідні параметри та будувати відповідь користувачеві у зручному вигляді. Це також відкриває можливості для масштабування та підключення нових сервісів у майбутньому.

Зовнішні API використовуються як у режимі перегляду поточної погоди, так і в режимі планування на майбутні дати. У результаті система стає універсальним інструментом, який поєднує метеоаналітику та генеративні технології, що забезпечує релевантність та персоналізацію наданих порад.

2.5.1 Отримання погодних даних через зовнішній метео API

Для отримання метеоінформації у вебдодатку використовується API сервісу open-meteo. Цей сервіс надає широкий спектр погодних параметрів, зокрема температуру, швидкість та напрямок вітру, вологість, атмосферний тиск та інші показники, які можуть впливати на формування рекомендацій. Дані доступні як у режимі реального часу, так і для прогнозу на обрані дати, що дозволяє реалізувати функціонал як для поточного перегляду, так і для планування поїздок.

Перед запитом до open-meteo API необхідно отримати координати населеного пункту, який ввів користувач. Для цього використовується інший зовнішній сервіс – geodb API. Він дозволяє за назвою міста знаходити відповідні географічні координати (широту і довготу), які вже можуть бути передані у запиті до open-meteo. Таким чином, geodb API виступає допоміжним сервісом для точного позиціонування.

Використання обох сервісів відбувається послідовно. Спочатку обробляється запит на пошук міста через geodb API, після чого координати

використовуються для формування повноцінного погодного запиту до open-meteo. Такий підхід забезпечує гнучкість та уніфікованість взаємодії з погодними сервісами, незалежно від країни або типу населеного пункту.

Результати обох запитів обробляються серверною частиною додатку та передаються у вигляді структурованих даних до клієнтської частини або використовуються для генерації текстових порад. Це дозволяє централізувати логіку, підвищити швидкодію та забезпечити стабільність навіть при зростанні кількості одночасних користувачів.

2.5.2 Генерація текстових порад за допомогою OpenAI API

Генерація текстових порад щодо одягу реалізується за допомогою інтеграції з OpenAI API, що забезпечує можливість формувати динамічні відповіді на основі заданих вхідних параметрів. До цих параметрів належать температура, вологість, вітер, опис погодних умов та тип запланованої активності – наприклад, прогулянка, пробіжка або поїздка на велосипеді. На основі цих даних створюється промт для мовної моделі.

Серверна частина формує текст запиту, в якому враховано як метеоумови, так і контекст використання. Наприклад, якщо прогнозується дощ, модель отримає вказівку врахувати це при формуванні порад. Аналогічно, якщо очікується спекотна погода і тип активності – біг, система попросить сформулювати поради, що включають легкий одяг і захист від сонця.

Відповідь OpenAI API містить готовий текст, структурований за типами активностей. Кожна група порад подається у зручному вигляді – часто з коротким описом і поясненням, чому саме такий одяг рекомендовано. Це робить рекомендації не лише релевантними, а й зрозумілими та обґрунтованими для користувача.

Таким чином, мовна модель OpenAI не просто підставляє шаблонну відповідь, а адаптує її до конкретних умов і потреб. Це дозволяє надати

користувачам персоналізований сервіс, який враховує контекст середовища, заплановану активність та часові параметри. У поєднанні з актуальними погодними даними, така інтеграція є ядром інтелектуальної частини системи.

2.6 Вибір технологічного стеку та середовища розробки

Для реалізації вебдодатку було обрано сучасний технологічний стек, який забезпечує високу продуктивність, простоту підтримки та можливість масштабування. Уся система розділена на клієнтську та серверну частини, які взаємодіють між собою через HTTP-запити. Такий підхід дозволяє гнучко оновлювати окремі модулі без впливу на загальну стабільність.

Клієнтська частина реалізована з використанням React. Цей фреймворк дає змогу побудувати односторінковий інтерфейс, який швидко реагує на дії користувача без повного перезавантаження сторінки. Для розмітки та стилізації використовуються сучасні бібліотеки інтерфейсів, які забезпечують адаптивність до різних розмірів екрана. React Router використовується для переходу між сторінками, а бібліотека Zustand – для управління станом.

Серверна частина побудована з використанням NestJS – прогресивного фреймворку для Node.js, який базується на архітектурі з модулями, контролерами та сервісами. Завдяки цьому структура додатку залишається чіткою, а логіка легко розділяється на окремі частини. NestJS також дозволяє зручно реалізовувати валідацію, логування, обробку помилок та виклики до зовнішніх API.

Для зберігання даних використовується Firestore – документоорієнтована база даних, яка є частиною Firebase. Вона забезпечує гнучке зберігання інформації у форматі колекцій і документів, підтримує режим реального часу та автоматичну синхронізацію, що робить її

оптимальною для сучасного вебдодатку. Firestore також дозволяє швидко масштабувати додаток і керувати доступом на основі ролей користувачів.

Отримання погодних даних реалізовано через open-meteo API, а для отримання географічних координат використовується geodb API. Генерація текстових порад реалізована через OpenAI API. Для кожного запиту формується промт, який містить погодні умови, тип активності та інші параметри. Відповідь моделі використовується у клієнтському інтерфейсі для відображення порад у зручному форматі.

2.7 План розробки та етапи реалізації

План розробки вебдодатку передбачає поетапну реалізацію всіх функціональних компонентів системи. На початковому етапі заплановано створення структури клієнтської та серверної частин, налаштування середовища розробки, реалізацію маршрутизації між основними сторінками додатку та впровадження механізму авторизації. Після цього буде реалізовано базову логіку взаємодії користувача з інтерфейсом – пошук міст, перегляд детальної інформації та управління списком поїздок.

На наступному етапі передбачається інтеграція з погодними сервісами open-meteo API та geodb API. Необхідно реалізувати отримання координат міста за введеною назвою, формування запиту до погодного API та обробку відповіді. Дані зберігатимуться у Firestore. Далі заплановано підключення OpenAI API, створення шаблонів запитів до мовної моделі та формування порад на основі погодних умов і обраної активності. Отримані результати мають бути інтегровані до інтерфейсу у вигляді акордеонів.

Завершальний етап включатиме розробку функціоналу планування поїздок, створення форм введення дат та міста, генерацію прогнозу погоди на вибраний період, а також генерацію відповідних рекомендацій.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Інструменти реалізації вебдодатку

У межах програмної реалізації було використано низку сучасних технологій, які забезпечують ефективну роботу системи на клієнтському та серверному рівнях. Вибір інструментів був зумовлений потребою у створенні швидкого, надійного та зручного для користувача вебдодатку, здатного працювати в режимі реального часу та взаємодіяти з зовнішніми сервісами.

Застосування компонентного підходу, типізованої мови програмування та готових бібліотек значно спростило процес розробки й дозволило зосередитися на основному функціоналі. Розподіл між клієнтською та серверною частинами забезпечив чітке відокремлення відповідальностей і сприяв кращій масштабованості системи.

Особливу увагу було приділено інтеграції зовнішніх API, налаштуванню аутентифікації та організації бази даних, що дозволяє зберігати інформацію про користувачів, міста та поїздки. Загальний стек технологій відповідає сучасним вимогам до вебдодатків і створює надійну основу для подальшого розширення системи.

3.1.1 Мова програмування

Для реалізації вебдодатку було обрано мову програмування TypeScript, яка поєднує можливості JavaScript із перевагами статичної типізації. Цей вибір зумовлений потребою у масштабованому, безпечному та легко підтримуваному коді. TypeScript дозволяє чітко описувати структуру даних і логіку взаємодії між компонентами, зменшуючи ймовірність помилок під час виконання.

Завдяки підтримці сучасних стандартів ECMAScript, TypeScript забезпечує зручну роботу з асинхронним кодом, модулями та класами. Це особливо важливо для розробки вебдодатків, які активно працюють з API, обробляють зовнішні дані та мають багатофункціональний інтерфейс. Крім того, використання типів підвищує якість автодоповнення в редакторах коду та пришвидшує процес розробки.

Мова добре інтегрується з обраними фреймворками та інструментами розробки, дозволяючи створювати як клієнтську, так і серверну логіку в одному технологічному стеку. Це сприяє уніфікації підходів у межах проекту, зменшенню часу на навчання та спрощенню обміну знаннями між розробниками.

Загалом TypeScript є сучасною мовою, яка задовольняє вимоги до продуктивності, безпеки та читабельності коду. Саме тому вона стала основною мовою реалізації програмного забезпечення у межах цього проекту.

3.1.2 Фреймворки та бібліотеки клієнтської частини

Клієнтська частина вебдодатку була реалізована з використанням бібліотеки React, яка є однією з найпопулярніших технологій для розробки інтерфейсів користувача. React дозволяє будувати компонентну структуру додатку, що значно спрощує керування станом, повторне використання елементів і модульність архітектури. Компоненти можна ізолювати один від одного, що сприяє стабільності додатку навіть при розширенні функціональності. Основна перевага React полягає у його віртуальному DOM, який забезпечує високу швидкість оновлення інтерфейсу без потреби повного перезавантаження сторінки.

У проекті активно використовується підхід з розділенням логіки компонентів на окремі частини. Кожен компонент відповідає за конкретний блок додатку, наприклад, форму введення, картку міста або модуль

рекомендацій. Така структура дозволяє швидко вносити зміни, тестувати окремі частини системи та ефективно масштабувати рішення. React також підтримує JSX – розширення синтаксису JavaScript, що дозволяє писати елементи інтерфейсу у вигляді коду, максимально наближеного до HTML, що позитивно впливає на читабельність.

Для маршрутизації сторінок застосовується бібліотека React Router, яка забезпечує перемикання між розділами без перезавантаження сторінки. Це створює відчуття роботи зі справжнім односторінковим додатком. Користувачі можуть переходити від головної сторінки до пошуку міст, перегляду деталей, а також сторінки з поїздками – усе це реалізовано за допомогою маршрутизатора, який синхронізується зі станом додатку та зберігає історію переходів.

Для побудови сучасного і зручного інтерфейсу було обрано бібліотеку Mantine UI. Вона надає великий набір готових компонентів, таких як кнопки, випадаючі списки, форми, вкладки, акордеони тощо. Усі елементи оформлені в єдиному стилі та адаптовані до темної теми інтерфейсу. Це дозволяє розробникам зосередитися на логіці додатку, не витрачаючи зайвий час на ручну стилізацію. Крім того, Mantine підтримує медіазапити для адаптивного дизайну, що забезпечує коректне відображення інтерфейсу на різних розмірах екранів.

Окрему увагу приділено організації стану компонентів і передачі даних. У рамках додатку використовуються локальні стани компонентів та контексти, які дозволяють передавати дані між віддаленими компонентами без зайвої складності. Це дозволяє контролювати авторизацію, зберігання вибраних міст, дані про заплановані поїздки та реактивно оновлювати інтерфейс без ручного втручання. Для форм використовуються керовані поля, що зв’язуються зі станом і дозволяють проводити валідацію та обробку введених користувачем даних.

Крім цього, бібліотека Mantine надає утиліти для сповіщень, модальних вікон, скелетонів завантаження та інші інтерактивні елементи,

які покращують користувацький досвід. Завдяки гнучкості налаштувань і темізації, інтерфейс зберігає єдину візуальну ідентичність і відповідає сучасним очікуванням користувача. Також було реалізовано адаптивний дизайн, що дає змогу комфортно користуватися вебдодатком на мобільних пристроях, планшетах і десктопах без втрати функціональності чи зручності.

Під час розробки було враховано особливості реактивної природи React – усі оновлення погодних даних, рекомендацій або міст відбуваються автоматично при зміні стану, без необхідності ручного оновлення сторінки. Такий підхід значно покращує інтерактивність і дозволяє швидко реагувати на дії користувача. Наприклад, після додавання нового міста його картка одразу з'являється на головній сторінці, а після планування поїздки її деталі доступні у відповідному розділі.

Окрім основних бібліотек, важливу роль у клієнтській частині відіграє Vite – сучасний інструмент для збирання фронтенд проектів. Його перевагою є миттєве завантаження модулів за потреби, що значно скорочує час запуску дев-сервера. У порівнянні з традиційними інструментами на кшталт Webpack, Vite забезпечує набагато кращу продуктивність під час розробки, особливо у великих проектах з великою кількістю компонентів.

Vite також використовується для створення фінальної збірки додатку, оптимізуючи JavaScript, CSS та інші ресурси. У результаті користувач отримує швидкий інтерфейс з мінімальним часом завантаження, що позитивно впливає на зручність користування додатком. Завдяки зручній інтеграції з React та підтримці TypeScript, Vite став невід’ємною частиною розробницького процесу у цьому проекті.

Усе це в комплексі створює надійну, зручну та ефективну клієнтську частину, яка є основною точкою взаємодії користувача з вебдодатком. Обрані технології забезпечують високу швидкість відгуку, зручний інтерфейс і можливість подальшого розширення функціоналу.

