**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ**
**І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ**
**Факультет інформаційних технологій**

**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
**Завідувач кафедри**
Комп'ютерних наук
____________ Голуб Б.Л.

"02" червня 2025 р.

# БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

**на тему**

**Програмне забезпечення системи моніторингу погодних умов**

Спеціальність 121 – «Інженерія програмного забезпечення»

**Гарант освітньої програми**
К.т.н., доцент ______________ Вайганг Г.О.

**Керівник бакалаврської кваліфікаційної роботи**

____________________ ______________ Степанов О.В.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

**Виконав** ____________ Архипов Вадим Андрійович
(підпис) (ПІБ студента)

**КИЇВ – 2025**

**НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ**
**Факультет інформаційних технологій**

**ЗАТВЕРДЖУЮ**
**Завідувач кафедри**
Комп'ютерних наук
____________ Голуб Б.Л.

"16" грудня 2024 р.

# ЗАВДАННЯ

**на виконання бакалаврської кваліфікаційної роботи студенту**

__________ Архипову Вадиму Андрійовичу ____________________
(прізвище, ім'я, по батькові)

Спеціальність 121 – «Інженерія програмного забезпечення»
Тема бакалаврської кваліфікаційної роботи _ Програмне забезпечення системи моніторингу погодних умов _
затверджена наказом ректора НУБіП України від "16" грудня 2024р. №2249С

Термін подання завершеної роботи на кафедру 2025. 05. _25__
(рік, місяць, число)

Вихідні дані до бакалаврської кваліфікаційної роботи:
опис програмного забезпечення

Перелік питань, які потрібно розробити:

- системний аналіз предметної області;
- проєктування інформаційного та програмного забезпечення;
- розробка інформаційного та програмного забезпечення;
- рекомендації щодо впровадження та експлуатації системи.

**Дата видачі завдання** "16" грудня 2025 р.
**Керівник бакалаврської кваліфікаційної роботи**

______________________ ______________ _Степанов О.В.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

**Завдання прийняв до виконання** ___________ Архипов В.А.
(підпис) (ПІБ студента)

# ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ.......... 5
ВСТУП.......... 6
1. СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ.......... 9
1.1 Опис предметної області.......... 9
1.2 Аналіз вимог до програмної системи.......... 10
1.3 Моделювання предметної області.......... 14
1.4 Огляд інформаційних джерел та існуючих рішень.......... 20
1.5 Постановка завдання.......... 23
2. ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.......... 26
2.1 Логічна модель даних у вигляді ER-діаграми.......... 26
2.2 Діаграма класів та кооперацій.......... 34
2.3 Діаграма пакетів.......... 39
2.4 Діаграма компонентів.......... 40
3. РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.......... 42
3.1 Система управління інформаційною базою.......... 42
3.2 Розробка інформаційної бази.......... 46
3.3 Вибір інструментарію для створення прикладного програмного забезпечення.......... 48
3.4 Алгоритмізація та програмування програмних модулів.......... 54
4. РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ.......... 59
4.1 Тестування системи.......... 59

4.2 Вимоги до апаратного та програмного забезпечення................................. 67
4.3 Склад інсталяційного пакету...................................................................70
ВИСНОВКИ...............................................................................................73
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.........................................................75
Додаток А..................................................................................................... 77
Додаток Б......................................................................................................79
Додаток В..................................................................................................... 83
Додаток Г......................................................................................................84
Додаток Д..................................................................................................... 87

# ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API - Application Programming Interface (інтерфейс програмування додатків)

ACID - Atomicity, Consistency, Isolation, Durability (атомарність, узгодженість, ізольованість, довговічність)

CSS - Cascading Style Sheets (каскадні таблиці стилів)

CDN - Content Delivery Network (мережа доставки контенту)

CI/CD - Continuous Integration/Continuous Deployment (безперервна інтеграція/безперервне розгортання)

DDD - Domain-Driven Design (предметно-орієнтоване проектування)

ER - Entity-Relationship (сутність-зв'язок)

HTML - HyperText Markup Language (мова розмітки гіпертексту)

HTTP - HyperText Transfer Protocol (протокол передачі гіпертексту)

JS - JavaScript (мова програмування JavaScript)

JSON - JavaScript Object Notation (формат обміну даними)

JWT - JSON Web Token (веб-токен формату JSON)

NoSQL - Not Only SQL (не тільки SQL)

OAuth - Open Authorization (відкрита авторизація)

PWA - Progressive Web App (прогресивний веб-додаток)

REST - Representational State Transfer (передача репрезентативного стану)

SCSS - Sassy CSS (синтаксичне розширення CSS)

SSL - Secure Sockets Layer (рівень захищених сокетів)

SQL - Structured Query Language (мова структурованих запитів)

SVG - Scalable Vector Graphics (масштабована векторна графіка)

UI - User Interface (інтерфейс користувача)

UML - Unified Modeling Language (уніфікована мова моделювання)

URL - Uniform Resource Locator (уніфікований локатор ресурсів)

UX - User Experience (досвід користувача)

# ВСТУП

Погода — це змінна, яка впливає на рішення людей частіше, ніж здається. Коли вирушити в дорогу, як вдягнутися, чи варто планувати подію на відкритому повітрі — усе це залежить від доступу до точних метеоданих. У відповідь на цю щоденну потребу постала ідея створення веб-додатку для моніторингу погодних умов.

Метою роботи є розробка та впровадження інтерактивної системи моніторингу погодних умов у вигляді веб-додатку, що забезпечує користувачам зручний доступ до персоналізованої метеорологічної інформації з можливістю створення власних колекцій міст та гнучкими налаштуваннями інтерфейсу.

Для досягнення поставленої мети були визначені такі завдання: проаналізувати існуючі рішення та системи моніторингу погодних умов, визначити їх переваги та недоліки; змоделювати предметну область та спроектувати структуру бази даних для ефективного зберігання та обробки метеорологічної інформації; проаналізувати та обрати оптимальні технології для розробки програмного додатку з урахуванням вимог до функціональності та продуктивності; розробити архітектуру програмного забезпечення з використанням компонентного підходу та сучасних патернів проектування; реалізувати інтерфейс користувача з адаптивним дизайном та підтримкою різних типів пристроїв; інтегрувати системи автентифікації та авторизації користувачів для забезпечення персоналізації; провести тестування програмного додатку та оптимізувати його роботу; підготувати рекомендації щодо подальшого розвитку та масштабування системи.

Для реалізації поставлених завдань були використані сучасні методи та технології розробки програмного забезпечення. Бібліотека React з TypeScript використовувалась для створення компонентної структури інтерфейсу, що забезпечує модульність та повторне використання коду. Firebase Authentication застосовувався для реалізації системи автентифікації з підтримкою різних методів входу (Email/Password, Google). Firebase Firestore був обраний як

нереляційна база даних для зберігання інформації про користувачів та їх колекції міст, що забезпечує гнучкість структури даних та масштабованість. Для отримання актуальної метеорологічної інформації з високою точністю використовувався OpenWeather API. Для створення адаптивного дизайну, який коректно відображається на різних типах пристроїв, застосовувались SCSS та Bootstrap. Material UI компоненти інтегровані для реалізації сучасних елементів інтерфейсу з підтримкою доступності. Також були впроваджені Progressive Web App (PWA) технології для забезпечення можливості використання додатку як десктопного застосунку без необхідності постійного відкриття браузера.

Результати дослідження та розробки програмного додатку було апробовано 24 квітня 2025 року на VII Всеукраїнській науково-практичній конференції студентів і аспірантів «Теоретичні та прикладні аспекти розробки комп'ютерних систем '2025», де представлено архітектурні рішення та функціональність системи.[10]

Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. Розділ "Системний аналіз предметної області" охоплює опис предметної області, аналіз вимог, моделювання предметної області, огляд існуючих рішень та постановку завдання для системи моніторингу погодних умов. В розділі "Проєктування інформаційного та програмного забезпечення" представлено логічну модель даних, діаграми класів, кооперацій, пакетів та компонентів, що описують структурну організацію системи. Розділ "Розробка інформаційного та програмного забезпечення" розглядає систему управління інформаційною базою, розробку інформаційної бази, вибір інструментарію та алгоритмізацію програмних модулів. Розділ "Рекомендації щодо впровадження та експлуатації системи" містить інформацію про тестування системи, вимоги до апаратного та програмного забезпечення, а також склад інсталяційного пакету.

Цей дипломний проєкт не лише демонструє технічні навички роботи з сучасними веб-технологіями, а й слугує готовим продуктом, який може бути масштабований або доповнений новими модулями — наприклад, історією змін

погоди, сповіщеннями або геолокацією. Його створення — це приклад того, як з простого завдання можна побудувати повноцінну програмну систему із зручним UI, багатофункціональним UX і можливістю персоналізації.

# 1. СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

## 1.1 Опис предметної області

Предметна область "Моніторинг погодних умов" охоплює систему знань, процесів та даних, пов'язаних із дослідженням, вимірюванням, аналізом та прогнозуванням атмосферних явищ та їхніх показників з особливим фокусом на потреби туристів та мандрівників.

Ключовим об'єктом в цій предметній області є погодні умови — комплекс атмосферних явищ та показників, що характеризують стан нижніх шарів атмосфери в певній географічній точці в конкретний момент часу. Ці показники включають температуру повітря, вологість, швидкість і напрям вітру.

У контексті туристичної активності, погодні умови є критичним фактором, що впливає на планування подорожей, вибір спорядження, організацію маршрутів та загальну безпеку під час переміщення. Різноманітні географічні локації характеризуються унікальними кліматичними особливостями, які можуть суттєво впливати на можливість та комфортність перебування в них.

Важливими концептами предметної області є:

Метеорологічні дані — набір кількісних та якісних показників, що описують стан атмосфери. Ці дані отримуються шляхом вимірювань за допомогою спеціалізованих приладів та мереж метеостанцій.

Географічні локації (міста) — просторові точки, до яких прив'язуються метеорологічні дані та прогнози. Кожна локація має свої унікальні кліматичні характеристики.

Погодні феномени — конкретні атмосферні явища, такі як дощ, сніг, туман, грози, які впливають на можливість здійснення туристичної активності.

Сезонність — циклічна змінність погодних умов протягом року, що визначає оптимальні періоди для відвідування певних місць.

Прогнозування погоди — науково-обґрунтоване передбачення зміни погодних умов у майбутньому на основі аналізу поточних даних.

Для туристів особливу цінність представляє актуальна та достовірна інформація про погодні умови, оскільки вона безпосередньо впливає на якість відпочинку, безпеку та загальні враження від подорожі. Несприятливі погодні умови можуть спричинити ризики для здоров'я, пошкодження майна або вимагати зміни запланованих активностей.

В епоху глобалізації та підвищення мобільності людей, туристи потребують зручних інструментів для моніторингу погодних умов у різних частинах світу. Ці інструменти мають забезпечувати не лише актуальні дані для конкретної локації, але й можливість порівняння показників для кількох потенційних напрямків подорожі, що допомагає у прийнятті обґрунтованих рішень щодо планування маршруту.

Сучасні технології дозволяють представляти метеорологічні дані у візуально зрозумілій формі, з використанням умовних позначень, кольорового кодування та інтерактивних елементів, що підвищує їхню доступність для широкого кола користувачів без спеціалізованих знань у галузі метеорології.

## 1.2 Аналіз вимог до програмної системи

Програмне забезпечення призначене для забезпечення туристів та мандрівників актуальною інформацією про погодні умови у різних географічних локаціях. Система має надавати можливість моніторингу основних метеорологічних показників, таких як температура, вологість, швидкість вітру та інші релевантні дані, з метою покращення планування подорожей та підвищення безпеки туристичної активності.

**Функціональні вимоги**

**Автентифікація та управління користувачами**

- Система повинна забезпечувати реєстрацію нових користувачів з використанням електронної пошти та пароля.
- Система повинна підтримувати автентифікацію через сторонні сервіси (Google).

- Реалізувати функцію відновлення паролю через електронну пошту.
- Забезпечити захист персональних даних користувачів відповідно до сучасних стандартів.
- Надавати можливість перегляду та редагування профілю користувача.

**Управління локаціями та погодними даними**

- Система повинна дозволяти користувачу додавати нові міста для моніторингу погодних умов.
- Реалізувати функціонал пошуку міст за назвою з автоматичними підказками.
- Забезпечити можливість оновлення інформації про місто (заміна на інше місто).
- Надати можливість видалення міста зі списку моніторингу.
- Система повинна автоматично запитувати дані при виборі міста зі списку результатів пошуку.

**Представлення метеорологічної інформації**

- Відображати поточні погодні умови для кожного вибраного міста.
- Представляти основні метеорологічні показники: температуру, вологість, швидкість вітру.
- Візуалізувати погодні умови з використанням інтуїтивно зрозумілих іконок.
- Надавати можливість перегляду погодного прогнозу на кілька днів вперед.

**Нефункціональні вимоги**

**Вимоги до продуктивності**

- Система повинна забезпечувати час відповіді на запит погодних даних не більше 1,5 секунди.
- Підтримка одночасної роботи не менше 100 користувачів.
- Забезпечення доступності системи на рівні 99,9%.

**Вимоги до інтерфейсу**

- Реалізація адаптивного дизайну для коректного відображення на різних типах пристроїв (десктоп, планшет, мобільний телефон).
- Підтримка світлої та темної теми інтерфейсу з можливістю автоматичного перемикання відповідно до системних налаштувань.
- Забезпечення інтуїтивно зрозумілого користувацького інтерфейсу з мінімальною кривою навчання.
- Розробка компонентів інтерфейсу відповідно до сучасних принципів UX/UI дизайну.
- Забезпечити доступність інтерфейсу відповідно до стандартів WCAG 2.1 рівня AA.

**Вимоги до безпеки**

- Захист API-ключів, особливо для взаємодії з метеорологічними сервісами, шляхом використання серверних компонентів.
- Шифрування даних автентифікації користувачів.
- Реалізація механізмів захисту від основних типів веб-атак (XSS, CSRF, SQL Injection).
- Періодичне резервне копіювання даних користувачів.
- Обмеження кількості невдалих спроб входу для запобігання підбору паролів.