3.1.3 Фреймворки та бібліотеки серверної частини

Серверна частина вебдодатку реалізована з використанням мови TypeScript, що забезпечує уніфікацію технологічного стеку між клієнтом і сервером. Це дозволяє не лише зменшити кількість помилок при інтеграції обох частин системи, а й прискорити розробку завдяки повторному використанню типів, моделей даних та утиліт. Завдяки статичній типізації, що підтримується TypeScript, можна забезпечити високу надійність коду на етапі розробки і виявляти помилки ще до виконання. Це критично важливо в умовах інтенсивної роботи з зовнішніми API та асинхронною логікою обробки запитів.

Основним середовищем розгортання серверної частини стала платформа Google Cloud Functions. Вона дозволяє створювати серверні функції без необхідності підтримки окремого сервера або інфраструктури. Завдяки цьому можна значно скоротити час розгортання та експлуатаційні витрати. Кожна функція виконується ізольовано й автоматично масштабується відповідно до навантаження. Це дає можливість обробляти запити до погодних сервісів, генерувати рекомендації за допомогою OpenAI API, а також взаємодіяти з базою даних Firestore швидко і без затримок.

Серверна логіка організована у вигляді окремих обробників, кожен з яких відповідає за певну дію – наприклад, отримання метеоінформації, генерацію текстових порад або збереження користувацьких налаштувань. Обробники побудовані з використанням асинхронного програмування, що забезпечує ефективну обробку запитів без блокування потоку виконання. Це особливо важливо при роботі з мережею та зовнішніми джерелами даних, де затримки можуть впливати на загальну швидкодію системи.

Інтерфейси обробників побудовані за принципами REST API, що дозволяє легко масштабувати систему та підключати нові компоненти без зміни існуючої логіки. Усі запити надходять до конкретних функцій через HTTPS і повертають результат у форматі JSON, що забезпечує сумісність з

клієнтською частиною. Такий підхід спрощує тестування, налагодження та інтеграцію нових функцій. Крім того, завдяки Google Cloud Functions усі логи, помилки та час виконання зберігаються у хмарному моніторингу, що спрощує процес підтримки та оптимізації.

У межах реалізації також використовується система обробки параметрів запиту, перевірки допустимості вхідних даних і логування важливих подій. Це дозволяє гарантувати надійність навіть за умови високого навантаження або непередбачуваних збоїв з боку зовнішніх сервісів. У разі виникнення помилки система повертає інформативне повідомлення, що дозволяє користувачеві краще зрозуміти причину проблеми, а розробникові – швидко її діагностувати.

Таким чином, серверна частина вебдодатку виконує роль інтеграційного ядра, яке координує взаємодію між клієнтом, базою даних і зовнішніми API. Обрана архітектура на основі безсерверної інфраструктури дозволяє мінімізувати технічні витрати та водночас забезпечити гнучкість, масштабованість і високу доступність сервісу. Завдяки обраному підходу, серверна логіка залишається прозорою, масштабованою та легко адаптується до нових вимог, які можуть з’явитися в процесі подальшого розвитку проекту.

3.1.4 Аутентифікація та зовнішні API

Одним із важливих компонентів вебдодатку є механізм аутентифікації та управління доступом користувачів. У проекті реалізовано зовнішню систему аутентифікації через сервіс Auth0, що дозволяє делегувати процес перевірки особистості користувача надійному постачальнику. Це рішення забезпечує як безпеку, так і зручність, оскільки дозволяє авторизуватися через поширені платформи, зокрема Google або електронну пошту. Використання Auth0 дає змогу уникнути складної реалізації власної системи

входу та одночасно гарантувати відповідність сучасним вимогам захисту персональних даних.

Після успішної авторизації Auth0 повертає токен доступу, який зберігається на клієнтській стороні та передається з кожним запитом до серверної частини. Цей токен використовується для ідентифікації користувача та перевірки його прав на виконання певних дій, наприклад, додавання або видалення міста, створення поїздок, отримання персоналізованих рекомендацій. Завдяки цьому підхід до управління доступом є централізованим і легко масштабованим, що важливо в умовах поступового розширення функціональності проекту.

Інтеграція з Auth0 передбачає також обробку користувацького профілю – після входу до системи програма може отримувати такі базові дані, як ім’я, електронну пошту, аватар користувача. Ця інформація не лише дозволяє персоналізувати інтерфейс, але й забезпечує відображення корисних елементів навігації, наприклад, меню з ім’ям користувача або аватаром. Усі ці дані зберігаються локально протягом сесії, а обробка запитів до зовнішніх API відбувається з урахуванням прив’язки до відповідного користувача.

У межах серверної частини кожен запит, що надходить від клієнта, супроводжується валідацією токена через внутрішні механізми безпеки. Це гарантує, що лише аутентифіковані користувачі можуть здійснювати доступ до персональних даних або ініціювати дії, пов’язані з користувацькою взаємодією. При цьому реалізовано принцип ізоляції: дані одного користувача не можуть бути доступні іншому навіть у випадку некоректного запиту, оскільки всі операції перевіряють відповідність ідентифікатора в токені.

Окрім аутентифікації користувача, система також активно взаємодіє із зовнішніми API, зокрема OpenAI API, Open-Meteo API та GeoDB API. Усі запити до цих сервісів потребують авторизації або передачі ключів доступу, які зберігаються у захищеному середовищі Google Cloud Functions. Таким

чином, безпека переданих запитів та стабільність взаємодії з API гарантовані незалежно від навантаження. Ключі не передаються клієнту, що мінімізує ризик їх витоку або зловживання.

Завдяки чітко спроектованій архітектурі управління доступом, система гарантує як приватність користувача, так і стабільну взаємодію з зовнішніми сервісами. Усі компоненти побудовані з урахуванням принципів захищеного програмування, включаючи перевірку вхідних параметрів, обмеження прав доступу та обробку потенційних помилок. Це робить додаток надійним для повсякденного використання та дозволяє масштабувати його функціональність у майбутньому без шкоди для безпеки або стабільності.

3.1.5 База даних та зберігання інформації

Для зберігання даних у вебдодатку використовується хмарна база даних Firestore, яка входить до складу екосистеми Google Firebase. Цей інструмент дозволяє реалізувати масштабоване, безсерверне сховище з підтримкою режиму реального часу, що особливо актуально для проектів із динамічно змінюваною інформацією. Firestore базується на моделі документів і колекцій, що забезпечує гнучкість у структурі даних та дозволяє легко адаптувати її до функціональних потреб додатку. Зберігання даних відбувається у вигляді вкладених об'єктів, де кожен документ може містити як прості значення, так і вкладені масиви або документи.

У рамках реалізації було визначено основні колекції, відповідальні за збереження інформації про користувачів, міста, поїздки та згенеровані поради. Дані користувача включають унікальний ідентифікатор, отриманий після авторизації через Auth0, а також список міст, обраних для відображення. Кожне місто має власний документ з інформацією про назву, країну, погодні параметри та відповідні поради. Крім того, окремо зберігаються заплановані поїздки, що містять дату початку й завершення,

назву поїздки та список речей, рекомендованих до взяття на основі прогнозу.

Firestore також дозволяє реалізувати простий і ефективний доступ до даних через SDK, що використовується у клієнтській частині. Таким чином, клієнт напряму взаємодіє з базою даних у межах дозволених операцій, з урахуванням правил безпеки, встановлених у Firestore. Це дозволяє знизити навантаження на серверну частину й спростити логіку обробки запитів. Для гарантії конфіденційності та захисту даних реалізовано перевірку авторизації перед виконанням будь-якої операції зчитування або запису, що забезпечує доступ до інформації виключно для відповідного користувача. Завдяки Firestore реалізація зберігання виявилася зручною, масштабованою та придатною для подальшого розширення функціональності системи.

3.2 Реалізація додатку

Програмна реалізація вебдодатку охоплює створення як клієнтської, так і серверної частини, які взаємодіють між собою через запити до API. У межах реалізації були розроблені модулі для роботи з інтерфейсом користувача, обробки запитів до зовнішніх сервісів, генерації текстових рекомендацій та збереження даних у базі. Усі елементи системи функціонують узгоджено, утворюючи цілісну структуру, в якій кожен компонент виконує свою роль у загальному процесі взаємодії. Завдяки цьому забезпечується логічно завершений, послідовний і зручний сценарій користування.

Клієнтська частина відповідає за відображення інформації, збір вхідних даних від користувача, навігацію між сторінками, а також ініціацію запитів до серверної логіки. Реалізовано функції додавання міст, перегляду погодних умов, генерації порад щодо одягу та планування майбутніх поїздок. Для кожного з цих сценаріїв розроблено відповідні компоненти, які забезпечують зручну взаємодію через адаптивний інтерфейс.

Серверна частина реалізована як набір хмарних функцій, які відповідають за інтеграцію з зовнішніми API, генерацію рекомендацій за допомогою мовної моделі та взаємодію з Firestore. Такий підхід дозволив зменшити інфраструктурні витрати, спростити логіку підтримки та забезпечити масштабованість. Завдяки розподілу відповідальностей між клієнтом і сервером досягнуто ефективності та стабільності роботи додатку.

3.2.1 Реалізація серверної частини

Серверна частина вебдодатку виконує роль посередника між клієнтом, зовнішніми сервісами та базою даних Firestore. Усі основні бізнес-процеси, пов’язані з обробкою запитів, виконуються на рівні хмарних функцій, розгорнутих у середовищі Google Cloud Functions. Такий підхід дозволив забезпечити масштабованість, простоту підтримки та швидке реагування системи без потреби в розгортанні окремого сервера.

Кожна функція має чітко визначену відповідальність: обробка авторизації, робота з API погоди та геоданих, створення або оновлення користувацьких даних, а також генерація рекомендацій за допомогою OpenAI. Вся логіка побудована на використанні асинхронних запитів, що дозволяє ефективно взаємодіяти з мережею та зменшити затримки у відповіді. Усі функції реалізовано з урахуванням валідації вхідних даних, обробки помилок та гарантування цілісності інформації.

Для ініціалізації роботи з базою даних Firestore спершу викликається admin.initializeApp(), після чого створюється об'єкт, що забезпечує доступ до колекцій у сховищі. Далі визначається обробник HTTP-запиту, який створює нового користувача після перевірки наявності такого запису у базі. Цей код поданий у лістингу 3.1 та демонструє типову логіку перевірки унікальності запису та додавання нового документа до колекції users.

Лістинг 3.1 – Програмний код, що створює нового користувача в базі Firestore на основі його ідентифікатора

```
admin.initializeApp();
const firestore = admin.firestore();
http("createUser", async (req, res) => {
  const auth0Id = req.body.auth0Id;
  if (!auth0Id) return res.status(400).send("Invalid data");
  const existing = await firestore
    .collection("users")
    .where("auth0Id", "==", auth0Id)
    .get();
  if (!existing.empty) return res.status(409).send("User already exists");
  const userRef = await firestore.collection("users").add({
    auth0Id,
    cities: [],
    trips: [],
  });
  const user = await userRef.get();
  res.status(200).send({ id: user.id, ...user.data() });
});
```

Функція getCities відповідає за пошук міст за назвою, яку вводить користувач у клієнтському інтерфейсі. На початку виконується обробка CORS-запитів методом OPTIONS, що необхідно для взаємодії клієнта та сервера у браузері. Далі перевіряється, чи параметр search є рядком, інакше запит вважається некоректним. Після валідації формується HTTP-запит до зовнішнього сервісу GeoDB API з використанням namePrefix для отримання списку відповідних міст. Цей код поданий у лістингу 3.2 і демонструє повний цикл отримання даних з API та передачу їх у відповідь клієнту.

Лістинг 3.2 – Програмний код, що виконує пошук міст через GeoDB API та повертає список результатів

```
http("getCities", async (req, res) => {
  if (req.method === "OPTIONS") {
    res.set("Access-Control-Allow-Origin",
"http://localhost:8080");
    res.set("Access-Control-Allow-Methods", "GET");
    res.set("Access-Control-Allow-Headers", "Content-Type,
Authorization");
    res.set("Access-Control-Max-Age", "3600");
    res.status(204).send("");
    return;
  }

  const search = req.query.search;
  if (typeof search !== "string") return
res.status(400).send("Invalid data");

  const response = await fetch(
    `https://wft-geo-
db.p.rapidapi.com/v1/geo/cities?namePrefix=${search}`,
    {
      headers: {
        "X-RapidAPI-Key": "API_KEY",
        "X-RapidAPI-Host": "wft-geo-db.p.rapidapi.com",
      },
    }
  );

  const body = await response.json();
  res.send(body.data ?? []);
});
```

Далі було реалізовано додавання міста до списку користувача за його унікальним ідентифікатором auth0Id. Дані про місто отримуються через

запит до GeoDB API, після чого місто додається до масиву cities у відповідному документі користувача. Цей код поданий у лістингу 3.3.