**Вимоги до надійності та обробки помилок**

- Реалізація ізольованої обробки помилок на рівні компонентів для забезпечення стійкості системи.
- Впровадження механізмів логування помилок з можливістю їх аналізу.
- Забезпечення коректної роботи системи в умовах нестабільного підключення до мережі.
- Запровадження механізмів автоматичного відновлення після збоїв.

- Реалізація інформативних та зрозумілих повідомлень про помилки для користувачів.

**Вимоги до масштабованості**

- Архітектурне рішення, що дозволяє горизонтальне масштабування системи.
- Можливість розширення функціональності без суттєвої реструктуризації існуючих компонентів.
- Підтримка додавання нових джерел метеорологічних даних у майбутньому.
- Забезпечення можливості інтеграції з іншими туристичними сервісами.

**Технологічні вимоги**

- Використання React з TypeScript для розробки клієнтської частини.
- Застосування Firebase для реалізації автентифікації та хостингу.
- Інтеграція з OpenWeather API або іншими метеорологічними сервісами для отримання даних.
- Використання Material UI та SCSS для забезпечення сучасного та узгодженого дизайну компонентів.
- Реалізація архітектури на основі контекстів для ефективного управління станом додатку.
- Забезпечення коректної роботи у основних сучасних браузерах (Chrome, Firefox, Safari, Edge).

Успішна реалізація всіх зазначених вимог забезпечить створення високоякісної, надійної та зручної системи моніторингу погодних умов, що відповідатиме потребам туристів та мандрівників у отриманні своєчасної та точної метеорологічної інформації.

## 1.3 Моделювання предметної області

**Сутність та призначення моделювання предметної області**

Моделювання предметної області (Domain Modeling) — це процес створення концептуальної моделі, яка описує основні об'єкти, сутності, їхні атрибути, взаємозв'язки та бізнес-правила, що існують у певній проблемній області.

Сутності можуть використовувати простий підхід, коли зв'язки між ними відповідають способу їх збереження (наприклад, у базі даних, коли одна сутність відповідає одній таблиці), або складний підхід, коли повністю ігнорується спосіб їх збереження, а до уваги береться саме модель предметної області. Таким чином, об'єкти містять логіку, доступ до даних обмежений відповідним модифікатором доступу, між об'єктами присутні зв'язки наслідування, а також існують класи, що описують логіку та не зберігаються у сховищі.[20]

**Важливість моделювання предметної області**

Моделювання предметної області є критично важливим етапом розробки програмного забезпечення з кількох ключових причин:

**Єдине розуміння системи** — модель створює спільну мову та розуміння між усіма зацікавленими сторонами: замовниками, аналітиками, розробниками та кінцевими користувачами. Це мінімізує комунікаційні бар'єри та запобігає неправильному тлумаченню вимог.

**Зниження складності** — розбиваючи складну предметну область на логічні компоненти та взаємозв'язки, моделювання допомагає спростити розуміння системи та керування нею. Це особливо важливо для систем з багатьма взаємопов'язаними сутностями, такими як погодні умови, географічні локації та користувацькі профілі.

**Основа для проектування бази даних** — детальна модель предметної області слугує фундаментом для проектування структури бази даних, визначаючи, які таблиці, поля та зв'язки необхідно створити для ефективного зберігання інформації.

**Гнучкість та масштабованість** — якісна модель предметної області забезпечує гнучкість системи щодо майбутніх змін. Вона дозволяє легше інтегрувати нові функції, наприклад, додавання нових типів метеорологічних даних або розширення географічного охоплення.

**Зменшення ризиків розробки** — раннє виявлення потенційних проблем та суперечностей у предметній області дозволяє усунути їх до початку фази активної розробки, що суттєво знижує ризики проекту.

**Методи та інструменти моделювання**

Для ефективного моделювання предметної області системи моніторингу погодних умов можуть використовуватися різні методи та інструменти:

1. **UML-діаграми (Unified Modeling Language)** — потужний набір стандартизованих діаграм для візуалізації предметної області:
   - Діаграми класів для відображення структури даних та взаємозв'язків;
   - Діаграми прецедентів (Use Case) для моделювання взаємодії користувачів з системою;
   - Діаграми послідовностей для ілюстрації динамічних взаємодій між об'єктами.
2. **Предметно-орієнтоване проектування (DDD, Domain-Driven Design)** — підхід, який ставить модель предметної області в центр розробки, акцентуючи увагу на:
   - Створенні загальної мови (Ubiquitous Language) між технічними та нетехнічними учасниками;
   - Виділенні агрегатів, сутностей та об'єктів-значень;
   - Визначенні меж контекстів для різних аспектів системи.

**Аналіз представлених діаграм**

Діаграма прецедентів — в UML, діаграма, на якій зображено відношення між акторами та прецедентами в системі. Діаграма прецедентів показує різні варіанти використання та різні типи користувачів системи і часто супроводжується іншими типами діаграм. Варіанти використання представлені колами або еліпсами. Актори (дійові особи) часто зображуються у вигляді паличок.[16]

Представлена діаграма прецедентів на рис.1.1 ілюструє ключові варіанти використання системи моніторингу погодних умов. Основним актором є Турист, який взаємодіє з системою через три основні прецеденти:

- **Вибір міста** — функціональність, яка дозволяє користувачу вибрати географічну локацію для перегляду погодних даних.
- **Перегляд погодних даних** — основний прецедент, що забезпечує доступ до метеорологічної інформації (температура, вологість, швидкість вітру).
- **Надання погодних даних** — прецедент, пов'язаний із зовнішньою системою "Метеорологічний API", який демонструє джерело отримання інформації.

Така діаграма чітко окреслює межі системи та основні точки взаємодії користувача з нею, що є важливим для розуміння ключових функціональних вимог.

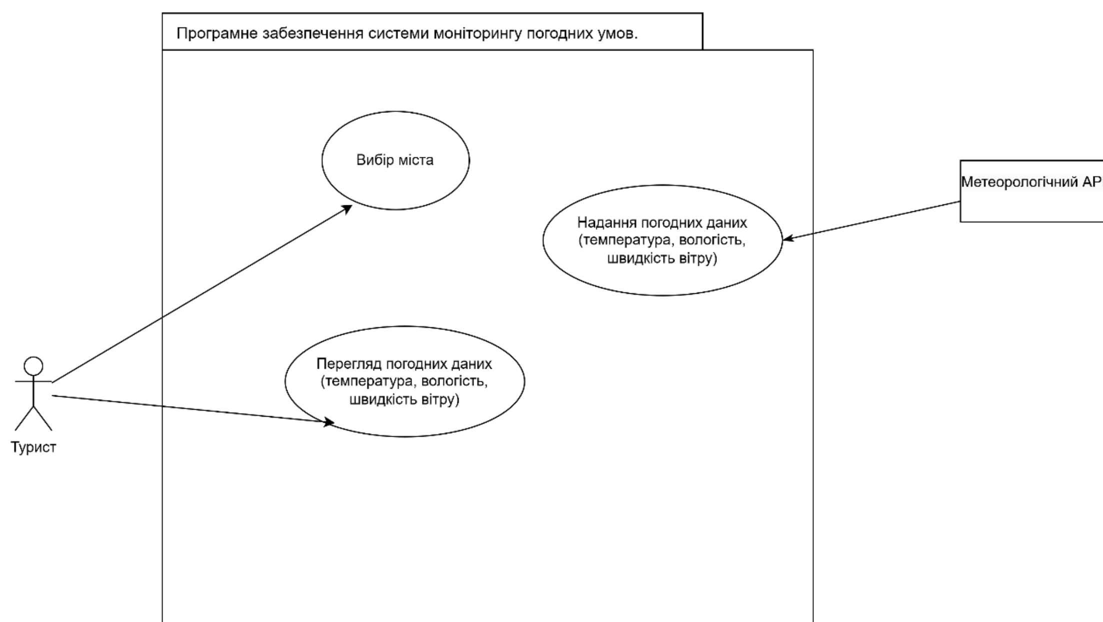


Рис. 1.1 Діаграма прецедентів (Use Case Diagram)

**Діаграма послідовності (Sequence Diagram)**

Діаграма послідовності — різновид діаграми в UML. Діаграма послідовності відображає взаємодії об'єктів впорядкованих за часом. Зокрема, такі діаграми відображають задіяні об'єкти та послідовність надісланих повідомлень.[15]

Діаграма послідовності на рис.1.2 детально відображає хронологію взаємодій між користувачем, додатком моніторингу погоди та API погоди. На ній можна простежити кілька ключових сценаріїв:

**Основний сценарій взаємодії:**

- Користувач здійснює вибір міста;
- Додаток надсилає запит погодних даних до API;
- API повертає результат з даними про погоду;
- Додаток відображає користувачу отримані погодні дані.

**Сценарій додавання нового міста:**

- Користувач ініціює додавання нового міста;
- Додаток зберігає нове місто;
- Додаток запитує погодні дані для нового міста;

- API повертає дані про погоду для нового міста;
- Додаток оновлює дані всіх міст;
- Користувач отримує оновлену інформацію.

Така діаграма допомагає зрозуміти потік даних та послідовність операцій, що важливо для проектування логіки взаємодії компонентів системи

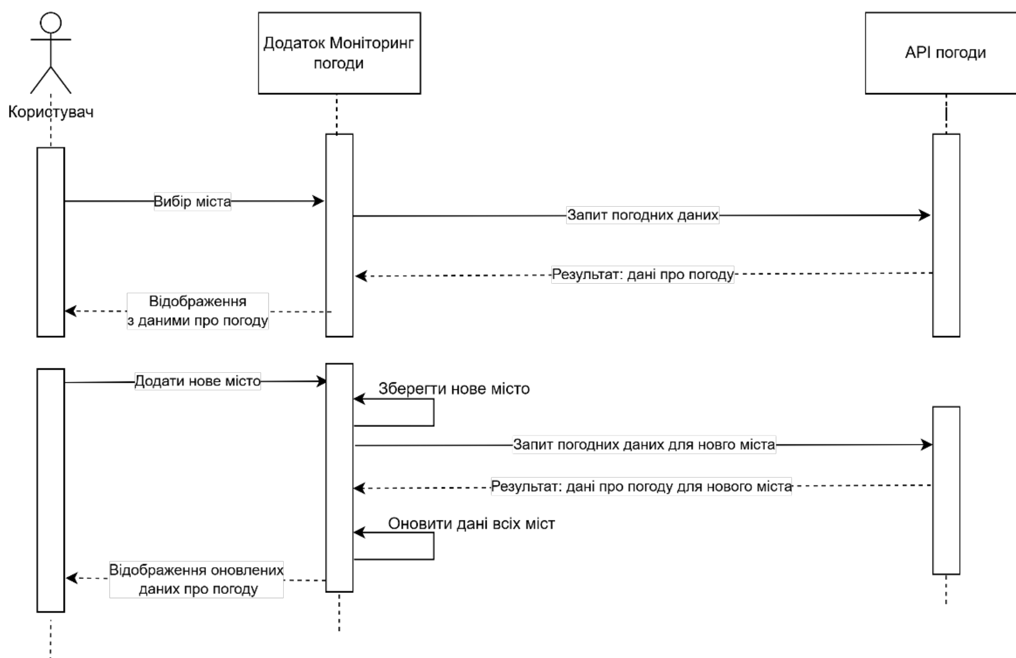


Рис. 1.2 Діаграма послідовності (Sequence Diagram)

Діаграма діяльності або активності— в UML та SysML, візуальне представлення графу діяльностей. Граф діяльностей є різновидом графу станів скінченного автомата, вершинами якого є певні дії, а переходи відбуваються по завершенню дій.[18]

Діаграма активності на рис.1.3 ілюструє робочий процес взаємодії користувача з системою та демонструє основні кроки та рішення, які відбуваються під час використання додатку:

**Початок взаємодії від стартової точки**

1. Вибір міста користувачем;
2. Запит до API погоди;
3. Отримання даних про погоду;
4. Відображення погодних даних.

**Точка прийняття рішення**:

- Завершення роботи (перехід до кінцевої точки);
- Додавання нового міста (повернення до початку процесу).

Ця діаграма наочно демонструє логіку роботи програми та допомагає розробникам зрозуміти послідовність дій та альтернативні шляхи виконання.

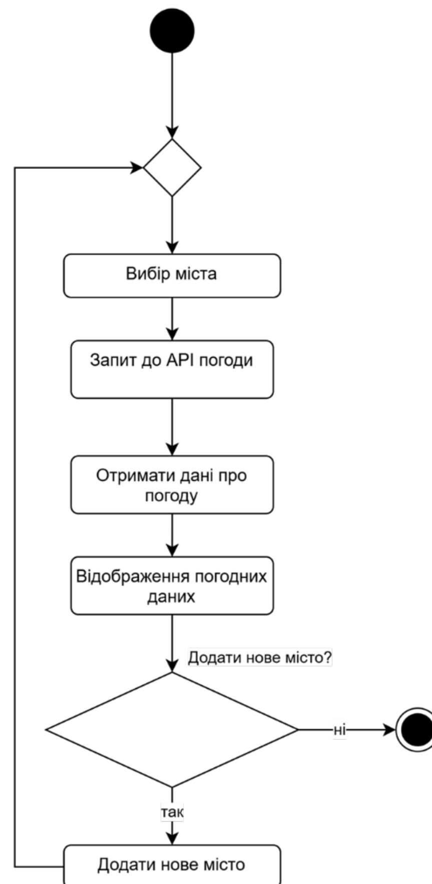


Рис. 1.3 Діаграма активності (Activity Diagram)

Представлені діаграми в комплексі формують цілісне розуміння предметної області та функціональності системи моніторингу погодних умов з різних перспектив, що є важливим для успішної реалізації проекту.