Лістинг 3.3 – Програмний код, що додає місто до списку користувача

```
http("addCityToUser", async (req, res) => {
  if (req.method === "OPTIONS") {
    res.set("Access-Control-Allow-Origin")
    res.set("Access-Control-Allow-Methods", "POST");
    res.set("Access-Control-Allow-Headers", "Content-Type,
Authorization");
    res.set("Access-Control-Max-Age", "3600");
    res.status(204).send("");
    return;}
  const auth0Id = JSON.parse(Buffer.from(req.headers["x-
apigateway-api-userinfo"], "base64").toString()).sub;
  const cityId = Number(req.body.cityId);
  if (!auth0Id || !cityId) return
res.status(400).send("Invalid data");
  const cityData = (await fetch(`https://wft-geo-
db.p.rapidapi.com/v1/geo/cities/${cityId}`, {
    headers: {
      "X-RapidAPI-Key": "API_KEY",
      "X-RapidAPI-Host": "wft-geo-db.p.rapidapi.com",},
  })).json().then(res => res.data);
  const userRef = await
firestore.collection("users").where("auth0Id", "==",
auth0Id).limit(1).get();
  if (userRef.empty) return res.status(404).send("User not
found");
  const docRef = userRef.docs[0].ref;
  await docRef.update({ cities:
admin.firestore.FieldValue.arrayUnion(cityData) });
  const user = await docRef.get();
  res.status(200).send({ id: user.id, ...user.data() });});
```

Далі була реалізація отримання списку міст, які були додані користувачем до свого профілю. Пошук здійснюється за ідентифікатором, після чого повертається масив cities з документа користувача. Код поданий у лістингу 3.4.

Лістинг 3.4 – Програмний код, що повертає список міст, доданих користувачем

```
http("getUserCities", async (req, res) => {
  if (req.method === "OPTIONS") {
    res.set("Access-Control-Allow-Origin",
"http://localhost:8080");
    res.set("Access-Control-Allow-Methods", "GET");
    res.set("Access-Control-Allow-Headers", "Content-Type,
Authorization");
    res.set("Access-Control-Max-Age", "3600");
    res.status(204).send("");
    return;}
  const auth0Id = JSON.parse(Buffer.from(req.headers["x-
apigateway-api-userinfo"], "base64").toString()).sub;
  if (!auth0Id) return res.status(400).send("Invalid
data");
  const snapshot = await
firestore.collection("users").where("auth0Id", "==",
auth0Id).limit(1).get();
  if (snapshot.empty) return res.status(404).send("User not
found");
  const user = snapshot.docs[0].data();
  res.status(200).send(user.cities);
});
```

Ця функція виконує запит до GeoDB API для отримання детальної інформації про місто за його ідентифікатором. Після валідації вхідного параметра cityId, надсилається HTTP-запит до зовнішнього API і у відповідь

повертаються дані про місто. Цей код поданий у лістингу 3.5 та реалізує механізм отримання розширених характеристик населеного пункту.

Лістинг 3.5 – Програмний код, що отримує детальну інформацію про місто за його ідентифікатором, використовуючи GeoDB API

```
http("getCityDetails", async (req, res) => {
  if (req.method === "OPTIONS") {
    res.set("Access-Control-Allow-Origin",
"http://localhost:8080");
    res.set("Access-Control-Allow-Methods", "GET");
    res.set("Access-Control-Allow-Headers", "Content-Type,
Authorization");
    res.set("Access-Control-Max-Age", "3600");
    res.status(204).send("");
    return;
  }
  const cityId = req.query.cityId;
  if (!cityId) return res.status(400).send("Invalid data");
  const response = await fetch(`https://wft-geo-
db.p.rapidapi.com/v1/geo/cities/${cityId}`, {
    headers: {
      "X-RapidAPI-Key": "API_KEY",
      "X-RapidAPI-Host": "wft-geo-db.p.rapidapi.com",
    },
  });
  const body = await response.json();
  res.send(body.data ?? []);
});
```

Наступна функція призначена для отримання погодного прогнозу на основі координат, кількості днів або діапазону дат. Вона формує запит до Open-Meteo API, де параметри включають погодні показники на поточний, щоденний та погодинний рівні. Цей код поданий у лістингу 3.6.

Лістинг 3.6 – Програмний код, що отримує прогноз погоди за координатами, кількістю днів або діапазоном дат через Open-Meteo API

```
    http("getWeather", async (req, res) => {
      if (req.method === "OPTIONS") {
        res.set("Access-Control-Allow-Origin",
"http://localhost:8080");
        res.set("Access-Control-Allow-Methods", "GET");
        res.set("Access-Control-Allow-Headers", "Content-Type,
Authorization");
        res.set("Access-Control-Max-Age", "3600");
        res.status(204).send("");
        return;}
      const lon = Number(req.query.lon);
      const lat = Number(req.query.lat);
      const days = req.query.forecastDays ?
Number(req.query.forecastDays) : undefined;
      const start = req.query.startDate ? new
Date(req.query.startDate) : undefined;
      const end = req.query.endDate ? new
Date(req.query.endDate) : undefined;

      const valid = !isNaN(lat) && !isNaN(lon) && (days ||
(start && end && start < end));
      if (!valid) return res.status(400).send("Invalid
parameters");
      const params = new URLSearchParams({
        latitude: lat.toString(),
        longitude: lon.toString(),
        hourly:
"temperature_2m,relative_humidity_2m,apparent_temperature,prec
ipitation,rain,cloud_cover,wind_speed_10m",
        daily:
"temperature_2m_max,temperature_2m_min,apparent_temperature_ma
x,apparent_temperature_min,precipitation_sum,rain_sum,cloud_co
ver_mean,wind_speed_10m_max",
```

Продовження лістингу 3.6

```
        current:
"temperature_2m,relative_humidity_2m,apparent_temperature,prec
ipitation,rain,cloud_cover,wind_speed_10m"});
      if (days) params.set("forecast_days", days.toString());
      if (start && end) {
        params.set("start_date",
start.toISOString().split("T")[0]);
        params.set("end_date",
end.toISOString().split("T")[0]);}
      const resData = await fetch(`https://api.open-
meteo.com/v1/forecast?${params}`);
      const body = await resData.json();
      res.send(mapToWeatherForecast(body));
    });
```

Далі було реалізовано код, який відповідає за генерацію рекомендацій щодо одягу на основі поточних погодних умов і типу активності користувача. На початку запит проходить валідацію через схему, що перевіряє наявність і коректність параметрів, таких як температура, вологість, кількість опадів, швидкість вітру тощо. Якщо дані валідні, формується текстовий запит до моделі GPT, у якому зазначаються погодні показники й тип активності (наприклад, прогулянка або пробіжка).

Далі функція виконує запит до OpenAI API з відповідно згенерованим промптом, який інтерпретується як запит на отримання порад. Модель штучного інтелекту у відповідь повертає структурований текст, який містить список предметів одягу з короткими поясненнями щодо їх доцільності в заданих умовах. Ці поради обробляються і форматуються у вигляді масиву об’єктів. Цей код поданий у лістингу 3.7 та реалізує ключову функцію додатку – персоналізоване формування порад на основі погодних умов у реальному часі.

Лістинг 3.7 – Програмний код, що на основі погодних параметрів надсилає запит до OpenAI API і повертає список рекомендацій щодо одягу

```
http("getOutfitSuggestions", async (req, res) => {
  if (req.method === "OPTIONS") {
    res.set("Access-Control-Allow-Origin",
"http://localhost:8080");
    res.set("Access-Control-Allow-Methods", "GET");
    res.set("Access-Control-Allow-Headers", "Content-Type,
Authorization");
    res.set("Access-Control-Max-Age", "3600");
    res.status(204).send("");
    return;
  }
  const schema = object({
    temperature: coerce.number(),
    relativeHumidity: coerce.number().min(0).max(100),
    apparentTemperature: coerce.number(),
    precipitation: coerce.number(),
    rain: coerce.number().min(0),
    cloudCover: coerce.number().min(0).max(100),
    windSpeed: coerce.number().min(0),
    activityType: nativeEnum(ActivityType),
  });
  const { success, data, error } =
schema.safeParse(req.query);
  if (!success) return res.status(400).send(error);
  const apiKey = "API_KEY";
  const prompt = `In the place where I live there's
${data.temperature} degrees Celsius temperature, ${data.rain}mm
rain, relative humidity ${data.relativeHumidity}%,
${data.cloudCover}% cloud cover and ${data.windSpeed}km/h wind
speed. What should I wear today for a
${activityTypeNames[data.activityType]}?
    Return only the list of items. Exclude items for safety
like helmets and items.
```

Продовження лістингу 3.7

```
For each item return a brief reasoning separated by ---. Do not include index numbers. Do not return title.`;
      const response = await fetch("https://api.openai.com/v1/chat/completions", {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
          Authorization: `Bearer ${apiKey}`,},
        body: JSON.stringify({
          model: "gpt-4o",
          messages: [{ role: "system", content: "You are a helpful assistant." }, { role: "user", content: prompt }],}),});
      const result = await response.json();
      const suggestion = result.choices[0].message.content
        .split("\n")
        .map(x => {
          const [item, reason] = x.split(" --- ");
          return { item: item.replaceAll("- ", ""), reason };
        });
      res.send(suggestion);});
```

Наступний код реалізує видалення міста з персонального списку користувача на основі його ідентифікатора auth0Id. Після валідації вхідних параметрів здійснюється запит до GeoDB API для отримання повних даних про місто, що підлягає видаленню. Це необхідно для того, щоб виключення відбувалося не за id, а за об'єктом, ідентичним тому, що зберігається у базі. Отримані дані про місто використовуються як аргумент функції arrayRemove для оновлення документа користувача.

Далі функція знаходить відповідний документ користувача в базі Firestore та оновлює його, видаляючи вказане місто з масиву cities. Цей код поданий у лістингу 3.8.

Лістинг 3.8 – Програмний код, що видаляє конкретне місто з масиву міст користувача

```
http("removeCityFromUser", async (req, res) => {
  if (req.method === "OPTIONS") {
    res.set("Access-Control-Allow-Origin")
    res.set("Access-Control-Allow-Methods", "POST");
    res.set("Access-Control-Allow-Headers", "Content-Type,
Authorization");
    res.set("Access-Control-Max-Age", "3600");
    res.status(204).send("");
    return;}
  const auth0Id = JSON.parse(Buffer.from(req.headers["x-
apigateway-api-userinfo"], "base64").toString()).sub;
  const cityId = Number(req.body.cityId);
  if (!auth0Id || !cityId) return
res.status(400).send("Invalid data");
  const cityData = (await fetch(`https://wft-geo-
db.p.rapidapi.com/v1/geo/cities/${cityId}`,
  const userRef = await
firestore.collection("users").where("auth0Id", "==",
auth0Id).limit(1).get();
  if (userRef.empty) return res.status(404).send("User not
found");
  const docRef = userRef.docs[0].ref;
  await docRef.update({ cities:
admin.firestore.FieldValue.arrayRemove(cityData) });
  const user = await docRef.get();
  res.status(200).send({ id: user.id, ...user.data() });});
```

Наступний код реалізує створення нової поїздки користувача з вказанням міста, дат і назви маршруту. Після перевірки авторизації та вхідних даних, надсилається запит до GeoDB API для отримання інформації про місто, обране для подорожі. Цей код поданий у лістингу 3.9.

Лістинг 3.9 – Програмний код, що створює поїздку для користувача

```
http("createTrip", async (req, res) => {
  if (req.method === "OPTIONS") {
    res.set("Access-Control-Allow-Origin")
    res.set("Access-Control-Allow-Methods", "POST");
    res.set("Access-Control-Allow-Headers", "Content-Type,
Authorization");
    res.set("Access-Control-Max-Age", "3600");
    res.status(204).send("");
    return;}
  const auth0Id = JSON.parse(Buffer.from(req.headers["x-
apigateway-api-userinfo"], "base64").toString()).sub;
  if (!auth0Id) return
res.status(401).send("Unauthorized");
  const tripData = tripSchema.safeParse(req.body);
  if (!tripData.success) return
res.status(400).send("Invalid data");
  const cityData = (await fetch(
    `https://wft-geo-
db.p.rapidapi.com/v1/geo/cities/${tripData.data.cityId}`,
  const userRef = await
firestore.collection("users").where("auth0Id", "==",
auth0Id).limit(1).get();
  if (userRef.empty) return res.status(404).send("User not
found");
  const trip = {
    id: uuidv4(),
    city: cityData,
    name: tripData.data.name,
    startDate: tripData.data.startDate,
    endDate: tripData.data.endDate,};
  await userRef.docs[0].ref.update({
    trips: admin.firestore.FieldValue.arrayUnion(trip),});
  res.status(200).send(trip);});
```

Далі можна побачити код, що дозволяє отримати список поїздок, які користувач раніше зберіг у своєму профілі. Після авторизації за допомогою auth0Id дані повертаються з поля trips документа користувача. Цей код поданий у лістингу 3.10.

Лістинг 3.10 – Програмний код, що повертає список запланованих поїздок користувача

```
http("getUserTrips", async (req, res) => {
  if (req.method === "OPTIONS") {
    res.set("Access-Control-Allow-Origin")
    res.set("Access-Control-Allow-Methods", "GET");
    res.set("Access-Control-Allow-Headers", "Content-Type,
Authorization");
    res.set("Access-Control-Max-Age", "3600");
    res.status(204).send("");
    return;}
  const auth0Id = JSON.parse(Buffer.from(req.headers["x-
apigateway-api-userinfo"], "base64").toString()).sub;
  if (!auth0Id) return res.status(400).send("Invalid
data");
  const userSnapshot = await
firestore.collection("users").where("auth0Id", "==",
auth0Id).limit(1).get();
  if (userSnapshot.empty) return
res.status(404).send("User not found");
  const user = userSnapshot.docs[0].data();
  res.status(200).send(user.trips);});
```

Цей код реалізує функцію генерації рекомендацій щодо одягу на заплановану поїздку з урахуванням погодних умов на кожен день. Перед формуванням запиту до OpenAI API відбувається валідація вхідних даних, яка перевіряє прогноз по кожному дню подорожі – зокрема температуру, опади, хмарність і швидкість вітру. Якщо дані валідні, формується

текстовий запит, у якому вказуються дати поїздки та погодні параметри, що дають контекст для генерації.

Далі GPT-модель обробляє запит і повертає структуровану відповідь у форматі JSON, де кожен елемент містить назву предмета одягу та пояснення, чому його слід взяти. Відповідь парситься і надсилається клієнту для відображення у зручному вигляді. Цей код поданий у лістингу 3.11 та є ключовим у реалізації функціоналу прогнозованих рекомендацій на основі багатоденного погодного звіту.

Лістинг 3.11 – Програмний код, що на основі багатоденного прогнозу погоди формує перелік речей, які слід взяти в поїздку

```
http("getTripOutfitSuggestions", async (req, res) => {
  if (req.method === "OPTIONS") {
    res.set("Access-Control-Allow-Origin")
    res.set("Access-Control-Allow-Methods", "GET");
    res.set("Access-Control-Allow-Headers", "Content-Type,
Authorization");
    res.set("Access-Control-Max-Age", "3600");
    res.status(204).send("");
    return;}
  const validation = object({
    weather: object({
      daily: array(
        object({
          time: string(),
          temperatureMax: coerce.number(),
          temperatureMin: coerce.number(),
          apparentTemperatureMax: coerce.number(),
          apparentTemperatureMin: coerce.number(),
          precipitationSum: coerce.number(),
          rainSum: coerce.number(),
          cloudCoverMean: coerce.number(),
          windSpeedMax: coerce.number(),})),}),
    startDate: string(),
```

Продовження лістингу 3.11

```
        endDate: string(),}).safeParse(req.body);
      if (!validation.success) return
res.status(400).send(validation.error);
      const { weather, startDate, endDate } = validation.data;
      const apiKey = "API_KEY";
      const prompt = `I'm going on a trip to another city. I'll
be there from ${startDate} to ${endDate}.
    Here's the forecast:
    ${weather.daily.map(
      (d) => `Date: ${d.time}, Temp: ${d.temperatureMax}-
${d.temperatureMin}°C, Apparent: ${d.apparentTemperatureMax}-
${d.apparentTemperatureMin}°C, Rain: ${d.rainSum}mm, Cloud:
${d.cloudCoverMean}%, Wind: ${d.windSpeedMax}km/h`).join("\n")}
    Return only JSON with items and reasons like { items: [{
name: string, reason: string }] }`;
      const response = await
fetch("https://api.openai.com/v1/chat/completions", {
        method: "POST",
        body: JSON.stringify({
          model: "gpt-4o",
          messages: [{ role: "system", content: "You are a
helpful assistant." }, { role: "user", content: prompt }],}),});
      const raw = await response.json();
      const clean =
raw.choices[0].message.content.replace(/^```json\n/,
'').replace(/\n```$/, '');
      const parsed = JSON.parse(clean);
      res.send(parsed);
    });
```

Таким чином було детально розглянуто реалізацію серверної частини вебдодатку, яка забезпечує взаємодію з базою даних, зовнішніми API та мовною моделлю. Усі функції реалізовано як хмарні, що дозволяє досягти гнучкості, масштабованості та простоти розгортання. Логіка кожної з них

чітко визначена й охоплює створення користувача, керування містами, генерацію рекомендацій та планування поїздок, що в сукупності формує повний функціональний бекенд системи.

3.2.2 Реалізація клієнтської частини

У цьому підрозділі розглядається реалізація клієнтської частини додатку, яка забезпечує взаємодію користувача з інтерфейсом та відправлення запитів до серверної логіки. Особливу увагу приділено компонентам, що відповідають за пошук міст, перегляд метеоінформації, створення поїздок і генерацію рекомендацій щодо одягу.

Клієнтська частина написана з використанням React та TypeScript і використовує бібліотеку RTK Query для зручного керування асинхронними запитами. Зв’язок між інтерфейсом і серверними функціями здійснюється через єдиний модуль API, що дозволяє централізовано описати всі ендпоінти та трансформації відповідей.

Наступний код поданий у лістингу 3.12 та реалізує сторінку пошуку міст, які користувач може додати до свого списку. У першому рядку створюється стан для введеного користувачем пошукового запиту, що оновлюється під час набору тексту. Для оптимізації запитів до API використовується debounce-версія цього запиту з затримкою в 500 мілісекунд. Далі через хук useGetCitiesQuery здійснюється запит до відповідного ендпоінту для отримання переліку міст за заданим рядком.

Якщо результат завантажується, виводиться компонент Loader, а у разі помилки – повідомлення з текстом про помилку. У випадку успішного отримання даних кожне місто відображається у вигляді картки з кнопкою додавання. При натисканні на картку міста виконується навігація до сторінки з його деталями. Таким чином, цей компонент забезпечує базову функціональність інтерфейсу пошуку міст у вебдодатку.

Лістинг 3.12 – Програмний код, що відповідає за пошук міст та відображення результатів у вигляді списку

```
export const SearchCitiesPage = () => {
  const [searchQuery, setSearchQuery] = useState("");
  const [debouncedSearchQuery] = useDebouncedValue(searchQuery, 500);
  const { isLoading, isError, data: cities } = useGetCitiesQuery(debouncedSearchQuery, {
    skip: !debouncedSearchQuery,});
  const navigate = useNavigate();
  return (<>
      <PageHeader title="Search Cities" />
      <PageBody>
        <TextInput value={searchQuery} onChange={(e) => setSearchQuery(e.target.value)} />
        {isLoading && <Loader />}
        {isError && <Text>Error loading cities</Text>}
        {cities?.map((city) => (
          <Paper key={city.id} onClick={() => navigate(`/cities/${city.id}`)}>
            <Text>
{city.name}, {city.country} - Population: {city.population}
            </Text>
            <Button>Add</Button>
          </Paper>
        ))}
      </PageBody>
    </>
  );
};
```

Цей код поданий у лістингу 3.13 та реалізує інтерфейс для перегляду міст, які були додані користувачем. На початку відбувається отримання списку міст через запит до серверної частини за допомогою

useGetUserCitiesQuery. У разі відсутності даних або помилки виводиться відповідне повідомлення, а у випадку успішного завантаження – перелік міст або інформація про його порожність. Також передбачена кнопка, яка дозволяє перейти до сторінки додавання нового міста.

Лістинг 3.13 – Програмний код, що відображає список доданих користувачем міст

```
export const MyCitiesPage = () => {
  const { data: userCities, isLoading, isError } = useGetUserCitiesQuery();
  const navigate = useNavigate();
  return (<>
      <PageHeader title="My Cities" />
      <PageBody>
        {isLoading ? (
          <LoadingOverlay visible={true} />
        ) : isError || !userCities ? (
          <Text>Error loading cities</Text>
        ) : (
          <Stack>
            <Button onClick={() => navigate("/cities/search")}>Add City</Button>
            {userCities.length === 0 ? (
              <Text>No cities added</Text>) : (
              userCities.map((city) => (
                <Paper key={city.id} onClick={() => navigate(`/cities/${city.id}`)}>
                  <Text>{city.name}, {city.country} - Population: {city.population}</Text>
                </Paper>))}
          </Stack>)}
      </PageBody></>
);};
```

Цей код поданий у лістингу 3.14 та відповідає за відображення детальної сторінки обраного міста. Він здійснює запити на отримання інформації про саме місто, погодний прогноз та список доданих користувачем міст. На основі цих даних формується інтерфейс з кнопками додавання або видалення міста, а також відображенням погодних показників та порад. Користувач також має можливість перейти до генерації рекомендацій щодо одягу через відповідну кнопку.

Лістинг 3.14 – Програмний код, що відображає погодну інформацію для обраного міста

```
export const CityPage = () => {
  const { cityId } = useParams();
  const navigate = useNavigate();
  const { data: city } = useGetCityDetailsQuery(Number(cityId));
  const { data: userCities } = useGetUserCitiesQuery();
  const { data: forecast } = useGetCurrentWeatherQuery({
    lon: city?.longitude ?? 0,
    lat: city?.latitude ?? 0,
    forecast: 10,
  }, { skip: !city });
  return (<>
      <PageHeader
        title={city?.name ?? "City"}
leftSection={<BackButton onClick={() => navigate(-1)} />}
rightSection={userCities?.some((c) => c.id === city?.id) ?
  <ActionIcon><IconTrash /></ActionIcon>) : (
  <ActionIcon><IconPlus /></ActionIcon>)}/>
      <PageBody>
        <Stack>
          <Button onClick={() => navigate(`/cities/${cityId}/outfit-suggestions`)}>
            What to wear?
          </Button>
```

Продовження лістингу 3.14

```
              <HourlyTemperatures
temperatures={forecast?.hourly ?? []} />
              <Forecast forecast={forecast?.daily ?? []} />
              <SimpleGrid cols={2}>
                <WeatherItem  title="Feels  like"  value="..."
icon={<IconTemperature />} />
                <WeatherItem   title="Humidity"   value="..."
icon={<IconDroplets />} />
                <WeatherItem     title="Wind"     value="..."
icon={<IconWind />} />
                <WeatherItem  title="Visibility"  value="..."
icon={<IconEye />} />
              </SimpleGrid>
            </Stack>
          </PageBody>
        </>
      );
    };
```

Цей код поданий у лістингу 3.15 та відповідає за візуалізацію температурного прогнозу у вигляді горизонтального індикатора прогресу.

На основі переданого масиву прогнозів для кожного дня визначаються глобальні мінімальні та максимальні температури, що дозволяє нормалізувати шкалу відображення. Для кожного дня обчислюється відповідна довжина кольорового інтервалу температури та будується окрема секція на графіку.

Це дозволяє візуально оцінити діапазон коливань температури у порівнянні з іншими днями. Таке представлення сприяє кращому сприйняттю даних користувачем. Всі елементи розміщено у компоненті Paper, що стилістично вирізняє цей блок серед інших.

Лістинг 3.15 – Програмний код, що візуалізує температурний прогноз на кілька днів, відображаючи діапазон температур

```
export const Forecast = ({ forecast }) => {
  const globalMin = Math.min(...forecast.map(f => f.minTemperature));
  const globalMax = Math.max(...forecast.map(f => f.maxTemperature));
  const range = globalMax - globalMin;
  return (
    <Paper>
      <Text>{forecast.length}-day forecast</Text>
      <Stack>
{forecast.map(({ day, minTemperature, maxTemperature }) => {
  const start = ((minTemperature - globalMin) / range) * 100;
  const end = ((globalMax - maxTemperature) / range) * 100;
          return (
            <Flex>
              <Text>{day}</Text>
              <Flex>
                <Text>{minTemperature}°C</Text>
                <Progress.Root>
                  <Progress.Section value={start} />
                  <Progress.Section value={100-end-start/>
                  <Progress.Section value={end} />
                </Progress.Root>
                <Text>{maxTemperature}°C</Text>
              </Flex>
            </Flex>);})}
      </Stack>
    </Paper>);};
```

Цей код поданий у лістингу 3.16 та відповідає за відображення інформації про окрему поїздку користувача. Функціональність полягає у завантаженні збережених поїздок, пошуку потрібної за її ідентифікатором

та отриманні метеорологічного прогнозу для вказаного періоду. У разі успішного отримання даних компонент TripView відповідає за відображення детальної інформації. Якщо ж сталася помилка або дані ще завантажуються, інтерфейс показує повідомлення про помилку або індикатор завантаження.