## 1.4 Огляд інформаційних джерел та існуючих рішень

Ринок веб-додатків для моніторингу погодних умов представлений різноманітними рішеннями з різним рівнем функціональності та зручності використання. Розглянемо чотири найбільш популярні системи, проаналізувавши їхні переваги та недоліки.

**Weather.com**

Провідний світовий постачальник метеорологічних даних. Дім для програми The Weather Channel, сайту weather.com та телемережі.[8]

**Переваги:**

- Всеохоплюючий спектр метеорологічних даних та прогнозів;
- Детальні карти опадів, радарні зображення та супутникові знімки;
- Інтеграція з системами сповіщень про екстремальні погодні умови.

**Недоліки:**

- Перевантажений інтерфейс з надмірною кількістю рекламних блоків;
- Складна навігація з численними вкладками та розділами;
- Обмежені можливості персоналізації без преміум-підписки;
- Значне споживання трафіку та повільне завантаження.

**AccuWeather**

AccuWeather надає прогнози погоди по всьому світу для ЗМІ, бізнесу та уряду, включаючи більш ніж половину компаній Fortune 500 та тисячі інших підприємств у всьому світі.[1]

**Переваги:**

- Інтуїтивно зрозумілий інтерфейс з акцентом на важливу інформацію;
- Детальні погодинні прогнози з високою точністю;
- Функція MinuteCast™ для попередження про зміни погоди в реальному часі;
- Добре оптимізований для мобільних пристроїв.

**Недоліки:**

- Обмеження кількості міст для відстеження у безкоштовній версії;
- Недостатня гнучкість у налаштуванні відображення даних;
- Відносно однотипний інтерфейс без можливості зміни теми оформлення;
- Обмежена функціональність офлайн-режиму.

**Windy.com**

Windy — чеський веб-сайт, що надає послуги інтерактивного прогнозу погоди по всьому світу.[9]

**Переваги:**

- Унікальна інтерактивна візуалізація вітрів та погодних явищ на картах;
- Потужні інструменти для аналізу метеорологічних даних;
- Підтримка різних моделей прогнозування погоди;
- Багатошарова система відображення даних з можливістю налаштування.

**Недоліки:**

- Висока складність інтерфейсу для пересічного користувача;
- Фокус на спеціалізованих метеорологічних даних, менше уваги базовим потребам;
- Обмежені можливості персоналізації профілю користувача;
- Надмірна кількість інформації для щоденного використання.

**Weather Underground**

**Переваги:**

- Унікальна система локальних метеостанцій для точних місцевих даних;
- Детальна історична інформація та статистика погоди;

- Міцна спільнота метеорологів-ентузіастів з можливістю обміну даними;
- Інформативні графіки зміни погодних показників.

**Недоліки:**

- Застарілий інтерфейс користувача на окремих сторінках;
- Нестабільна швидкість роботи під час пікових навантажень;
- Відсутність єдиного стилю між різними розділами сайту;
- Обмежена функціональність у режимі офлайн.

**Висновки з аналізу існуючих рішень**

Проведений аналіз існуючих рішень дозволяє виявити їх основні переваги та недоліки, а також визначити ключові можливості для вдосконалення у власній розробці.

Основними недоліками більшості існуючих систем є:

- Перевантаженість інтерфейсу зайвою інформацією або рекламою;
- Недостатня персоналізація користувацького досвіду;
- Обмежена підтримка різних тем оформлення;
- Неоптимальна робота на різних пристроях;
- Обмежені можливості відстеження погоди у кількох місцях одночасно.

Водночас, позитивними аспектами, які варто перейняти, є:

- Інтуїтивно зрозуміла візуалізація погодних умов;
- Детальні прогнози з різними часовими горизонтами;
- Збереження історії переглядів та налаштувань користувача;
- Швидкий доступ до найбільш важливої інформації.

Розроблюване програмне забезпечення спрямоване на подолання виявлених недоліків через впровадження зручного, персоналізованого інтерфейсу з адаптивним дизайном, можливістю зміни тем оформлення, фоновим музичним супроводом та ефективною роботою в режимі PWA для швидкого доступу без переходу на сайт.

## 1.5 Постановка завдання

Основною метою проекту є розробка сучасної системи моніторингу погодних умов, що забезпечить користувачів, зокрема туристів та мандрівників, оперативним доступом до актуальної метеорологічної інформації. Призначення системи полягає у створенні зручного та персоналізованого інструменту для відстеження погодних умов у різних географічних локаціях з метою ефективного планування подорожей, підвищення комфорту та безпеки під час туристичної активності.

**Ключові завдання проекту**

**Розробка системи автентифікації та персоналізації**

Впровадження надійної системи управління користувачами з використанням Firebase Authentication, що включає:

- Реєстрацію та вхід через електронну пошту та пароль;
- Інтеграцію з Google Auth для швидкої автентифікації через Google-акаунт;
- Механізм відновлення забутого пароля;
- Захист персональних даних та налаштувань користувачів.
- Створення системи персоналізації, що забезпечить:
- Збереження індивідуальних налаштувань користувача в Firebase Firestore та LocalStorage ;
- Управління профілем користувача;
- Запам'ятовування переліку обраних для моніторингу міст.

**Проектування та реалізація інтерфейсу користувача**

Створення ергономічного та функціонального інтерфейсу, що передбачає:

- Розробку інтуїтивно зрозумілої навігації та логіки взаємодії;
- Реалізацію адаптивного дизайну для різних типів пристроїв (ПК, ноутбуки, планшети, смартфони);
- Впровадження системи тем оформлення (світла/темна) з автоматичною адаптацією всіх елементів інтерфейсу;

- Розробку системи SVG-компонентів для іконок з автоматичною зміною кольору;
- Підтримку технології PWA (Progressive Web App) для можливості встановлення додатку з браузера.

**Інтеграція з метеорологічними сервісами**

Забезпечення взаємодії з зовнішніми API для отримання актуальних погодних даних:

- Налаштування стабільного зв'язку з OpenWeather API;
- Розробка механізму захисту API-ключів через серверні компоненти;
- Реалізація обробки помилок при роботі з зовнішніми сервісами.

**Функціональність для роботи з погодними даними**

- Реалізація комплексу функцій для роботи з метеорологічною інформацією:
- Розробка компонента додавання нових міст з перевіркою на дублювання;
- Створення функціональності оновлення та редагування міст;
- Впровадження механізму видалення міст зі списку моніторингу;
- Розробка системи обмеження максимальної кількості міст для відображення;
- Реалізація автоматичного оновлення погодних даних.

**Додаткові функціональні можливості**

Впровадження додаткових функцій для покращення користувацького досвіду:

- Інтеграція фонового музичного супроводу;
- Розробка механізмів зворотного зв'язку для комунікації з користувачами.

**Технічні вимоги до реалізації**

- Використання сучасного стеку технологій: React, TypeScript, Firebase, Material UI;

- Розробка архітектури на основі контекстів для ефективного управління станом додатку;
- Впровадження ізольованої обробки помилок для забезпечення стабільності системи;
- Розробка механізмів захисту маршрутів для обмеження доступу неавторизованих користувачів;
- Використання принципів чистого коду та стандартів якості для підтримуваності проекту;
- Оптимізація продуктивності через відкладене завантаження компонентів.

**Очікувані результати**

Кінцевим результатом проекту має стати повнофункціональний "Програмне забезпечення системи моніторингу погодних умов", який відповідає всім зазначеним вимогам та забезпечує користувачам зручний інструмент для моніторингу погодних умов. Система повинна функціонувати стабільно, бути захищеною від несанкціонованого доступу, надавати точні та актуальні погодні дані, а також забезпечувати приємний та інтуїтивно зрозумілий користувацький досвід на різних пристроях.

Реалізована система має стати конкурентоспроможним продуктом, що задовольняє сучасні потреби туристів та мандрівників у отриманні метеорологічної інформації, і при цьому відповідає високим стандартам якості програмного забезпечення.

# 2. ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГОЗАБЕЗПЕЧЕННЯ

## 2.1 Логічна модель даних у вигляді ER-діаграми

Модель «сутність-зв'язок» (ER-модель) — модель даних, яка дозволяє описувати концептуальні схеми за допомогою узагальнених конструкцій блоків.

ER-модель — це метамодель даних, тобто засіб опису моделей даних. Існує ряд моделей для представлення знань, але одним з найзручніших інструментів уніфікованого представлення даних, незалежного від програмного забезпечення, що його реалізує, є модель «сутність-зв'язок». Важливим є той факт, що з моделі «сутність-зв'язок» можуть бути породжені всі існуючі моделі даних (ієрархічна, мережева, реляційна, об'єктна), тому вона є найзагальнішою.[19]

**Основні компоненти ER-діаграми**

**Сутності (Entities)** — це головні герої нашої історії. Якщо уявити базу даних як фільм, то сутності — це персонажі, яких ми будемо відстежувати. На діаграмі вони постають у вигляді прямокутників — простих, але змістовних.

**Атрибути (Attributes)** — це характеристики наших героїв. Якщо сутність "Користувач" — це Джеймс Бонд, то атрибути — це його ім'я, номер 007, уміння користуватися гаджетами і любов до мартіні.

**Відношення (Relationships)** — це сюжетні лінії, що пов'язують наших героїв. "Користувач переглядає Фільм", "Студент відвідує Лекцію", "Розробник страждає через Баги" — ось такі історії розповідають нам відношення.

**Ключі (Keys)** — це як паспортні дані для сутностей. Без них неможливо точно визначити, хто є хто. Уявіть світ, де всі люди виглядають однаково! Як знайти потрібну людину? Ось для цього і потрібні ключі.

**Кардинальність (Cardinality)** — це правила взаємодії між сутностями. "У людини дві руки", "у кішки один господар, але у господаря може бути багато кішок", "у студента багато предметів, а предмет вивчають багато студентів" — це і є приклади кардинальності.

**Важливість ER-діаграми в розробці програмного забезпечення**

**1. Ефективне проектування бази даних, або "Сім разів відміряй, один раз відріж"**

Уявіть, що готуєте складну страву без рецепта. Можливо, результат буде їстівним, але... чи буде він таким, як ви очікували? **ER-діаграма — це рецепт для вашої бази даних.** Вона допомагає:

- Вирішити, які інгредієнти (таблиці) потрібні для вашої страви
- Визначити пропорції (структуру полів)
- Встановити порядок змішування (зв'язки між таблицями)
- Уникнути ситуації "ой, забули додати цукор, доведеться все викидати"

Добре спроектована база даних — це як добре організована шафа: все на своєму місці, нічого не випадає на голову, коли відкриваєш дверцята.

**2. Комунікація між стейкхолдерами, або "А він мене зрозумів інакше"**

Бували ситуації, коли пояснювали одне, а люди розуміли зовсім інше? В розробці програмного забезпечення це може коштувати тижнів роботи та тисяч доларів. **ER-діаграма — це універсальна мова, яка дозволяє усім говорити про одне й те саме:**

- Менеджер проекту: "Нам потрібно зберігати дані про користувачів"
- Розробник (без ER-діаграми): "Ок, зроблю таблицю users з полями name і email"
- Менеджер через місяць: "А де зберігаються адреси доставки та історія покупок?"
- Розробник: важкі зітхання та переробка всього з нуля

З ER-діаграмою такі сюрпризи стають рідкістю, бо всі бачать повну картину з самого початку.

**3. Аналіз цілісності та зменшення складності, або "Розплутуємо клубок ниток"**

Створення ER-діаграми — це як розкладання великого плутаного клубка ниток на акуратні моточки. У процесі раптом помічаєте:

- "Чекайте, а чому у нас телефон користувача зберігається в п'яти різних таблицях?"
- "Якщо видалити замовлення, то вся інформація про клієнта теж зникне? Це ж катастрофа!"
- "Як це так вийшло, що товар може належати до категорії, якої не існує?"

Виявити ці проблеми на папері набагато дешевше, ніж коли вони проявляться в живій системі.

**4. Підтримка еволюції програмного забезпечення, або "Карта для майбутніх поколінь"**

Уявіть, що взяли на роботу археолога без карти і сказали: "Десь тут має бути стародавнє місто, копай". Приблизно з такими ж відчуттями зустрічаються розробники, яким доводиться підтримувати програму без документації.

**ER-діаграма — це карта скарбів для майбутніх розробників.** Вона показує, де що заховано і як все пов'язано. З нею:

- Новачки не будуть ходити з виразом "за що мені це все?" на обличчі
- Додавання нових функцій не перетвориться на гру "знайди де вибухне"
- Рефакторинг не буде схожий на розмінування поля з закритими очима

**5. Проміжна ланка між мрією і реальністю**

**ER-діаграма — це міст між "було б круто зробити таке" і "ось конкретний код, який це робить".** Це як перетворення ескізу художника на креслення інженера.

Без цього мосту ви ризикуєте отримати шедевр сучасного мистецтва замість працюючої програми. Красиво, але користуватись неможливо.

**Практичне значення під час розробки**

1. **Зниження вартості змін** — помилка на ER-діаграмі коштує приблизно чашку кави і пів години роботи. Та сама помилка, виявлена після двох місяців розробки, може коштувати як хороший вживаний автомобіль.
2. **Стандартизація підходів до даних** — коли є одна ER-діаграма на проект, не виникає ситуацій, коли один розробник називає це "users", другий — "customers", а третій створює окремі таблиці "clients" та "accounts" для тих самих сутностей.
3. **Спрощення документації** — замість 50-сторінкового документа з описом структури бази даних, ви можете показати одну красиву діаграму. І (увага, магія!) люди її навіть прочитають.
4. **Автоматизація розробки** — сучасні інструменти можуть перетворити гарну ER-діаграму на SQL-скрипти швидше, ніж ви можете сказати "нормалізація даних третьої форми".

Отже, ER-діаграма — це не просто галочка в плані проекту або академічна вправа. Це реальний робочий інструмент, який рятує проекти, нерви та бюджети. Як казав один мудрий розробник: "ER-діаграма варта тисячі рядків коду... які не доведеться переписувати".

**Від теорії до практики: логічна модель системи моніторингу погодних умов**

Тепер, розуміючи важливість та принципи побудови ER-діаграм, можемо перейти до конкретної логічної моделі даних для системи моніторингу погодних умов.

Представлена діаграма на рис.2.1 демонструє структурований підхід до зберігання та організації метеорологічної інформації.

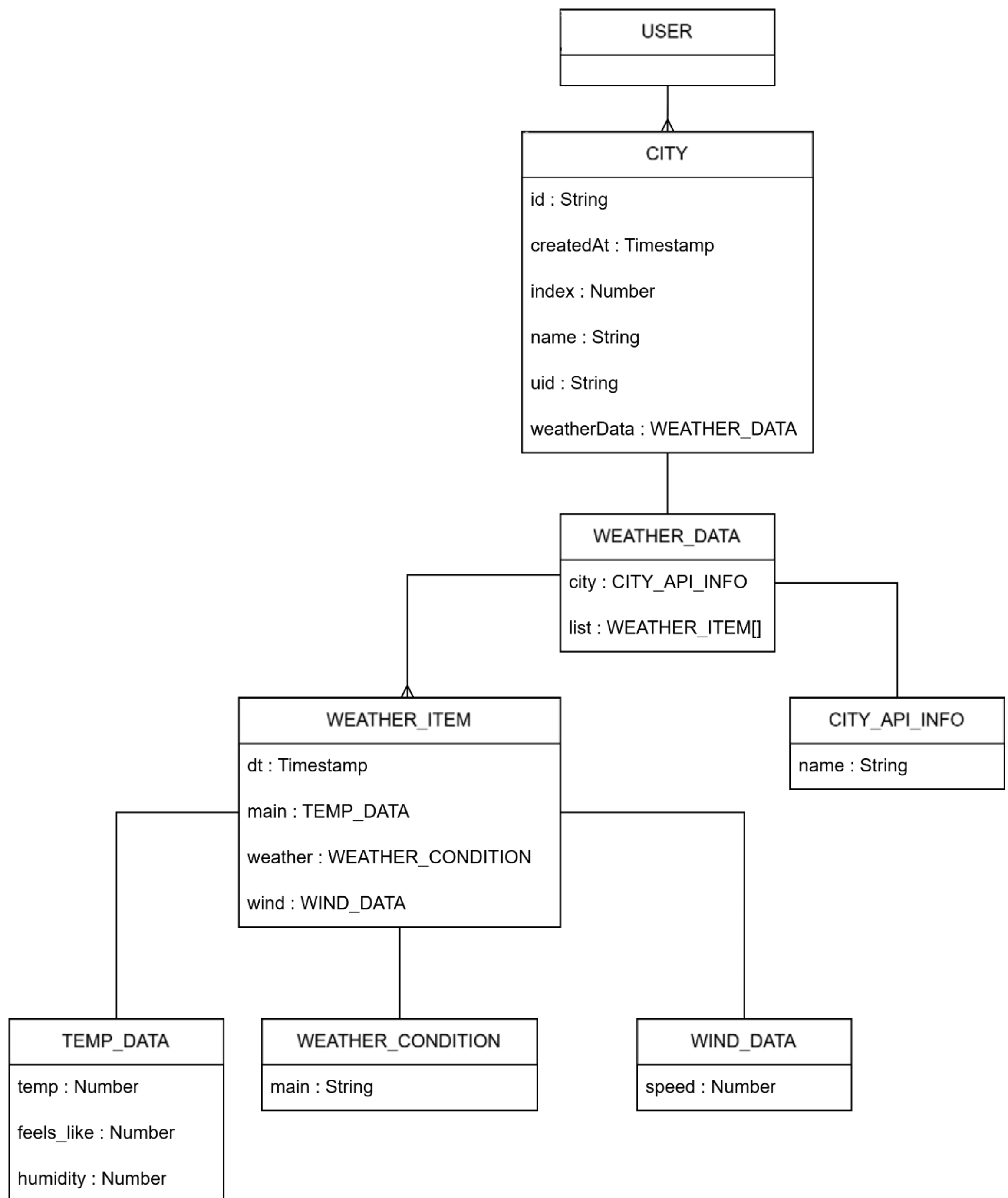


Рис. 2.1Логічна модель у вигляді ER-діаграми

Кожен документ у колекції містить:

- **Метадані міста** (назва, дата створення, індекс, власник).
- **Погодні дані** (поточний стан та прогнози).

**Основні сутності**

Таблиця 2.1

**Опис головної сутності CITY (Місто)**

Головна сутність, що зберігає дані про місто та пов’язану погоду.

| **Поле** | **Тип** | **Опис** |
|---|---|---|
| id | string | Унікальний ідентифікатор (наприклад, XXXXXX123456). |
| createdAt | timestamp | Дата та час додавання міста (напр., 2025-03-29T00:00:00.000Z). |
| index | number | Порядковий номер для сортування (напр., 1). |
| name | string | Назва міста |
| uid | string | ID користувача, який додав місто (напр., "USER123456789"). |
| weatherData | object | Вкладені погодні дані. |

Таблиця 2.2

**Опис сутності WEATHER_DATA (Погодні дані)**

Містить актуальну інформацію про погоду в місті.

| **Поле** | **Тип** | **Опис** |
|---|---|---|
| city | object | Дані міста з погодного API (ID та назва). |
| list | array | Масив прогнозів погоди (зазвичай на 5 днів з кроком 3 години). |

Таблиця 2.3

**Опис сутності CITY_API_INFO (Дані міста з API)**

Частина weatherData.city.

| **Поле** | **Тип** | **Опис** |
|---|---|---|
| name | string | Назва міста в API. |

Таблиця 2.4

**Опис сутності WEATHER_ITEM (Прогноз погоди)**

Елемент масиву weatherData.list.

| **Поле** | **Тип** | **Опис** |
|---|---|---|
| dt | timestamp | Час прогнозу (напр., 1711800000 — Unix-час). |
| main | object | Основні показники (температура, вологість). |
| weather | array | Опис погоди (напр., "Хмарно"). |
| wind | object | Швидкість вітру. |

Таблиця 2.5

**Опис сутності TEMP_DATA (Температура та вологість)**

Частина WEATHER_ITEM.main.

| **Поле** | **Тип** | **Опис** |
|---|---|---|
| temp | number | Температура (°C, напр. 20.0). |
| feels_like | number | Відчувається як (°C, напр. 18.0). |
| humidity | number | Вологість (%, напр. 50). |

Таблиця 2.6

**Опис сутності WEATHER_CONDITION (Стан погоди)**

Частина WEATHER_ITEM.weather[0].

| Поле | Тип | Опис |
|---|---|---|
| main | string | Короткий опис (напр., "Хмарно"). |

Таблиця 2.7

**Опис сутності WIND_DATA (Вітер)**

Частина WEATHER_ITEM.wind.

| Поле | Тип | Опис |
|---|---|---|
| speed | number | Швидкість вітру (м/с, напр. 5.0). |

Представлена логічна модель даних є яскравим прикладом **негеляційної (NoSQL) структури**, що використовується в документо-орієнтованих базах даних, як-от Firebase Firestore. Це підтверджується наступними характеристиками:

**Вкладена ієрархічна структура** — модель демонструє вкладені об'єкти, де сутність CITY містить weatherData як вкладений об'єкт типу WEATHER_DATA, який у свою чергу містить список погодних елементів WEATHER_ITEM.

**Використання складних типів даних** — замість нормалізації інформації у окремих таблицях, модель використовує композитні типи (TEMP_DATA, WEATHER_CONDITION, WIND_DATA) як атрибути батьківських об'єктів.

**Денормалізація для оптимізації запитів** — модель зберігає всі погодні дані разом із містом, що дозволяє отримати всю необхідну інформацію одним запитом без виконання операцій з'єднання, типових для реляційних БД.

Для порівняння, у реляційній моделі, що відповідає третій нормальній формі (3НФ), ця структура була б розділена на окремі таблиці з зовнішніми ключами, де:

- Кожна неключова колонка залежала б тільки від первинного ключа (не від інших неключових колонок);
- Не було б вкладених структур і масивів;
- Зв'язки реалізувалися б через зовнішні ключі з операціями з'єднання.

Обрана нереляційна структура оптимально підходить для додатку моніторингу погоди, забезпечуючи швидкий доступ до даних, гнучкість схеми та ефективне зберігання ієрархічної інформації.

## 2.2 Діаграма класів та кооперацій

Діагра́ма кла́сів — статичне представлення структури моделі в UML. Відображає статичні (декларативні) елементи, такі як: класи, типи даних, їх зміст та відношення. Діаграма класів може містити позначення для пакетів та може містити позначення для вкладених пакетів. Також слугує мостом між концептуальним проектуванням та фактичною реалізацією програмного забезпечення, створюючи чіткий план для процесу розробки.[11]

У контексті проектування об'єктно-орієнтованих систем діаграма класів виконує такі ключові функції:

1. **Візуалізація структури системи** — надає наочне представлення про складові компоненти системи та їхні взаємозв'язки.
2. **Документування архітектури** — створює формальний запис дизайнерських рішень, який можна використовувати протягом всього життєвого циклу розробки.
3. **Комунікація між розробниками** — забезпечує спільну мову для обговорення структурних аспектів системи.
4. **Кодогенерація** — служить основою для автоматичного генерування програмного коду в сучасних середовищах розробки.

**Структура та елементи діаграми класів**

Основними елементами діаграми класів є:

1. **Класи** — абстрактні шаблони, які визначають структуру та поведінку об'єктів. Кожен клас зображується як прямокутник, розділений на три секції: назва класу, атрибути та операції/методи.
2. **Атрибути** — властивості, що характеризують клас. Вони визначають дані, які зберігаються в об'єктах даного класу. На діаграмі атрибути часто позначаються з префіксами видимості: "+" для публічних, "-" для приватних.
3. **Методи/операції** — функції або процедури, які визначають поведінку об'єктів класу. Вони також можуть мати позначення видимості та включати параметри й тип значення, що повертається.
4. **Взаємозв'язки** — відношення між класами, що показують як об'єкти різних класів взаємодіють між собою. Основні типи зв'язків:
   - **Асоціація** — загальний зв'язок між класами;
   - **Агрегація** — зв'язок "частина-ціле", де частини можуть існувати незалежно від цілого;
   - **Композиція** — сильніша форма агрегації, де частини не можуть існувати окремо від цілого;
   - **Успадкування/Генералізація** — зв'язок, що показує спеціалізацію класів;
   - **Реалізація** — зв'язок між інтерфейсом та класом, що його реалізує.

**Діаграми кооперацій: динамічний аспект взаємодії**

Якщо діаграма класів відображає статичну структуру системи, то діаграми кооперацій (або взаємодії) показують динамічні аспекти взаємодії між об'єктами в процесі виконання певних сценаріїв. Вони ілюструють, як об'єкти співпрацюють для виконання конкретних завдань, демонструючи послідовність обміну повідомленнями та потоки даних.

Діаграми кооперацій дозволяють:

- Візуалізувати алгоритми взаємодії між компонентами системи;
- Перевіряти коректність проектування класів через аналіз їх взаємодії;
- Ідентифікувати можливі проблеми у взаємодії та оптимізувати архітектуру.

**Аналіз представлених діаграм для системи моніторингу погодних умов**

Представлені діаграми демонструють чітку об'єктно-орієнтовану архітектуру системи моніторингу погодних умов. Розглянемо ключові компоненти:

Почнемо з діаграми класів зображену на рис.2.2.

Рис. 2.2 Діаграма класів

**Клас "Мандрівник"** виступає у ролі користувача системи і містить:

- Атрибути: ім'я, місто;
- Методи: вибратиМісто(), переглянутиПогодніДані().

Цей клас представляє основного актора, який взаємодіє з системою моніторингу погоди.

**Клас "Моніторинг погоди"** є центральним компонентом системи:

- Атрибути: список міст, погодніДані;
- Методи: обробитиЗапити(), відобразитиПогодніДані(), оновитиПогодніДані().

Він відповідає за управління даними про погоду та координацію взаємодії між користувачем і зовнішнім API.

**Клас "Погодні дані"** відображає інформаційну модель метеорологічних показників:

- Атрибути: температура, описПогоди, вологість, швидкістьВітру.

Це структура даних, яка зберігає актуальну метеорологічну інформацію.

**Клас "API сервіс"** представляє інтерфейс для взаємодії з зовнішнім джерелом даних:

- Методи: обробитиЗапити(), повернутиПогодніДані().

Цей клас абстрагує комунікацію з метеорологічним API, інкапсулюючи деталі HTTP-запитів.

**Взаємозв'язки між класами**

- **Мандрівник — Моніторинг погоди**: асоціація з кардинальністю 1:1, що вказує на взаємодію користувача з системою.
- **Моніторинг погоди — Погодні дані**: композиція з кардинальністю 1:0..*, показуючи, що система містить колекцію погодних даних для різних міст.
- **Моніторинг погоди — API сервіс**: асоціація з кардинальністю 1:1, що відображає використання API для отримання даних.

- **API сервіс — Погодні дані**: зв'язок "повертає" з кардинальністю 1:1, демонструючи, що API надає дані у відповідь на запити.

Поняття кооперації є одним з фундаментальних понять в мові UML. Воно призначене для позначення множини об'єктів, що взаємодіють для досягнення певної мети. Метою кооперації є специфікувати особливості реалізації окремих найбільш значущих операцій в системі. Кооперація визначає структуру поведінки системи в термінах взаємодії учасників цієї кооперації. [13]

Діаграма кооперацій на рис.2.3 додатково ілюструє логіку взаємодії компонентів системи:

1. Мандрівник ініціює процес вибору міста та запиту погодних даних
2. Моніторинг погоди обробляє запит та звертається до API сервісу
3. API сервіс повертає актуальні погодні дані
4. Моніторинг погоди обробляє отримані дані та відображає їх користувачу

Такий підхід до моделювання дозволяє чітко відокремити обов'язки між компонентами системи, забезпечуючи високу модульність та гнучкість архітектури.