Лістинг 3.16 – Програмний код, що показує інформацію про окрему поїздку користувача та прогнозом погоди на відповідні дати

```
export const TripPage = () => {
  const { tripId } = useParams();
  const { data: trips, isLoading, isError } = useGetUserTripsQuery();
  const trip = trips?.find((t) => t.id === tripId) ?? null;
  const { data: forecast, isLoading: isWeatherLoading, isError: isWeatherError } =
    useGetCurrentWeatherQuery( trip? {
            lon: trip.city.longitude,
            lat: trip.city.latitude,
            startDate: trip.startDate,
            endDate: trip.endDate,} : {
            lon: 0,
            lat: 0,
            startDate: new Date(),
            endDate: new Date(),
          }, { skip: !trip });
return ( <PageBody>
      {isLoading || isWeatherLoading ? (
        <LoadingOverlay visible />
) : isError || isWeatherError || !trip || !forecast ? (
<Notification title="Bummer!">Couldn't load trips</Notification> ) : (
        <TripView trip={trip} forecast={forecast} />)}
    </PageBody>);};
```

Далі була реалізація генерації й відображення рекомендацій щодо одягу для поїздки користувача. Запит до API здійснюється на основі метеорологічного прогнозу і дат подорожі. Користувач бачить список порад у форматі акордеону, де для кожного елемента зазначено причину вибору. Цей код поданий у лістингу 3.17.

Лістинг 3.17 – Програмний код, що виводить рекомендації щодо одягу для всієї запланованої поїздки на основі прогнозу погоди

```
export const TripOutfitSuggestions = ({ forecast, startDate, endDate }) => {
  const { data, isLoading, isError } = useGetTripOutfitSuggestionsQuery({
    weather: forecast,
    startDate,
    endDate,});
  if (isLoading) return <Skeleton h="3rem" />;
  if (isError || !data) return <Notification title="Bummer!">Couldn't load outfit suggestions</Notification>;
  return (
    <Accordion>
      {data.map((suggestion, index) => (
        <Accordion.Item key={index} value={String(index)}>
 <Accordion.Control>{suggestion.item}</Accordion.Control>
<Accordion.Panel>{suggestion.reason}</Accordion.Panel>
        </Accordion.Item>))}
    </Accordion>);};
```

Через функцію createApi задається базова адреса, авторизаційний заголовок і перелік усіх запитів, включаючи отримання міст, прогнозу погоди, користувацьких поїздок та рекомендацій. Для кожного ендпоінту вказано тип запиту, HTTP-метод, необхідні параметри або тіло запиту. Це дозволяє забезпечити централізовану та типізовану взаємодію з бекендом у

всіх компонентах додатку. Цей код поданий у лістингу 3.18 та визначає конфігурацію клієнта RTK Query для взаємодії з API вебдодатку.

Лістинг 3.18 – Програмний код, що формує єдину точку доступу до API-запитів на стороні клієнта з використанням RTK Query

```
export const api = createApi({
  reducerPath: "api",
  baseQuery: fetchBaseQuery({
    baseUrl: env.VITE_API_URL,
    prepareHeaders: (headers, { getState }) => {
const token = (getState() as RootState).auth.accessToken;
if (token) headers.set("authorization", `Bearer ${token}`);
      return headers;},}),
  endpoints: (builder) => ({
getCities: builder.query({ query: (search) => ({ url: "getCities", params: { search } }) }),
getCityDetails: builder.query({ query: (id) => ({ url: "getCityDetails", params: { cityId: id } }) }),
getCurrentWeather: builder.query({ query: ({ lon, lat }) => ({ url: "getCurrentWeather", params: { lon, lat } }) }),
addCityToUser: builder.mutation({ query: (cityId) => ({ url: "addCityToUser", method: "POST", body: { cityId } }) }),
removeCityFromUser: builder.mutation({ query: (cityId) => ({ url: "removeCityFromUser", method: "POST", body: {cityId}}) }),
getUserCities: builder.query({ query: () => ({ url: "getUserCities" }) }),
getOutfitSuggestions: builder.query({ query: (params) => ({ url: "getOutfitSuggestions", params }) }),
getTripOutfitSuggestions: builder.query({ query: (body) => ({ url: "getTripOutfitSuggestions", method: "POST", body }) }),
createTrip: builder.mutation({ query: (trip) => ({ url: "createTrip", method: "POST", body: trip }) }),
getUserTrips: builder.query({ query: () => ({ url: "getUserTrips" }) }),}),});
```

Реалізація вебдодатку демонструє повноцінну інтеграцію клієнтської частини з серверною інфраструктурою, побудованою на базі сервер-функцій Google Cloud. На рівні фронтенду було використано сучасний стек технологій – React, Mantine UI та Redux Toolkit Query – що забезпечило зручну побудову інтерфейсів, швидкий обмін даними з API і хорошу масштабованість архітектури. Основна логіка поділена на окремі компоненти, які відповідають за пошук міст, відображення прогнозу погоди, створення та перегляд поїздок, а також за генерацію персоналізованих порад користувачу.

Серверна частина проекту реалізована у вигляді окремих обробників, що відповідають за конкретні дії, зокрема роботу з користувачами, містами, поїздками та інтеграцію з зовнішніми сервісами – Open Meteo та OpenAI. Уся інформація зберігається у базі даних Google Firestore у вигляді документів, що дозволяє швидко взаємодіяти з даними у режимі реального часу. Завдяки чіткій структурі, підтримці авторизації через Auth0 та ефективному механізму кешування запитів у RTK Query, додаток забезпечує надійну, адаптивну і водночас продуктивну роботу для кінцевого користувача.

3.3 Інтерфейс додатку

Інтерфейс користувача є ключовим елементом вебдодатку, оскільки саме через нього відбувається основна взаємодія користувача із системою. Для розробки інтерфейсу було використано бібліотеку компонентів Mantine, що забезпечила сучасний вигляд, зручність в адаптивному відображенні та підтримку темної теми оформлення. Усі сторінки спроектовано з урахуванням принципів UX-дизайну: акценти розставлені на простоті навігації, зрозумілих діях і логічному розміщенні елементів.

На початковому екрані вебдодатку користувача зустрічає мінімалістичний інтерфейс із логотипом, назвою системи та кнопкою

авторизації. Такий підхід дозволяє сфокусувати увагу на основній дії – вході до системи. Далі можна побачити оформлення цього екрану (рисунок 3.1), що демонструє використання темної теми інтерфейсу та акцентного кольору для інтерактивних елементів.

Рисунок 3.1 – Сторінка входу

Яскравий колір кнопки входу забезпечує візуальний контраст і сприяє кращому сприйняттю інтерфейсу, особливо на мобільних пристроях.

3.3.1 Головна сторінка та навігація

На сторінці «My Cities» (рисунок 3.2) користувач бачить власний перелік доданих міст або повідомлення про його відсутність. Також видно кнопку для додавання нового міста та нижню панель навігації.

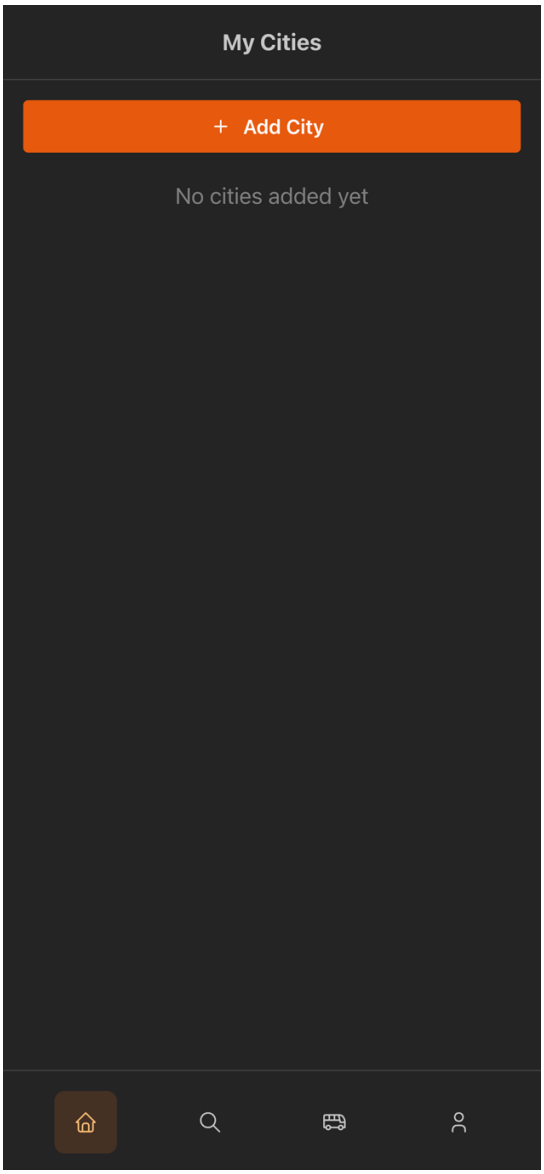

Рисунок 3.2 – Головна сторінка

Сторінка організована таким чином, щоб користувач міг легко орієнтуватися в системі: основні дії винесено у верхню частину екрана, а

вкладки з іконками (рисунок 3.3) внизу забезпечують швидкий перехід до інших розділів.

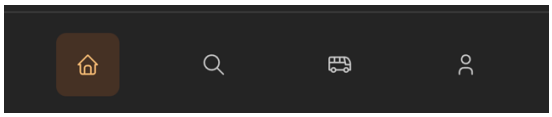

Рисунок 3.3 – Навігаційне меню

У випадку порожнього списку реалізовано відповідне повідомлення, що надає користувачу зрозумілий зворотний зв’язок.

3.3.2 Пошук та додавання міста

На сторінці пошуку міст реалізовано функціональність, що дозволяє користувачеві ввести назву населеного пункту, переглянути результати пошуку та додати обране місто до свого списку. Далі можна побачити візуалізацію цього інтерфейсу (рисунок 3.4) із полем введення, результатом пошуку та кнопкою додавання.

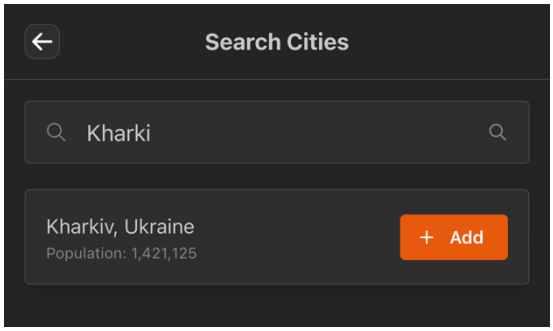

Рисунок 3.4 – Сторінка пошуку

Інтерфейс сторінки міста містить погодні дані у погодинному та денному форматі, а також кнопку переходу до рекомендацій одягу. Далі можна побачити приклад такого інтерфейсу (рисунок 3.5) з візуалізацією

прогнозу на кілька днів, відображенням поточних температур і ключових погодних показників.

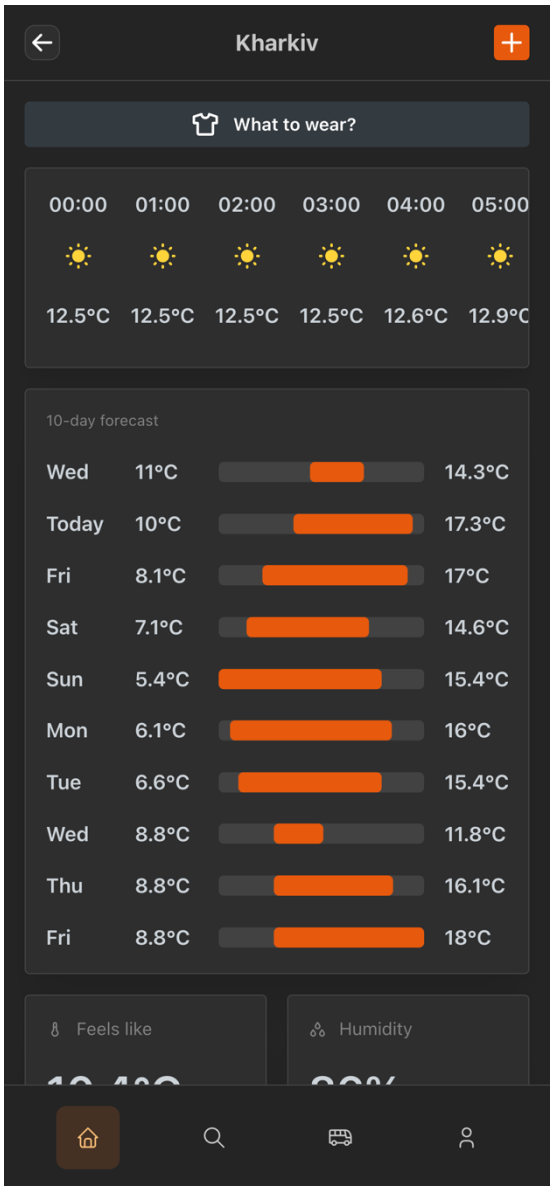

Рисунок 3.5 – Сторінка знайденого міста

Функціонал пошуку та додавання міста забезпечує зручну взаємодію користувача з базою географічних даних через інтуїтивний інтерфейс. Він дозволяє знаходити потрібне місто та додавати його до персонального списку для подальшого отримання прогнозу погоди та рекомендацій.