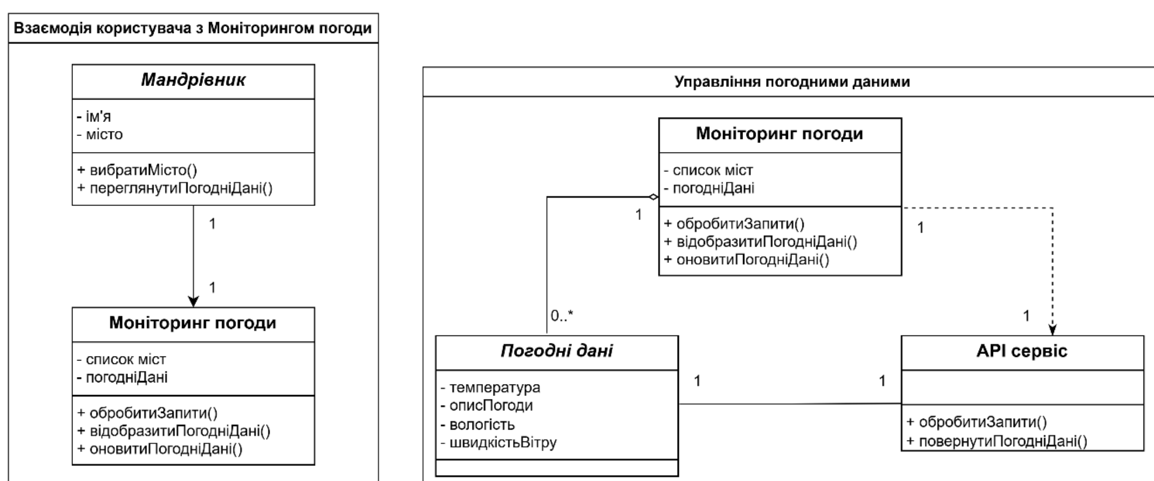


Рис. 2.3 Діаграма кооперацій

Представлені діаграми в комплексі надають повноцінне розуміння як статичної структури, так і динамічних аспектів взаємодії в системі моніторингу погодних умов, створюючи міцну основу для подальшої розробки.

## 2.3 Діаграма пакетів

Діаграми пакетів уніфікованої мови моделювання (UML) відображають залежності між пакетами, з яких і складається модель. Пакет (package) — елемент моделі, який використовують для групування інших елементів моделі. Елементи моделі, які входять до складу певного пакета, називаються членами пакета. Пакет володіє усіма своїми членами. [14]

Діаграма пакетів представляє логічне групування компонентів системи, відображаючи високорівневу архітектуру програмного забезпечення.

На представленій діаграмі на рис.2.4 система моніторингу погодних умов розділена на два основні пакети: "Data Management" та "User Interface".

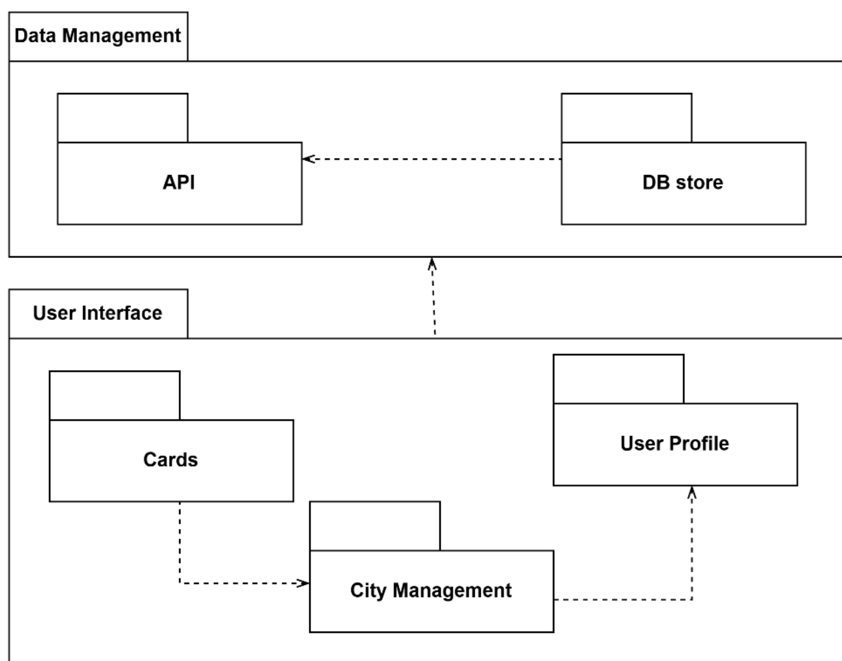


Рис. 2.4 Діаграма пакетів

Пакет "Data Management" інкапсулює компоненти, відповідальні за взаємодію з даними: "API" для отримання метеорологічної інформації та "DB store" для збереження користувацьких налаштувань.

Пакет "User Interface" містить компоненти, орієнтовані на взаємодію з користувачем: "Cards" для відображення погодних даних, "City Management" для управління списком міст та "User Profile" для роботи з профілем користувача.

Зв'язки між пакетами ілюструють потоки даних та функціональні залежності, демонструючи чітке відокремлення рівня даних від презентаційного рівня, що забезпечує модульність та полегшує подальшу підтримку системи.

## 2.4 Діаграма компонентів

Діаграма компонентів — в UML, діаграма, на якій відображаються компоненти, залежності та зв'язки між ними. Діаграма компонент відображає залежності між компонентами програмного забезпечення, включаючи компоненти вихідних кодів, бінарні компоненти, та компоненти, що можуть виконуватись. [12]

Діаграма компонентів візуалізує модульну структуру системи моніторингу погодних умов, яка розділена на чотири основні підсистеми, зображено на рис.2.5.

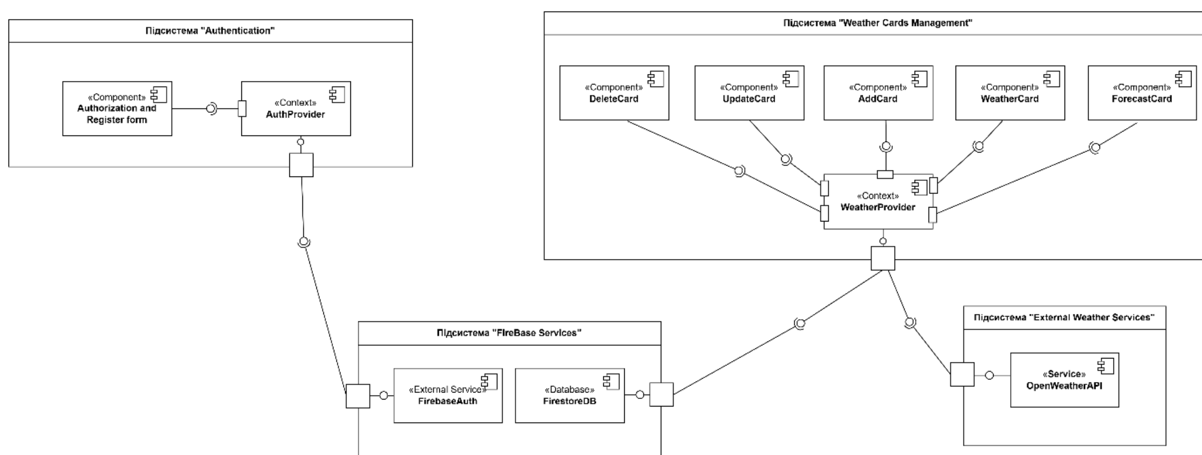


Рис. 2.5 Підсистема "Authentication"

Підсистема "Authentication" забезпечує функціонал автентифікації через компоненти форми реєстрації та AuthProvider. Підсистема "Weather Cards Management" є ядром додатку, що містить компоненти для маніпуляції погодними

картками (DeleteCard, UpdateCard, AddCard, WeatherCard, ForecastCard) та контекст WeatherProvider для управління даними. Підсистеми зовнішніх сервісів включають "Firebase Services" з компонентами FirebaseAuth та FirestoreDB для авторизації та зберігання даних, а також "External Weather Services" з OpenWeatherAPI для отримання метеорологічної інформації.

Додаткова підсистема "User Personalization" на рис.2.6 відповідає за персоналізацію досвіду користувача через компоненти ThemeSwitch для зміни теми інтерфейсу з використанням LocalStorage та MusicSystem для фонового музичного супроводу через інтеграцію з YouTubeAPI.

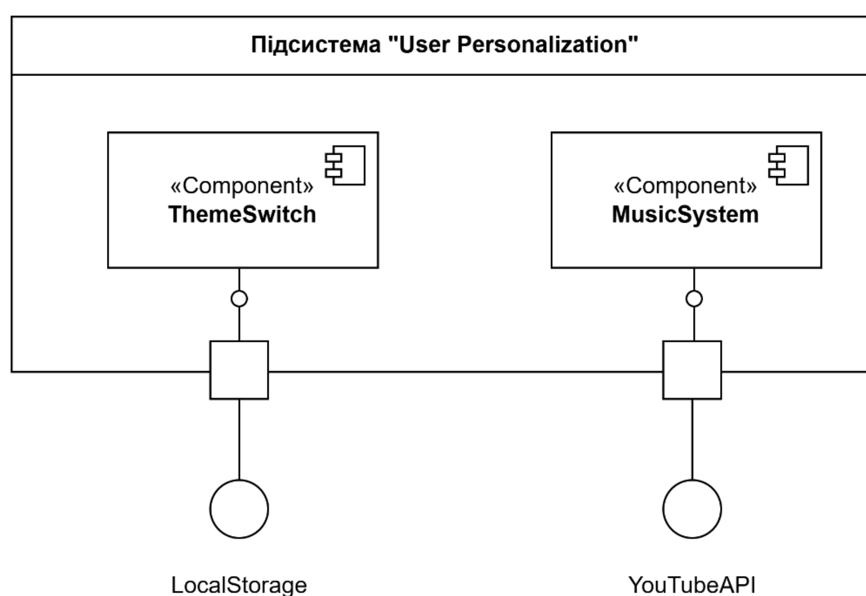


Рис. 2.6 Підсистема "User Personalization"

Взаємозв'язки між компонентами демонструють чітку модульну архітектуру з розділенням функціональних обов'язків.

# 3. РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 3.1 Система управління інформаційною базою

Вибір системи управління базами даних (СУБД) є одним із ключових архітектурних рішень, що визначає характеристики, продуктивність та масштабованість програмного забезпечення.

Система управління базами даних (СУБД) – це комплекс програмних і лінгвістичних засобів, що призначені для створення, зберігання, управління та використання баз даних. Він забезпечує ефективне та надійне зберігання даних, а також надає користувачам зручний інтерфейс для роботи з ним. Роль СУБД в управлінні даними полягає в забезпеченні структурованого доступу до інформації, мінімізації дублювання даних, забезпеченні цілісності та безпеки даних.[21]

Для системи моніторингу погодних умов, яка передбачає роботу з динамічними даними, підтримку користувацьких профілів та забезпечення швидкого доступу до інформації, необхідно врахувати такі критерії:

**Модель даних** — відповідність структури СУБД логічній моделі даних проекту, можливість ефективного представлення ієрархічних зв'язків між сутностями (користувач → місто → погодні дані).

**Продуктивність** — швидкість обробки запитів, особливо при одночасному доступі багатьох користувачів до погодних даних у реальному часі.

**Масштабованість** — здатність системи зростати разом із збільшенням кількості користувачів та обсягу даних без суттєвого падіння продуктивності.

**Надійність та доступність** — гарантія збереження даних та мінімальний час простою, що критично для сервісу, який повинен надавати актуальну інформацію про погоду.

**Безпека** — захист персональних даних користувачів та можливість гнучкого налаштування прав доступу.

**Інтеграція з іншими сервісами** — простота взаємодії з системами автентифікації, API погодних сервісів та іншими компонентами інфраструктури.

**Складність адміністрування** — вимоги до технічної експертизи для налаштування та підтримки СУБД.

**Вартість** — сукупна вартість володіння, включаючи ліцензії, інфраструктуру та людські ресурси для підтримки.

**Порівняльний аналіз альтернатив**

**Традиційні реляційні СУБД (MySQL, PostgreSQL)**

**Переваги:**

- Високий рівень узгодженості даних завдяки підтримці ACID-транзакцій;
- Добре стандартизована мова запитів SQL;
- Потужні механізми для складних аналітичних запитів;
- Великий вибір інструментів для адміністрування та моніторингу.

**Недоліки:**

- Обмежена горизонтальна масштабованість без додаткових рішень;
- Складнощі при роботі з ієрархічними та напівструктурованими даними;
- Потребують окремого управління інфраструктурою та налаштування;
- Менш гнучкі при зміні схеми даних.

**Документо-орієнтовані NoSQL СУБД (MongoDB)**

**Переваги:**

- Гнучка схема даних, що полегшує еволюцію моделі;
- Висока продуктивність при операціях читання;
- Ефективна робота з ієрархічними документами.

**Недоліки:**

- Обмежена підтримка складних транзакцій, що охоплюють кілька документів;

- Менша узгодженість даних порівняно з реляційними СУБД;
- Відносно висока вартість для великих проектів;
- Потреба у власній інфраструктурі та налаштуванні.

**Хмарні СУБД та Backend-as-a-Service (Firebase Firestore)**

**Переваги:**

- Автоматичне масштабування без необхідності управління інфраструктурою;
- Інтеграція з сервісами автентифікації та хостингу в єдиній екосистемі;
- Підтримка реального часу з автоматичною синхронізацією даних;
- Вбудовані механізми безпеки та контролю доступу;
- Можливість роботи в офлайн-режимі з подальшою синхронізацією.

**Недоліки:**

- Обмеження на гнучкість та типи запитів порівняно з традиційними СУБД;
- Потенційно вища вартість при значному масштабуванні;
- Залежність від сторонньої інфраструктури та можливі обмеження сервісу;
- Менша кількість інструментів для складного аналізу даних.

**Обґрунтування вибору Firebase Firestore**

Для системи моніторингу погодних умов оптимальним вибором є Firebase Firestore як основна СУБД з таких причин:

**Ідеальна відповідність моделі даних** — документо-орієнтована структура Firestore природно узгоджується з логічною моделлю системи, де інформація про міста та погодні дані представлена як ієрархічна структура документів з вкладеними об'єктами.

**Інтегрована екосистема** — Firebase надає не лише СУБД, але й комплексне рішення з автентифікацією, хостингом, функціями на стороні сервера та аналітикою, що значно прискорює процес розробки та розгортання.

**Реальний час та синхронізація** — вбудовані механізми підписок на зміни даних дозволяють автоматично оновлювати інтерфейс користувача при зміні погодних даних без додаткових запитів.

**Безпека з мінімальними зусиллями** — Firebase Security Rules дозволяють декларативно встановлювати права доступу до даних, забезпечуючи захист інформації користувачів без написання власного серверного коду.

**Офлайн-підтримка** — критична функція для мобільних користувачів, які можуть мати нестабільне підключення до мережі, але потребують доступу до збережених погодних даних.

**Зниження операційної складності** — відсутність необхідності налаштовувати, масштабувати та підтримувати власну інфраструктуру бази даних дозволяє зосередитись на розробці основної функціональності.

**Передбачувана модель витрат** — оплата за фактичне використання ресурсів з безкоштовним стартовим планом дозволяє оптимізувати витрати на ранніх етапах розвитку проекту.

**Швидкість розгортання** — мінімальна початкова конфігурація та наявність SDK для різних платформ значно зменшують time-to-market для системи.

Для зберігання більших обсягів історичних погодних даних або створення складних аналітичних звітів можливе додаткове використання Firebase BigQuery, що дозволить проводити більш комплексний аналіз без зміни основної архітектури системи.

Варто зазначити, що вибір Firebase Firestore також узгоджується з загальною архітектурою додатку на React з TypeScript, оскільки існує добре документоване та підтримуване SDK для таких технологій, а також спеціалізовані бібліотеки типу React Firebase Hooks, що спрощують інтеграцію.

У підсумку, хоча кожна з розглянутих СУБД має свої переваги, Firebase Firestore надає оптимальний баланс між функціональністю, простотою розробки та швидкістю виведення продукту на ринок для системи моніторингу погодних

умов, особливо враховуючи вимоги до реального часу, персоналізації та мультиплатформності.

Система моніторингу погодних умов використовує хмарну платформу Firebase як основу інформаційної бази даних, що забезпечує надійність, масштабованість та швидкий доступ до даних. Архітектура інформаційної бази розроблена з урахуванням особливостей роботи з метеорологічними даними та потреб персоналізації користувацького досвіду.

Впроваджена система управління інформаційною базою забезпечує надійне зберігання даних, швидкий доступ та захист інформації, що є критично важливим для стабільного функціонування системи моніторингу погодних умов.

## 3.2 Розробка інформаційної бази

Розробка інформаційної бази для системи моніторингу погодних умов проходила в кілька послідовних етапів з урахуванням вимог до функціональності, продуктивності та безпеки даних.

Спочатку було проведено аналіз вимог до зберігання даних, який виявив необхідність підтримки персоналізованих списків міст для кожного користувача, зберігання актуальних метеорологічних показників та можливості швидкого доступу до інформації. На основі цих вимог була спроектована структура документів Firestore з визначенням основних колекцій (users, cities, weather_data) та їх взаємозв'язків.

Наступним кроком стала розробка схеми даних для кожної колекції з деталізацією полів та їх типів.

```
"cities": {
 "XXXXXX123456": {
  // Метадані
  "createdAt": "2025-03-29T00:00:00.000Z", // timestamp
  "index": 1, // number (для сортування)
  "name": "Назва міста", // string (дубльоване)
  "uid": "USER123456789", // string (зв'язок з user)
  // Погодні дані
  "weatherData": {
```

```
"city": {
  "name": "Назва міста" // string
},
"list": [
  {
    "dt": 1711800000, // timestamp
    "main": {
      "temp": 20.0, // number (°C)
      "feels_like": 18.0, // number (°C)
      "humidity": 50 // number (%)
    },
    "weather": [
      {
        "main": "Хмарно" // string
      }
    ],
    "wind": {
      "speed": 5.0 // number (м/с)
    }
  }
]
```


Основними колекціями в системі є:

1. **users** — зберігає інформацію про користувачів системи:
   - Унікальний ідентифікатор (uid);
   - Персональні дані (ім'я, email);
   - Налаштування профілю;
   - Преференції інтерфейсу (обрана тема);
   - Часовий пояс.
2. **cities** — містить інформацію про міста, додані користувачами для моніторингу:
   - Унікальний ідентифікатор міста
   - Назва міста
   - Посилання на користувача (uid)
   - Часова мітка створення
   - Порядковий індекс для сортування
   - Метеорологічні дані (вкладена структура)

3. **weather_data** — кешовані погодні дані для оптимізації запитів:
   - Ідентифікатор міста
   - Часова мітка останнього оновлення
   - Поточні погодні умови
   - Короткотерміновий прогноз

Для колекції cities була впроваджена вкладена структура для зберігання повних погодних даних без необхідності додаткових запитів, що суттєво прискорило візуалізацію інформації на клієнтській стороні.

Важливим аспектом розробки стало впровадження правил безпеки Firebase, які гарантують, що користувачі мають доступ лише до власних даних. Правила були налаштовані з використанням умовних виразів та перевірки автентифікаційних токенів.

Фінальним етапом стало тестування інформаційної бази з перевіркою швидкодії, цілісності даних та коректності роботи правил доступу в різних сценаріях використання системи.

## 3.3 Вибір інструментарію для створення прикладного програмного забезпечення

Вибір технологічного стеку для розробки системи моніторингу погодних умов ґрунтувався на кількох ключових критеріях:

- Продуктивність і швидкодія відображення метеорологічних даних;
- Підтримка адаптивного дизайну для різних пристроїв;
- Зручність розробки та підтримки коду;
- Можливість створення інтерактивних компонентів інтерфейсу;
- Інтеграція з хмарними сервісами та API;
- Масштабованість для підтримки зростання кількості користувачів;
- Доступність інструментів для тестування та забезпечення якості коду.

**Основні технології фронтенд-розробки**

**React з TypeScript**

Для створення клієнтської частини системи обрано React — відкрита JavaScript бібліотека для створення інтерфейсів користувача, яка покликана вирішувати проблеми часткового оновлення вмісту вебсторінки, з якими стикаються в розробці односторінкових застосунків.[5]

Ця технологія має ряд переваг, що відповідають вимогам проекту:

- **Компонентний підхід** дозволяє створювати модульну архітектуру з перевикористовуваними блоками інтерфейсу (WeatherCard, AddCard, UpdateCard);
- **Віртуальний DOM** забезпечує високу продуктивність при оновленні інтерфейсу з динамічними погодними даними;
- **React Hooks** спрощують управління станом компонентів та роботу з життєвим циклом;
- **Context API** надає механізм для передачі даних через дерево компонентів без пропс-дрілінгу, що критично для реалізації WeatherContext, AuthContext та ThemeContext.

Використання TypeScript як надбудови над JavaScript додає переваги статичної типізації:

- Раннє виявлення помилок ще на етапі компіляції;
- Покращена підтримка IDE з автодоповненням та рефакторингом;
- Чітке визначення інтерфейсів даних (WeatherData, CityInfo);
- Полегшення командної розробки через самодокументований код.

**SCSS для стилізації**

Sass (Syntactically Awesome Style Sheets) — один із найпопулярніших таких інструментів. У нього додано можливості, які поки що недоступні в CSS. Ці можливості спрощують обслуговування сайтів та застосунків. [7]

Основним інструментом для стилізації компонентів у системі обрано SCSS (Sass), що надає суттєві переваги порівняно з чистим CSS:

- **Вкладена структура селекторів** для більш логічної організації стилів;
- **Змінні** для уніфікації кольорів, розмірів та інших повторюваних значень;
- **Міксини** для повторного використання блоків стилів;
- **Функції** для динамічних обчислень значень;
- **Модульна організація** стилів з імпортами окремих файлів;
- **Математичні операції** для гнучкого розрахунку розмірів та позиціонування.

Цей підхід дозволив створити унікальний, кастомізований дизайн для основних компонентів системи (WeatherCard, MainCards, ForecastCard, тощо) з високим рівнем контролю над візуальними деталями.

**Bootstrap**

Bootstrap – це відкритий та безкоштовний HTML, CSS та JS фреймворк для швидкої верстки адаптивних дизайнів сайтів та WEB-додатків. Найчастіше його використовують для фронтенд розробки сайтів та інтерфейсів адмінпанелей. Цей фреймворк сильно полегшує розробку сайтів завдяки наявності шаблонів дизайну на основі HTML і CSS для типографіки, форм, кнопок, таблиць, навігації, плагінів JavaScript та ін. Все це також дає вам можливість легко створювати адаптивні дизайни. [2]

Для забезпечення базової адаптивності та прискорення розробки стандартних елементів інтерфейсу використовувався Bootstrap, який надав:

- **Гнучку сітку** для організації елементів на сторінці;
- **Готові компоненти** для типових елементів інтерфейсу;
- **Адаптивні утиліти** для різних розмірів екранів;
- **Крос-браузерну сумісність** без додаткових зусиль.

**Material UI (обмежене використання)**

Material UI — це бібліотека компонентів React з відкритим кодом, яка реалізує Material Design від Google. Вона є комплексною та може використовуватися у продакшені одразу після встановлення. Вона містить повну колекцію попередньо створених компонентів, готових до використання у продакшені одразу після встановлення, а також пропонує набір опцій налаштування, які дозволяють легко впроваджувати власну систему дизайну поверх наших компонентів. [4]

Material UI використовувався вибірково, переважно для специфічних сторінок та компонентів:

- **LoginPage** — форми автентифікації та реєстрації;
- **ProfilePage** — інтерфейс управління профілем;
- **UserMenu** — випадаюче меню користувача.

Цей підхід дозволив поєднати переваги кастомізованого дизайну через SCSS для основних компонентів з готовими рішеннями Material UI для стандартизованих елементів інтерфейсу.

**React Router**

React Router — це API для веб-додатків, які використовують бібліотеку React. Більшість поточних кодів написано з використанням React Router 3, хоча вже була випущена версія 4. Маршрутизатор React використовує динамічну маршрутизацію (тобто маршрутизацію, яка здійснюється під час рендерингу вашої програми, а не в конфігурації додатка). [6]

Для організації навігації між різними сторінками додатку (LoginPage, HomePage, ProfilePage) використовується React Router, що забезпечує:

- Декларативне визначення маршрутів;
- Параметризовані URL;

- Захищені маршрути (ProtectedRoute) для автентифікованих користувачів;
- Програмну навігацію через useNavigate hook.

**Хмарна платформа та бекенд-сервіси**

Firebase – це платформа розробки мобільних застосунків із величезним функціоналом. Починалася вона як стартап, а сьогодні її використовують під час розроблення найкращих кросплатформних застосунків. Головна перевага платформи в тому, що вона дає змогу розробнику не відволікатися на створення бекенда, тобто прихованої від користувача програмної частини проєкту, наприклад, серверного коду. І це спрощує і прискорює створення мобільних застосунків, дає змогу повністю зосередитися саме на UX/UI, тобто на користувацькому інтерфейсі та досвіді. [3]

Екосистема Firebase обрана як комплексне рішення для багатьох аспектів додатку:

1. **Firebase Authentication** для управління користувачами:
   - Підтримка входу через email/пароль та Google;
   - Безпечне зберігання облікових даних;
   - Управління сесіями та токенами.
2. **Firestore** як NoSQL база даних:
   - Гнучка документо-орієнтована структура;
   - Підтримка реального часу з автоматичною синхронізацією;
   - Офлайн-режим для роботи без підключення.
3. **Firebase Hosting** для розгортання веб-додатку:
   - Швидка доставка контенту через CDN;
   - Автоматичні SSL-сертифікати;
   - Інтеграція з процесом збірки та CI/CD.

**Інструменти розробки та оптимізації**

Vite — це сучасний, блискавично швидкий інструмент для створення скелетів і групування проєктів, який швидко стає популярним завдяки майже миттєвій компіляції коду та швидкій гарячій заміні модулів. У даній статті ми з вами розберемось, що ж дозволяє йому бути таким продуктивним. [20]

Для збірки та розробки проекту використовується Vite замість традиційного Create React App, що надає:

- Надшвидкий старт сервера розробки завдяки нативним ES-модулям;
- Ефективну гарячу заміну модулів (HMR);
- Оптимізовану продакшн-збірку з поділом коду.

**ESLint та Prettier**

- Для забезпечення якості коду та узгодженого стилю застосовуються:
- ESLint з набором правил для TypeScript та React;
- Prettier для автоматичного форматування;
- Преткомміт-хуки для перевірки коду перед комітом.

**Інтеграції із зовнішніми сервісами**

**OpenWeather API**

Для отримання метеорологічних даних обрано OpenWeather API через:

- Широке покриття глобальних локацій;
- Доступність різних типів погодних даних;
- Надійність та стабільність сервісу;
- Зрозумілу документацію та формат відповідей.

**YouTube API**

Для реалізації фонового музичного супроводу використовується інтеграція з YouTube API, що дозволяє:

- Вбудовувати відтворення аудіоконтенту;

**Висновки щодо обраного інструментарію**

Обраний технологічний стек забезпечує оптимальний баланс між продуктивністю, зручністю розробки та користувацьким досвідом. Комбінація React з TypeScript як основи, SCSS для кастомізованої стилізації більшості

компонентів, Bootstrap для структурної організації та обмежене використання Material UI для специфічних елементів інтерфейсу створює гнучку і потужну основу для розробки.

Такий підхід дозволяє реалізувати унікальний, брендований дизайн основних компонентів, зберігаючи при цьому узгодженість інтерфейсу та адаптивність на різних пристроях. Використання Firebase як бекенду забезпечує швидке розгортання та масштабування, а інтеграції з зовнішніми API розширюють функціональні можливості системи.