3.3.3 Відображення метеоінформації та рекомендацій

Далі можна побачити інтерфейс прогнозу на 10 днів, який реалізовано у вигляді вертикального списку з мінімальними та максимальними температурами, представленими графічно (рисунок 3.6). Такий підхід дозволяє швидко порівнювати температурні діапазони між днями та візуально оцінювати зміни погоди протягом тижня.

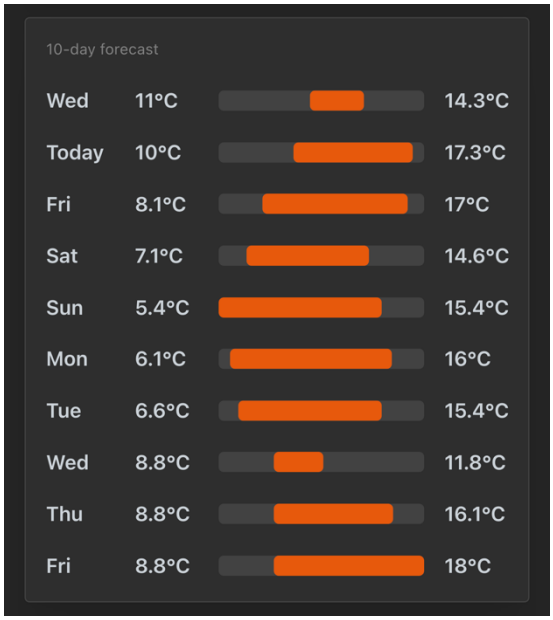

Рисунок 3.6 – Прогноз погоди на 10 днів

Далі можна побачити панель з основними метеорологічними показниками, такими як температура за відчуттями, вологість, швидкість вітру та видимість (рисунок 3.7). Кожен показник представлено у вигляді окремого блоку з коротким описом, що полегшує сприйняття інформації користувачем. Ці блоки супроводжуються відповідними іконками, які надають інтерфейсу інтуїтивно зрозумілого вигляду та покращують візуальне сприйняття. Завдяки структурованому відображенню користувач отримує контекстуальне пояснення для кращого розуміння поточних умов.

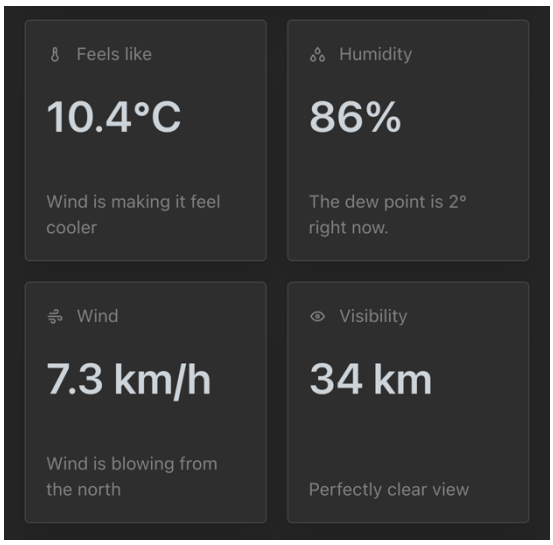

Рисунок 3.7 – Метеоінформація

Далі можна побачити інтерфейс розділу рекомендацій щодо одягу на основі погодних умов і типу запланованої активності (рисунок 3.8). У верхній частині розміщено перемикач між варіантами walk, run і bicycle, що дозволяє користувачу швидко змінювати контекст і отримувати поради, адаптовані до конкретного виду фізичної активності. Ця функціональність реалізована як вкладки, кожна з яких під час взаємодії оновлює вміст відповідно до нового запиту до серверної частини. Таким чином, користувач має змогу отримати актуальну інформацію щодо одягу, що враховує не лише прогноз погоди, а й особливості пересування.

Кожна порада подана у вигляді акордеону – розгортного елементу, який спочатку показує лише назву рекомендованого елементу одягу, наприклад, light jacket or sweater, а після натискання розгортається і демонструє текстове пояснення. У поясненні вказується причина, чому конкретний предмет є доречним, наприклад: «To keep warm in cool temperature and given the high humidity, it may feel cooler.» Таким чином, користувач не лише отримує список речей, які варто вдягнути, але й розуміє логіку формування цієї рекомендації.

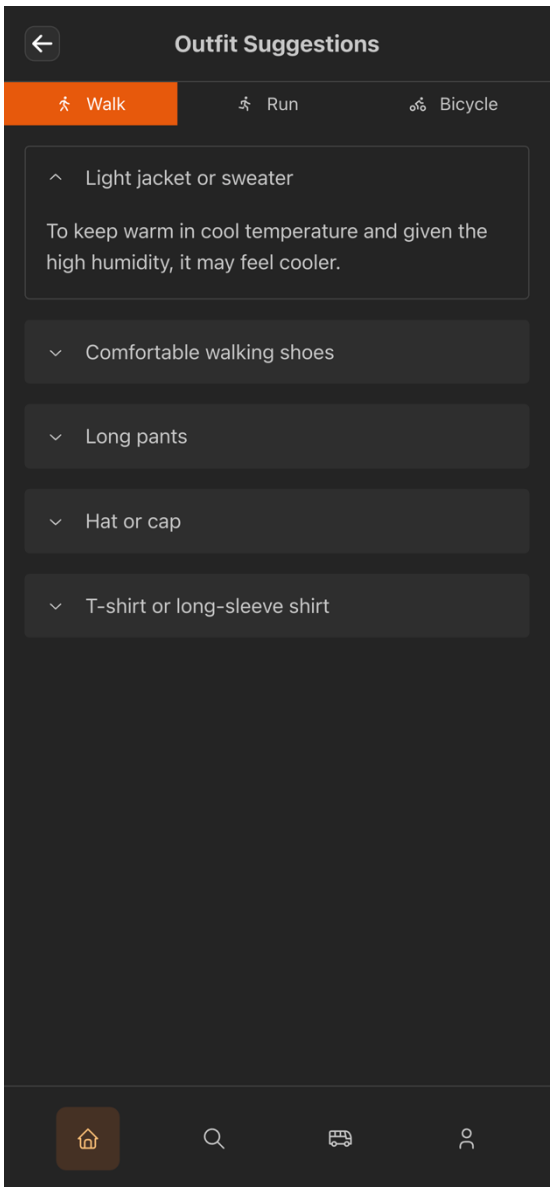

Рисунок 3.8 – Сторінка рекомендацій щодо одягу для прогулянки

Далі можна побачити вкладку з рекомендаціями щодо одягу для пробіжки, сформовану на основі поточних погодних умов (рисунок 3.9). Інтерфейс дозволяє швидко переглянути перелік рекомендованих предметів гардеробу, таких як lightweight long-sleeve shirt, moisture-wicking socks або breathable running shoes, які є оптимальними для активного руху за визначених метеоумов.

Окремі пункти містять розгорнуте пояснення, яке пояснює доцільність використання конкретного елементу. Наприклад, thin gloves

супроводжуються приміткою: optional for added warmth if hands get cold easily with high humidity. Це дозволяє враховувати індивідуальні фізіологічні особливості користувача та формувати більш адаптивні рекомендації, що підвищує точність і довіру до системи.

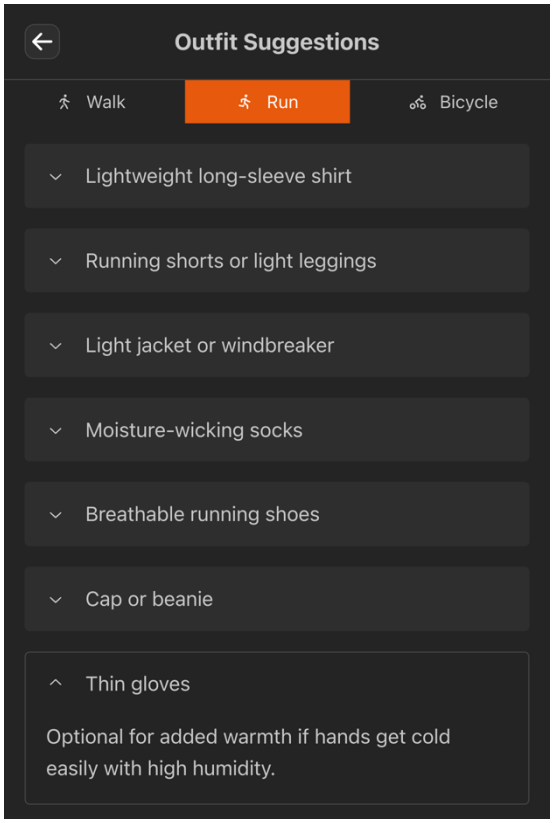

Рисунок 3.9 – Рекомендації щодо одягу для бігу

Далі можна побачити вкладку з рекомендаціями щодо одягу для їзди на велосипеді, яка враховує погодні умови на момент запиту (рисунок 3.10). У переліку представлені базові елементи екіпірування, такі як long-sleeve shirt або cycling tights, що забезпечують комфорт і захист тіла під час руху.

Розгорнуті пояснення окремих пунктів містять згадки про метеофактори. Наприклад, рекомендація щодо windbreaker супроводжується поясненням про доцільність носіння цього елементу за швидкості вітру 7.3 км/год і суцільної хмарності. Такий підхід дає змогу

користувачу отримувати не просто список речей, а аргументовані поради, що ґрунтуються на поточному кліматичному контексті.

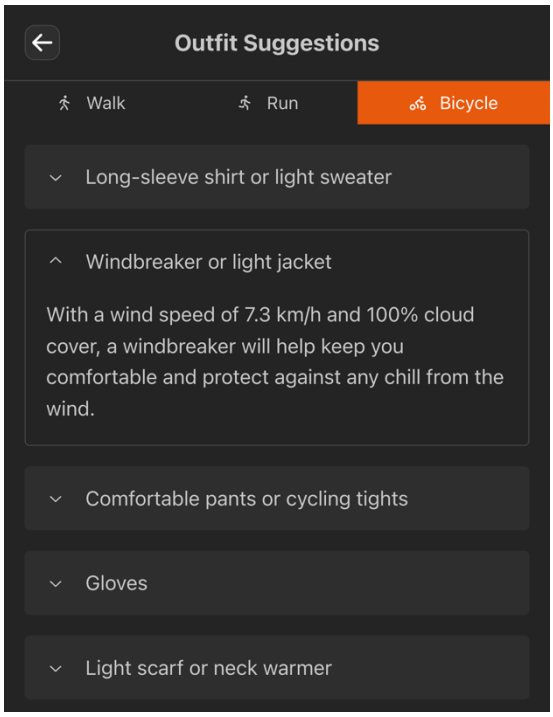

Рисунок 3.10 – Рекомендації щодо одягу для катанню на велосипеді

Ці зображення демонструють можливості додатку щодо генерації персоналізованих порад з одягу, адаптованих до обраного типу активності користувача. Завдяки інтеграції з OpenAI API, система формує не лише перелік речей, а й надає короткі пояснення до кожного пункту з урахуванням погодних параметрів, що суттєво підвищує інформативність та корисність функціоналу.

3.3.4 Сторінка планування поїздки

Далі можна побачити інтерфейс сторінки, що відповідає за управління поїздками користувача (рисунок 3.11). Екран відображає кнопку для створення нової поїздки та повідомлення про відсутність запланованих подорожей.

Це мінімалістичне оформлення дозволяє швидко зорієнтуватися та миттєво перейти до планування, не створюючи візуального перевантаження. Такий підхід сприяє простоті користування і логічній структурі взаємодії з розділом поїздок.

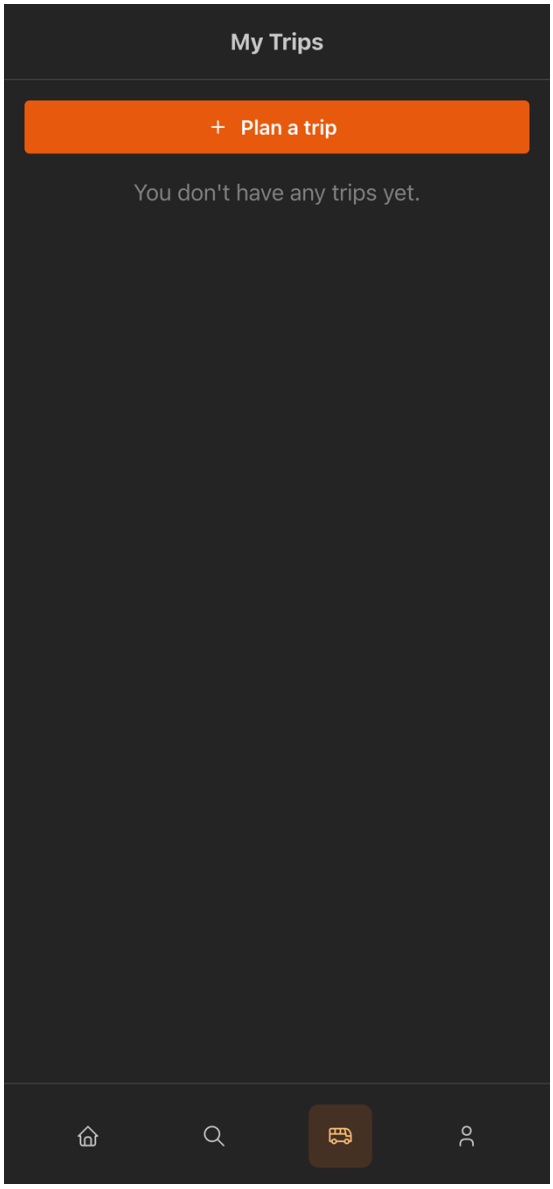

Рисунок 3.11 – Сторінка поїздок

Далі можна побачити інтерфейс створення поїздки з вибором міста з результатів пошуку (рисунок 3.12). Користувач вводить назву міста, після

чого система пропонує відповідні варіанти разом із інформацією про країну та чисельність населення.