## 3.4 Алгоритмізація та програмування програмних модулів

Архітектура системи моніторингу погодних умов побудована за принципом компонентної організації з використанням контекстного підходу для управління станом додатку. Це дозволяє забезпечити високий рівень модульності, повторного використання коду та чіткого розділення відповідальності між компонентами.

Основу архітектури складає трирівнева модель:

- Рівень представлення (компоненти інтерфейсу)
- Рівень бізнес-логіки (контексти, сервіси)
- Рівень даних (API-інтеграції, Firebase)

**Контекстний підхід до управління станом**

Центральним елементом архітектури є система контекстів, що забезпечує єдине джерело істини для різних аспектів додатку. WeatherContext відповідає за управління погодними даними та містить алгоритми для взаємодії з API погоди, обробки та відображення метеорологічної інформації.

Реалізація WeatherContext включає функціональність для додавання, оновлення та видалення міст, а також оновлення погодних даних. Алгоритм додавання міста передбачає перевірку на дублікати, валідацію вхідних даних та оновлення стану додатку після успішної операції. Детальну реалізацію описано в додатку Б.

AuthContext інкапсулює логіку автентифікації та авторизації, забезпечуючи захищений доступ до функціональності додатку. Важливим аспектом є алгоритм обробки помилок автентифікації з трансформацією технічних повідомлень у зрозумілий для користувача формат. Детальну реалізацію описано в додатку А.

ThemeContext реалізує механізм управління темою інтерфейсу з автоматичним застосуванням стильових змін до всіх компонентів системи, включаючи динамічну адаптацію SVG-іконок. Детальну реалізацію описано в додатку В.

JoyrideContext є важливим компонентом системи, який відповідає за інтерактивне навчальне керівництво для нових користувачів. Забезпечує покрокове ознайомлення з основними функціями додатку, допомагаючи користувачам швидко освоїти інтерфейс. Контекст керує станом навчального туру, зберігає інформацію про пройдені кроки та надає можливість відкласти або призупинити керівництво. Детальну реалізацію описано в додатку Г.

**Обробка погодних даних**

Алгоритм отримання та обробки погодних даних реалізовано з урахуванням оптимізації мережевих запитів та кешування результатів. Процес починається з валідації назви міста, формування параметрів запиту та надсилання його до OpenWeather API. Отримана відповідь проходить декілька етапів обробки: нормалізація структури, перетворення одиниць вимірювання, форматування дати та часу.

Важливою частиною алгоритму є обробка помилкових відповідей від API та реалізація механізму повторних спроб з експоненційною затримкою в разі тимчасових мережевих проблем.

**Модульна система компонентів**

Програмування інтерфейсу користувача базується на детальній декомпозиції функціональності на незалежні компоненти. Кожен компонент має чітко визначену відповідальність та реалізує власні алгоритми для відображення та обробки даних.

Компонент WeatherCard відповідає за візуалізацію погодних даних для конкретного міста. Він реалізує алгоритми форматування температури, визначення піктограми погоди на основі коду умов та інтерактивних переходів між станами картки.

Механізм взаємодії між компонентами забезпечується через систему пропсів та контекстів, що мінімізує зв'язки між модулями та спрощує підтримку коду.

**Ізольована обробка помилок**

Особлива увага в системі приділена алгоритмам обробки помилок. Кожен компонент, що взаємодіє з зовнішніми сервісами, реалізує власну ізольовану обробку виняткових ситуацій, що запобігає каскадному поширенню помилок.

В компонентах AddCard та UpdateCard помилки локалізовані на рівні окремої картки, що дозволяє іншим карткам функціонувати без переривання навіть при збоях в одному з компонентів. Алгоритм передбачає перехоплення помилок на рівні форм вводу, їх аналіз та відображення відповідних повідомлень користувачу.

Такий підхід до алгоритмізації та програмування модулів забезпечує надійність, масштабованість та зручність підтримки системи моніторингу погодних умов.

Для ілюстрації описаних принципів розглянемо один із ключових алгоритмів системи — додавання нового міста для моніторингу. Цей процес є фундаментальним для роботи додатку, оскільки дозволяє користувачам персоналізувати набір відстежуваних локацій.

Алгоритм додавання міста представлено на рис.3.1 у вигляді блок-схеми, що демонструє логічну послідовність дій та прийняття рішень:

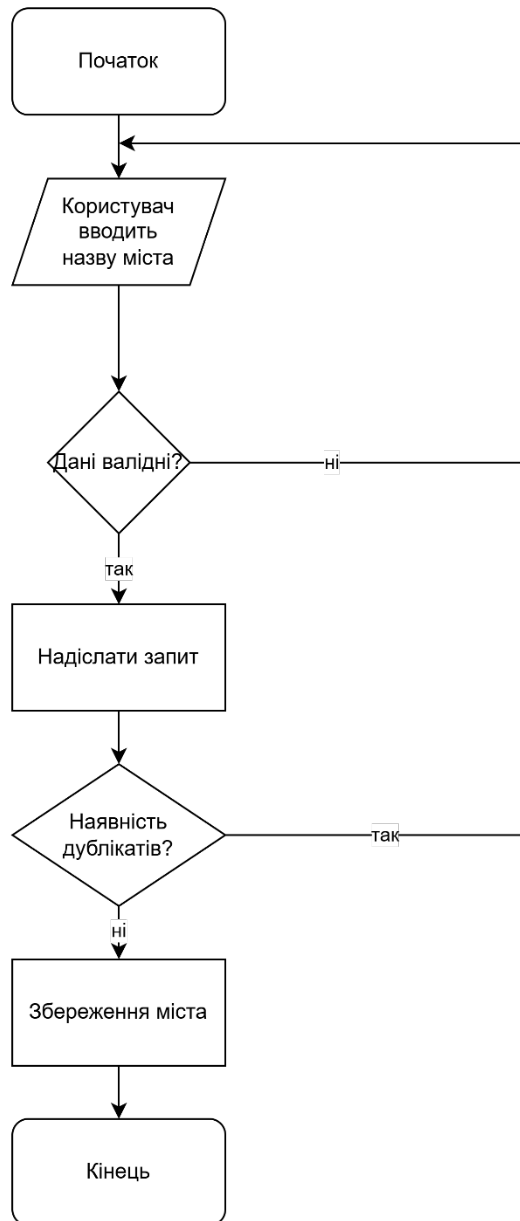


Рис. 3.1 Блок схема додавання погоди

Представлено послідовність кроків від введення назви міста користувачем до збереження даних у базі. Особливу увагу приділено валідації даних та перевірці на дублікати, що забезпечує цілісність даних у системі.

Цей алгоритм реалізовано в коді компонента AddCard:

```
const handleSubmit = async (e) => {
  e.preventDefault();
  setIsLoading(true);
  setLocalError('');
```

```
  if (!cityName.trim()) {
    setLocalError('Назва міста не може бути порожньою');
    setIsLoading(false);
    return;
  }

  const result = await addCity(cityName);

  if (result.error) {
    setLocalError(result.error);
  } else {
    setCityName('');
  }
  setIsLoading(false);
};
```

Наведений фрагмент коду демонструє практичну реалізацію алгоритму з обробкою різних сценаріїв виконання. Код відображає ключові етапи, зазначені на блок-схемі:

- Валідація введених даних через перевірку if (!cityName.trim());
- Виклик функції addCity() , яка здійснює перевірку на дублікати та взаємодію з API;
- Обробка результату операції з відповідним оновленням інтерфейсу.

Такий структурований підхід до реалізації алгоритмів забезпечує стабільність та прогнозованість поведінки системи, навіть при роботі з неконтрольованими зовнішніми ресурсами, такими як погодні API.

# 4. РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇСИСТЕМИ

## 4.1 Тестування системи

Тестування системи моніторингу погодних умов базувалося на практичному підході із залученням реальних користувачів. Такий метод дозволив отримати безпосередній зворотний зв'язок щодо зручності використання, функціональності та загального враження від додатку. На відміну від автоматизованого тестування, ручне тестування з реальними користувачами надало більш глибоке розуміння того, як система сприймається цільовою аудиторією.

Процес тестування проводився в кілька етапів, починаючи від базової перевірки функціональності розробником та поступово залучаючи ширше коло тестувальників з різним рівнем технічної підготовки та потребами.

**Етапи користувацького тестування**

Початкова фаза тестування включала внутрішню перевірку основних функцій додатку розробником. Цей етап дозволив виявити та виправити очевидні помилки в коді та лозіці роботи системи перед залученням зовнішніх тестувальників.

Наступним кроком було тестування з обмеженою групою користувачів, які мали технічне розуміння та могли надати конструктивний зворотний зв'язок щодо функціональності системи. Під час цього етапу було зібрано детальні звіти про проблеми та запропоновано потенційні покращення інтерфейсу.

Фінальний етап включав відкрите тестування із залученням ширшого кола потенційних користувачів різного віку та з різним досвідом роботи з подібними додатками. Цей етап був особливо цінним для оцінки загальної зручності використання та інтуїтивності інтерфейсу.

**Основні напрямки тестування**

Функціональне тестування охоплювало перевірку всіх ключових можливостей системи, зокрема правильність роботи механізмів автентифікації, точність відображення погодних даних, коректність процесів додавання, оновлення та видалення міст.

Тестування користувацького досвіду фокусувалося на оцінці простоти навігації, зрозумілості інтерфейсу та загальної задоволеності користувачів. Тестувальникам пропонувалося виконати типові сценарії використання системи без попередніх інструкцій, що дозволило виявити неочевидні проблеми з інтуїтивним розумінням функцій додатку.

Перевірка адаптивності інтерфейсу проводилася на різних пристроях (смартфонах, планшетах, ноутбуках) для підтвердження коректності відображення компонентів та їх функціональності в різних умовах використання.

Важливою складовою тестування була оцінка стабільності роботи в різних браузерах та при різних умовах підключення до інтернету, що дозволило виявити та усунути проблеми із сумісністю та підвищити загальну надійність системи.

Збір та аналіз зворотного зв'язку від користувачів став основою для ітеративного покращення системи, визначення пріоритетів розвитку та формування дорожньої карти майбутніх оновлень.

**Опис роботи програмного забезпечення**

За результатами тестування та з урахуванням зворотного зв'язку від користувачів був сформований кінцевий варіант програмного забезпечення для моніторингу погодних умов. Нижче представлено детальний огляд функціональності системи з візуальними прикладами роботи основних компонентів.

Візуальні матеріали демонструють реальну взаємодію користувача з системою та наочно ілюструють результати впровадження архітектурних рішень та дизайнерських підходів, що були описані в попередніх розділах. Кожен скріншот супроводжується описом відповідної функціональності та особливостей реалізації.

Представлені матеріали відображають фінальну версію інтерфейсу, яка була оптимізована на основі користувацького тестування для забезпечення максимальної зручності та ефективності взаємодії з системою моніторингу погодних умов.

На рис.4.1 представлено сторінку автентифікації системи, реалізовану з використанням Material UI. Інтерфейс містить форму для входу з полями для електронної пошти та пароля, опцію відновлення забутого пароля та альтернативний метод автентифікації через Google. Також, якщо користувач небажає авторизуватися, є можливість користування першим Weather Explore.

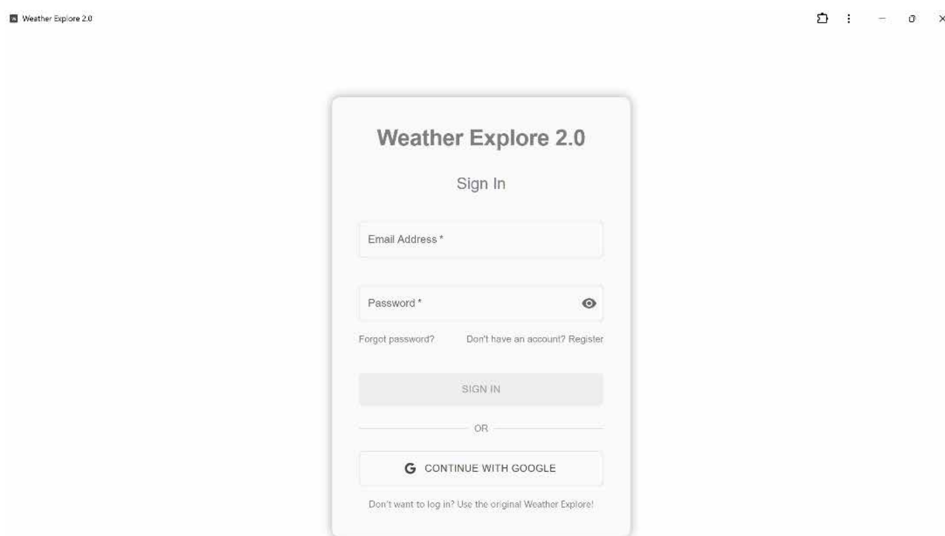


Рис. 4.1 Система автентифікації

Після автентифікації користувач потрапляє на головну сторінку додатку, зображено на рис.4.2, де відображено поле для додавання міста. Мінімалістичний дизайн з адаптивною структурою фокусує увагу користувача на основній функції — додаванні локацій для моніторингу погоди.

Верхня частина інтерфейсу містить панель керування фоновою музикою та елементи профілю користувача. Ліворуч розташований горизонтальний повзунок регулювання гучності, що дозволяє користувачу плавно змінювати рівень звуку без переходу до окремих налаштувань. Поруч з регулятором

розміщена кнопка керування відтворенням у вигляді двох горизонтальних ліній, яка виконує функцію паузи під час програвання треку.

По центру відображається інформаційний блок з назвою поточного треку що надає користувачу контекстну інформацію про музичний супровід без перевантаження інтерфейсу додатковими деталями.

Праворуч розміщені елементи профілю та комунікації. Аватарка користувача із зображенням, служить візуальним ідентифікатором та одночасно кнопкою доступу до налаштувань профілю. Поруч розташований значок форми зворотного зв'язку, представлений іконкою документа з олівцем, що забезпечує швидкий доступ до функції надсилання коментарів та пропозицій щодо роботи додатку.