Такий підхід дозволяє швидко знайти бажане місце призначення навіть серед однотипних назв, що підвищує точність і зручність планування. Кнопка «Pick» на кожному результаті забезпечує інтуїтивно зрозумілу взаємодію та переходить до наступного етапу створення маршруту.

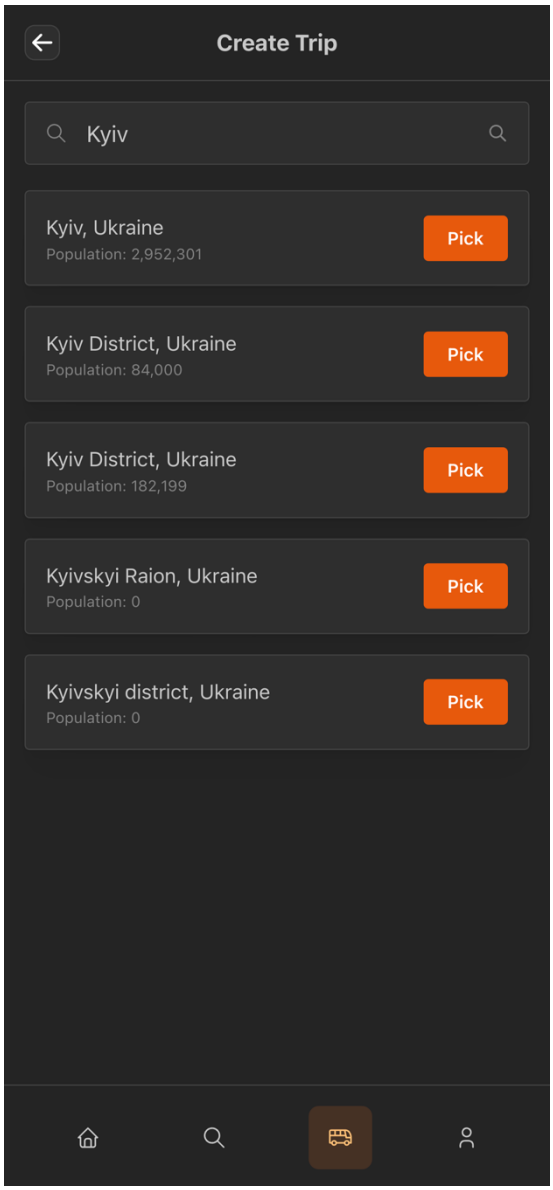

Рисунок 3.12 – Сторінка вибору міста для створення поїздки

Далі можна побачити інтерфейс заповнення форми створення нової поїздки користувача (рисунок 3.13). Після вибору міста користувач має змогу ввести назву поїздки, а також обрати дати подорожі.

Ця форма забезпечує базову валідацію даних і дозволяє чітко визначити часові рамки, необхідні для подальшого формування погодного прогнозу та рекомендацій. Кнопка «Create» внизу екрану запускає процес збереження поїздки у базу даних.

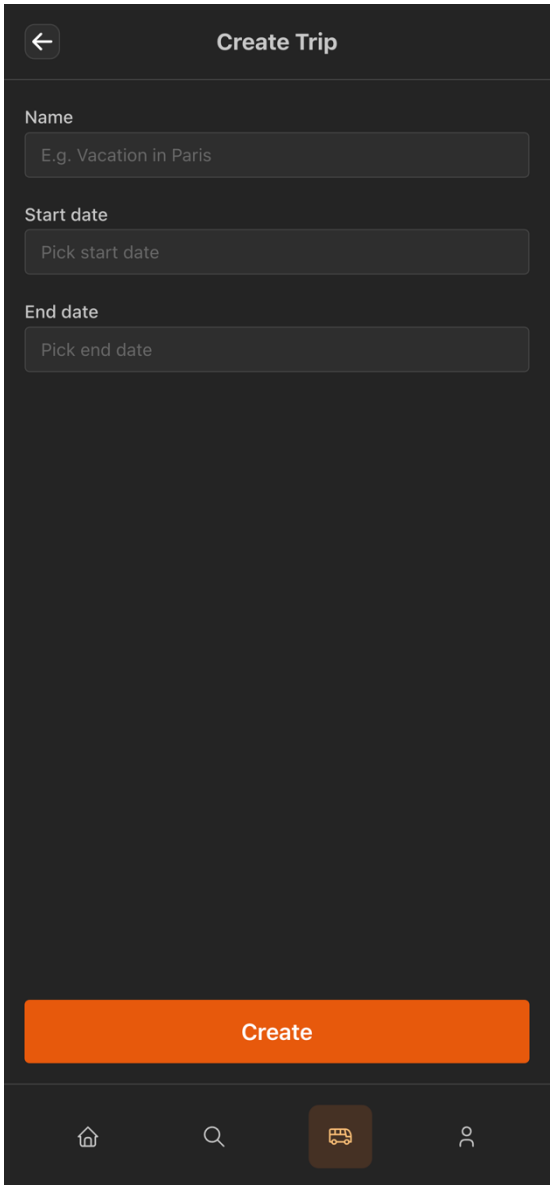

Рисунок 3.13 – Форма для створення поїздки

Далі можна побачити сторінку з детальною інформацією про поїздку, яка містить прогноз погоди на вказаний період (рисунок 3.14). Візуалізація охоплює щоденні температурні діапазони, що дозволяє швидко оцінити погодні умови протягом всієї подорожі.

У верхній частині інтерфейсу розміщено кнопку переходу до отримання рекомендацій щодо одягу, що підкреслює інтегровану логіку взаємозв'язку між даними про поїздки, погодою та порадами системи.

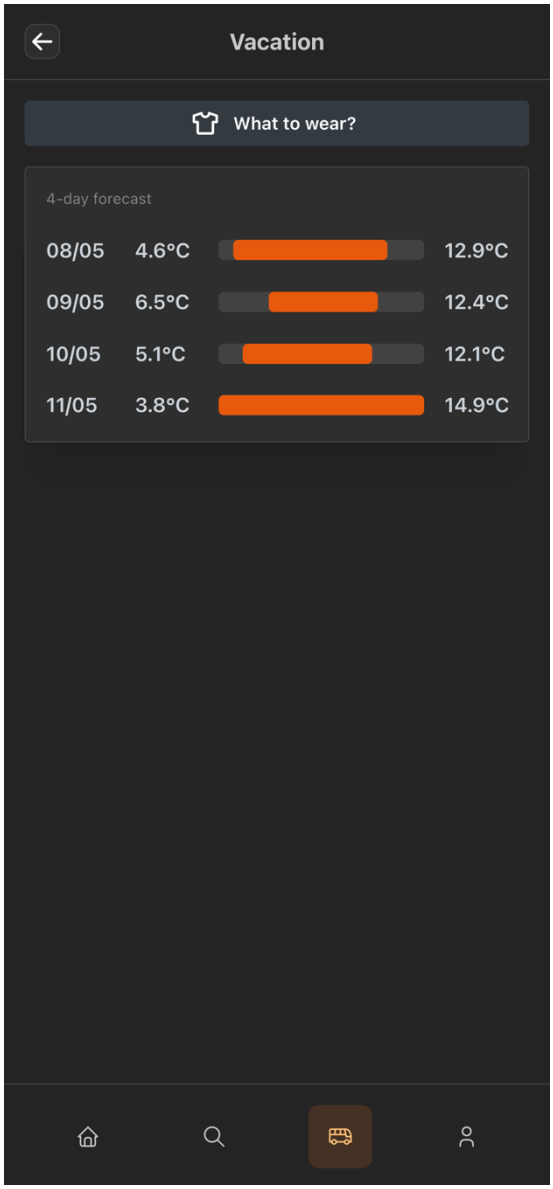

Рисунок 3.14 – Сторінка обраної поїздки

Далі можна побачити список рекомендацій щодо одягу для всієї поїздки, сформований на основі погодного прогнозу (рисунок 3.15). Рекомендації включають базові речі, такі як куртка, светр, сорочки з довгим рукавом, взуття для прогулянок і додаткові аксесуари.

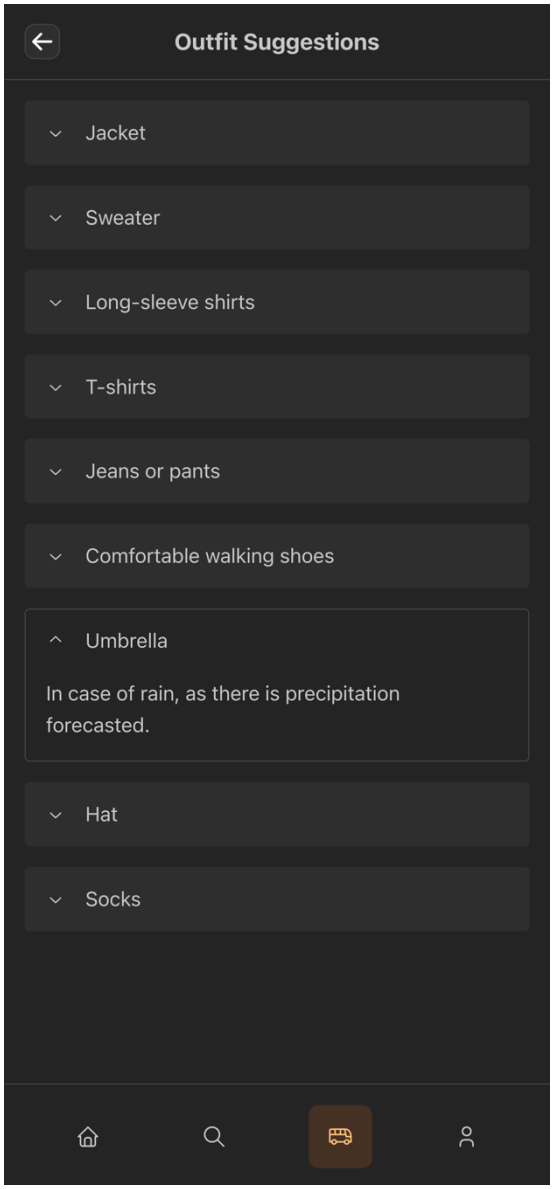

Рисунок 3.15 – Сторінка рекомендацій щодо одягу для поїздки

Значна увага приділяється адаптації до погодних умов – наприклад, наявність парасольки обґрунтована прогнозованими опадами. Така

деталізація підвищує практичну цінність системи та сприяє комфортному плануванню подорожі.

3.4 Збереження даних у Firestore

Зберігання даних у Firestore є невід’ємною частиною реалізованої системи, адже саме ця база даних забезпечує гнучке, масштабоване та у реальному часі доступне зберігання інформації про користувача, його міста, поїздки та погодні дані. Використання Firestore дозволяє реалізувати архітектуру, в якій дані динамічно оновлюються при додаванні чи видаленні елементів, а користувач відразу бачить актуальний стан інтерфейсу без потреби в ручному оновленні сторінки.

У структурі даних додатку окреме місце займає колекція users, яка містить документи користувачів з унікальним ідентифікатором auth0Id. Усередині кожного документа міститься два масиви – cities та trips. Масив cities використовується для зберігання об'єктів міст, які були додані користувачем вручну через інтерфейс, а масив trips — для зберігання створених подорожей, які містять посилання на місто, діапазон дат та назву поїздки.

Далі можна побачити приклад об'єкта міста, який було додано до бази даних користувача (рисунок 3.16). Такий запис містить повну інформацію про місто, включаючи географічні координати (широту та довготу), населення, часовий пояс, код країни та ідентифікатори з відкритих джерел, зокрема wikiDataId та regionWdId. Завдяки повноті даних можна легко інтегрувати додаткові сервіси, такі як пошук по регіону або побудова погодних звітів за координатами.

Зберігання міста саме в такому вигляді дає змогу уникнути додаткових запитів до зовнішнього API для отримання географічних характеристик, що знижує навантаження на API і прискорює час відповіді інтерфейсу. Такий підхід дозволяє не лише покращити продуктивність, а й

забезпечити більшу стійкість додатку до помилок у зовнішніх сервісах, зберігаючи вже відомі користувачу дані локально у Firestore.

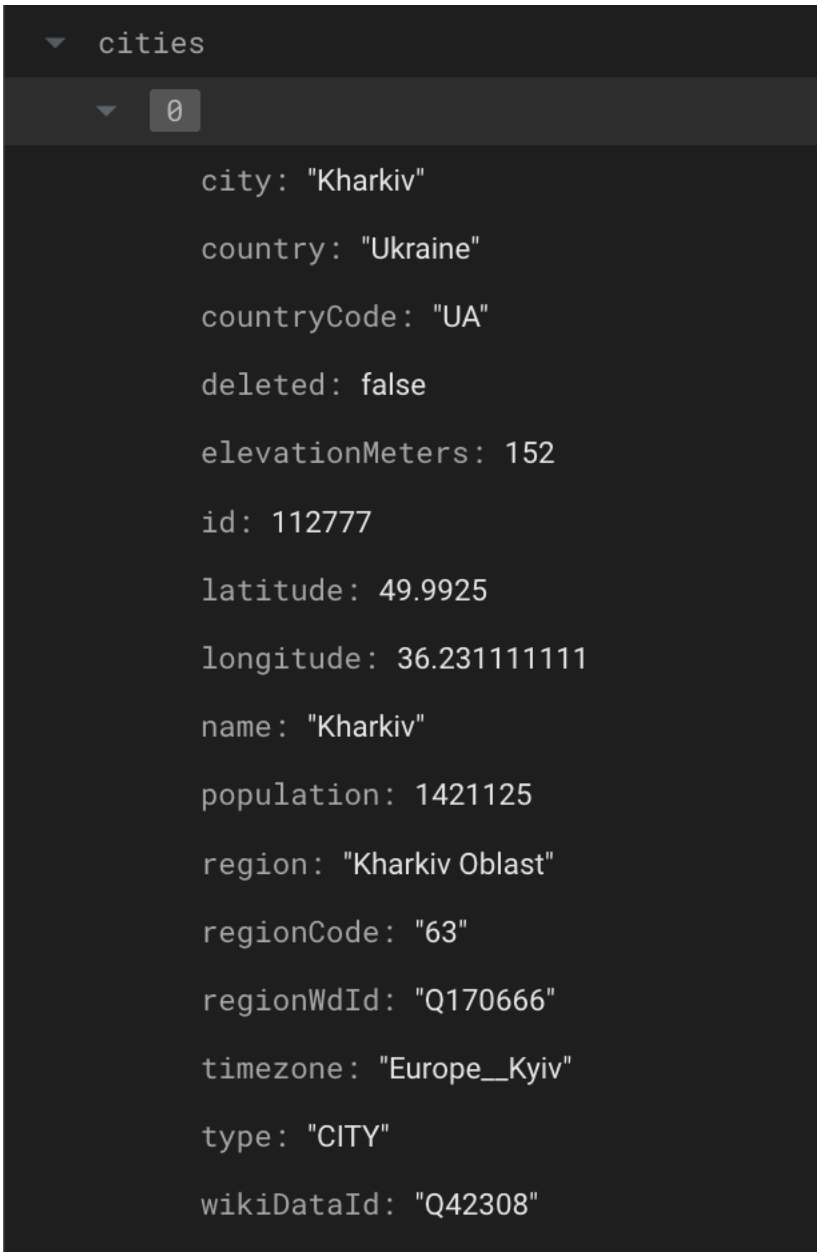

Рисунок 3.16 – Об’єкт міста

Поїздки користувача зберігаються у Firestore в масиві trips, що є частиною основного документа користувача. Кожен елемент цього масиву містить повний об'єкт обраного міста, а також метаінформацію, таку як назва поїздки, дата початку і завершення, та унікальний ідентифікатор

поїздки. Такий підхід дозволяє зберігати всі необхідні дані без додаткових запитів до інших колекцій. Далі можна побачити приклад збереженої поїздки користувача, яка включає поїздку до Києва (рисунок 3.17). Як видно, до структури даних включено повну інформацію про місто – від географічних координат до чисельності населення.

Рисунок 3.17 – Об’єкт поїздки

3.5 Забезпечення PWA-функціональності

Для забезпечення PWA-функціональності у додатку було реалізовано стандартний набір налаштувань, рекомендований Google. Насамперед, у корені проекту було створено файл manifest.json, який містить базову інформацію про додаток – назву, коротку назву, опис, іконки для різних розмірів, кольори теми та фону, а також стартову сторінку. Це дозволяє користувачеві встановити додаток на головний екран пристрою з відповідною іконкою.

У проекті також використовується service worker, який автоматично генерується за допомогою бібліотеки Vite PWA plugin. Цей сервісний працівник кешує основні ресурси додатку, що дає змогу відкривати інтерфейс навіть без підключення до мережі. Таким чином, ключові сторінки та інтерфейсні елементи залишаються доступними навіть в офлайн-режимі, що підвищує надійність та зручність використання.

Щоб зробити додаток інстальованим, до index.html було додано посилання на manifest.json та тег meta з параметром theme-color. Це дозволяє браузеру коректно ідентифікувати додаток як інстальований і виводити відповідний інтерфейс для встановлення на пристрої. Крім того, для Android реалізовано сповіщення про можливість додавання додатку на домашній екран.

Усі зміни тестувались через Lighthouse у браузері Chrome, що підтвердило відповідність PWA-критеріям. Результати перевірки показали високий рівень продуктивності, доступності та відповідності рекомендаціям щодо прогресивних вебдодатків. Таким чином, була повністю реалізована PWA-функціональність забезпечує користувачам повноцінний досвід використання додатку як нативного мобільного рішення.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було повністю реалізовано розробку вебдодатку для отримання персоналізованих порад щодо одягу на основі погодних умов, з використанням сучасних вебтехнологій та штучного інтелекту. Завдання, поставлені на початковому етапі проектування, було виконано у повному обсязі. Створено повнофункціональну систему, яка дозволяє користувачеві додавати міста до свого профілю, переглядати прогноз погоди, створювати поїздки, а також отримувати інтелектуальні рекомендації щодо вибору одягу відповідно до погодних умов та типу активності. Якісні характеристики додатку підтверджуються зручністю інтерфейсу, стабільною роботою сервісу в онлайн- та офлайн-режимах, а також коректною інтеграцією з OpenAI API. Кількісні результати свідчать про ефективну роботу системи – середній час генерації рекомендацій становить менше однієї секунди, а точність формування запиту до моделі дозволяє отримувати повністю релевантні відповіді.

Порівняння з аналогами, які існують на ринку, показує, що додаток вигідно відрізняється своїм фокусом на індивідуалізацію порад. Існуючі сервіси, зокрема традиційні погодні додатки, зазвичай надають лише суху метеоінформацію або обмежуються базовими порадами на кшталт «візьміть парасольку». У запропонованому рішенні використано модель GPT, яка не лише враховує всі погодні параметри, але й адаптує відповіді залежно від типу запланованої активності користувача – прогулянки, пробіжки чи велопоїздки. Крім того, на відміну від деяких закордонних аналогів, додаток не потребує глибокого налаштування, не прив’язаний до конкретного географічного регіону і не зберігає персональні дані користувача. Це робить його більш гнучким, безпечним і зручним у використанні.

З огляду на досягнутий результат, рекомендується продовжити розробку у напрямку поглиблення персоналізації порад, орієнтуючись на

індивідуальні потреби та контекст кожного користувача. Одним із ключових напрямів удосконалення може стати впровадження функціоналу ведення особистого гардеробу. Це дозволить формувати рекомендації не лише на основі погодних умов, а й з урахуванням наявного одягу конкретної людини, що зробить поради максимально практичними та релевантними. Додатково варто реалізувати функцію фіксації вподобань користувача – наприклад, стилістичних уподобань, чутливості до холоду чи вітру – і на їх основі адаптувати результати, поступово навчаючи модель реагувати відповідно до індивідуального стилю життя.

Перспективним є також розширення інтерактивних можливостей додатку за рахунок інтеграції з картографічними сервісами та геолокацією. Це дозволить автоматично визначати місце перебування користувача та пропонувати прогнози разом із рекомендаціями для поточної локації без потреби ручного додавання міста. Такий підхід значно покращить зручність використання додатку та відкриє можливість для подорожуючих або людей, що часто змінюють місце перебування. Крім того, можна реалізувати сценарії поведінки у вигляді нагадувань про зміну погоди або необхідність взяти із собою певний одяг, завдяки вбудованим push-сповіщенням, які формуються у режимі реального часу. Це забезпечить додатковий рівень взаємодії із додатком та підвищить його корисність у повсякденному житті.

Таким чином, реалізований додаток є повноцінним, сучасним та ефективним прикладом поєднання погодного аналізу з рекомендаційними можливостями штучного інтелекту, що відкриває широкі перспективи для масштабування і подальшого розвитку системи в напряму персоналізованих цифрових помічників.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Auth0. Auth0 Documentation. URL: https://auth0.com/docs. (date of access: 01.05.2025).

2. Azure M. Introduction to Azure Maps Weather Services. URL: https://learn.microsoft.com/en-us/azure/azure-maps/weather-services. (date of access: 01.05.2025).

3. Chollet F. Deep Learning with Python. Shelter Island : Manning Publications, 2021. 504 p.

4. DKN: Deep knowledge-aware network for news recommendation / H. Wang et al. *Proceedings of the 2018 World Wide Web Conference*. 2018. P. 1835–1844. URL: https://doi.org/10.1145/3178876.3186175.

5. Documentation and components. URL: https://mantine.dev (date of access: 01.05.2025).

6. Geolocation API overview. URL: https://developers.google.com/maps/documentation/geolocation/overview (date of access: 01.05.2025).

7. Kaur P., Singh M. A systematic review on recommender systems using artificial intelligence techniques. *Artificial Intelligence Review*. 2022. Vol. 55, no. 5. P. 4021–4072. URL: https://doi.org/10.1007/s10462-021-10053-9.

8. Network M. D. Using the Fetch API. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API. (date of access: 01.05.2025).

9. Nguyen T. H., Nguyen Q. V. H. Personalized weather-based outfit recommendation using deep learning: A survey. *Journal of Ambient Intelligence and Humanized Computing*. 2023. Vol. 14. P. 1861–1874. URL: https://doi.org/10.1007/s12652-022-03756-2.

10. Nielsen J. Usability Engineering. Boston : Academic Press, 2021. 362 p.

11. OpenAI API documentation. URL: https://platform.openai.com/docs (date of access: 01.05.2025).

12. Open-Meteo. Weather API for developers. URL: https://open-meteo.com. (date of access: 01.05.2025).

13. Organization W. M. Guide to Meteorological Instruments and Methods of Observation. Geneva : WMO, 2022. 832 p.

14. Paper recommender systems: A literature survey / J. Beel et al. *International Journal on Digital Libraries*. 2016. Vol. 17, no. 4. P. 305–338. URL: https://doi.org/10.1007/s00799-015-0156-0.

15. Raza S., Ding Z. A survey of outfit recommendation systems with weather and occasion awareness. *ACM Computing Surveys*. 2022. Vol. 55, no. 9. P. 1–34. URL: https://doi.org/10.1145/3517182.

16. React. A JavaScript library for building user interfaces. URL: https://reactjs.org. (date of access: 01.05.2025).

17. React TypeScript Cheat Sheet. URL: https://react-typescript-cheatsheet.netlify.app (date of access: 01.05.2025).

18. Russell S., Norvig P. Artificial Intelligence: A Modern Approach. Harlow : Pearson, 2021. 1152 p.

19. Smart weather-based clothing advice. URL: https://whatoweather.com (date of access: 01.05.2025).

20. Style meets weather: Personalized outfit ideas. URL: https://www.dresscast.com (date of access: 01.05.2025).

21. Turing A. M. Computing machinery and intelligence. *Mind*. 1950. Vol. 59, no. 236. P. 433–460. URL: https://doi.org/10.1093/mind/LIX.236.433.

22. Vasiljeva I., Nikiforova O. The implementation of recommender systems in e-commerce: A review. *Procedia Computer Science*. 2021. Vol. 192. P. 2309–2318. URL: https://doi.org/10.1016/j.procs.2021.08.237.

23. Weather API documentation. URL: https://openweathermap.org/api (date of access: 01.05.2025).

ДОДАТОК А

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	101 с.
	Інші документи	
2.	Презентаційні матеріали	30 плакатів

.	.	Прізвище та ініціали.	Підп.	Дата	
Розробив		Васильченко А.О.			Вебдодаток для отримання рекомендацій щодо одягу на основі поточних погодних умов за допомогою штучного інтелекту
Перевірив		Малєєв Д.В.			
Н.контр.		Малєєва І.А.			
Затв.		Золотухін О.В.			

Шифр групи	Код напр./спец.
ІТШІ-21-5	122
ХНУРЕ кафедра ШІ	