Рис. 4.2 Початковий екран додавання міста

На рис.4.3, реалізовано форму зворотного зв'язку через Google Forms, що дозволяє користувачам оцінити додаток, надати коментарі та пропозиції щодо покращення. Це важлива складова системи для збору користувацьких відгуків та постійного вдосконалення функціональності

Рис. 4.3 Зворотний зв'язок через Google Forms

Рис.4.4 демонструє сторінку профілю користувача, де відображається аватар, ім'я та електронна пошта. Мінімалістичний інтерфейс з кнопкою редагування профілю забезпечує просте управління особистими даними.

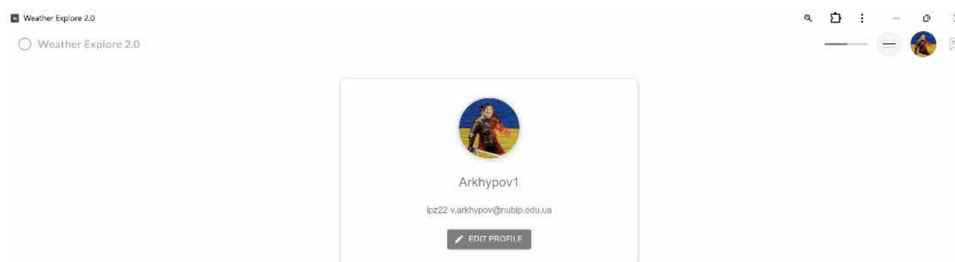


Рис. 4.4 Профіль користувача

На рис.4.5 представлено головну сторінку з відображенням погодних даних для одного міста та картками для додавання нових локацій. Погодна картка містить основну інформацію — поточну температуру, вологість та швидкість вітру разом з відповідною іконкою погодних умов.

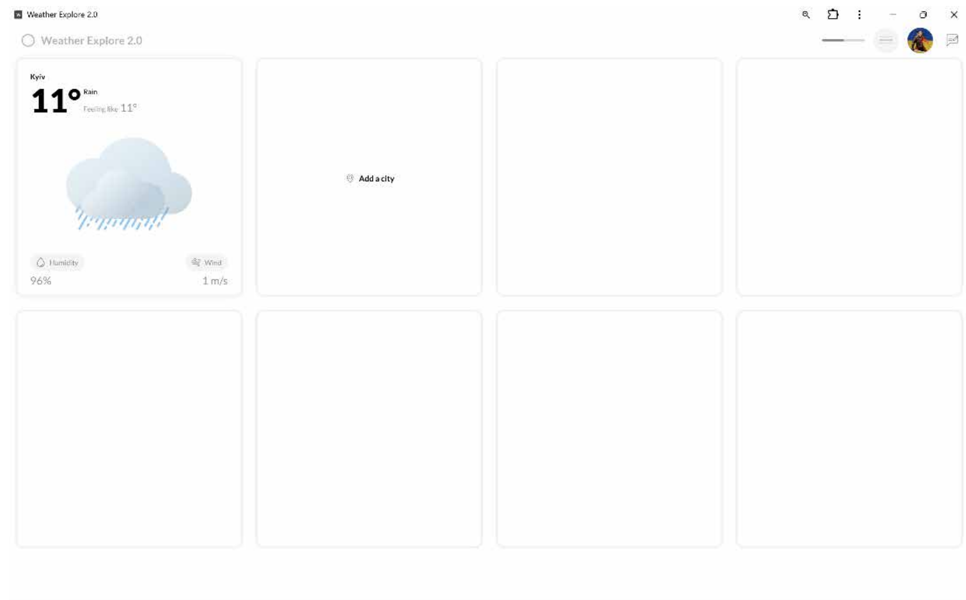


Рис. 4.5 Головна сторінка з відображенням погодних даних

Рис.4.6 демонструє інтерфейс з кількома містами та їх погодними даними. Крім основної інформації про погоду, система відображає прогноз на кілька днів для обраного міста. Також присутня функціональність для редагування, оновлення, видалення міст із списку моніторингу.

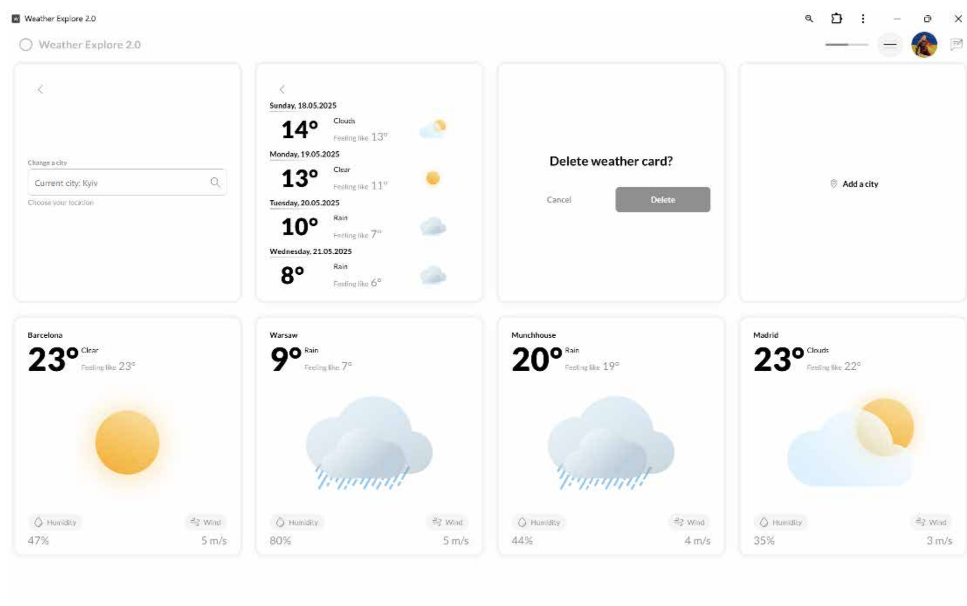


Рис. 4.6 Інтерфейс з кількома містами та їх погодними даними

На рис.4.7 показано меню вибору теми інтерфейсу з чотирма варіантами: White (світла), Black (темна), Gray (сіра) та Higgs (спеціальна). Ця функція дозволяє персоналізувати візуальне представлення додатку відповідно до вподобань користувача.

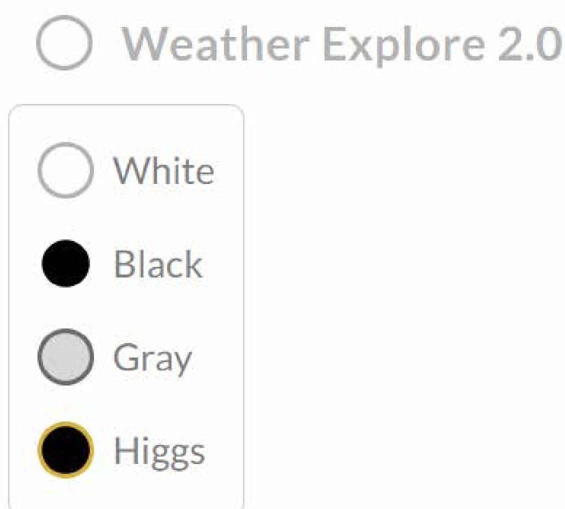


Рис. 4.7 Теми інтерфейсу

Скріншоти на рис. 4.8-4.10 демонструють роботу системи в різних темах. Всі елементи інтерфейсу, включаючи погодні картки, автоматично адаптуються до обраної теми зі збереженням контрасту та читабельності.

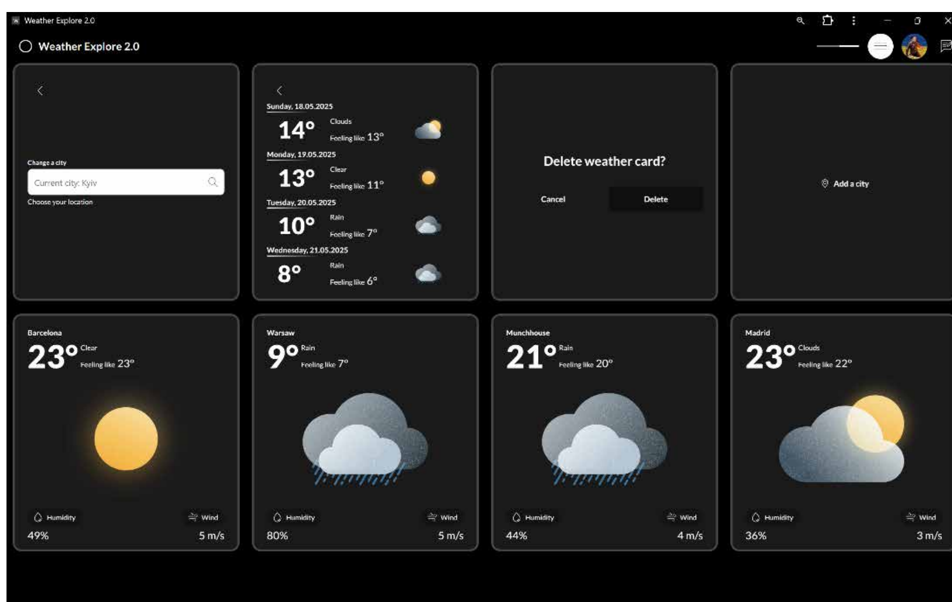


Рис. 4.8 Black тема

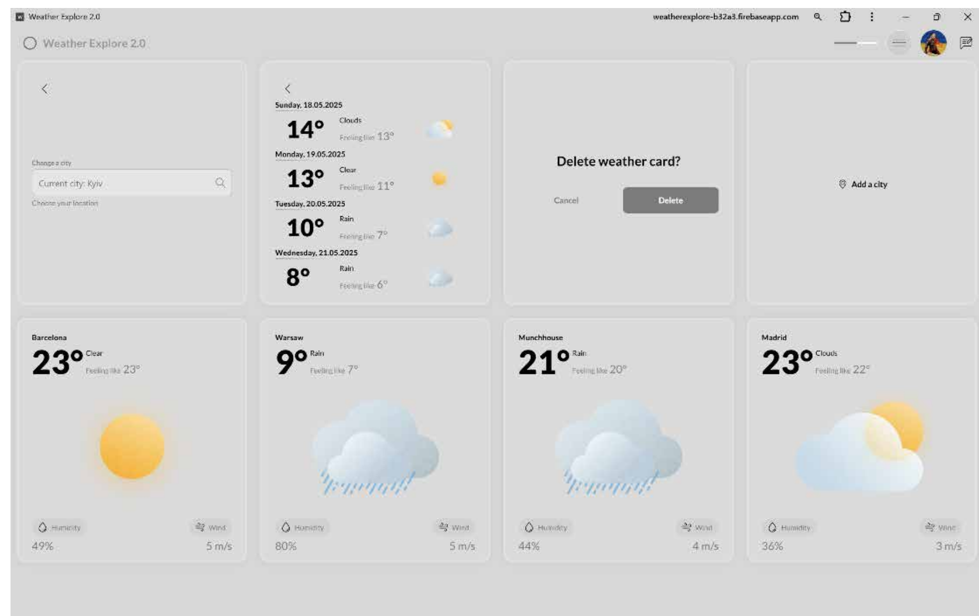


Рис. 4.9 Gray тема

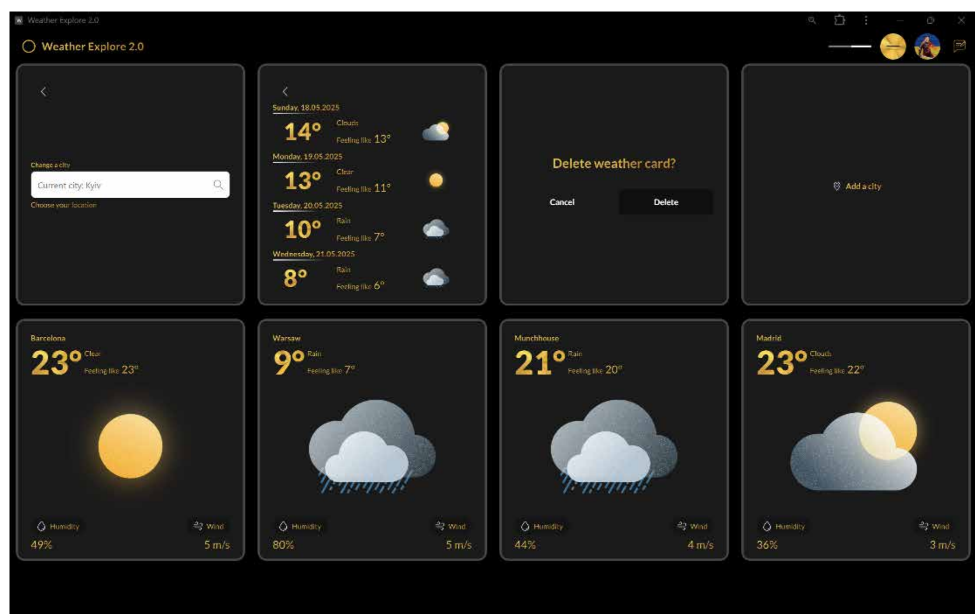


Рис. 4.10 Higgs тема

Система моніторингу погодних умов поєднує функціональність отримання метеорологічних даних з персоналізованим користувацьким досвідом, забезпечуючи зручний доступ до інформації про погоду для різних локацій.

## 4.2 Вимоги до апаратного та програмного забезпечення

Діаграма розгортання — діаграма в UML, на якій відображаються обчислювальні вузли під час роботи програми, компоненти, та об'єкти, що виконуються на цих вузлах. Компоненти відповідають представленню робочих екземплярів одиниць коду. [17]

Представлена на діаграмі розгортання зображено на рис.4.11, системи моніторингу погодних умов демонструє взаємодію між основними компонентами програмного забезпечення. Центральним елементом є веб-додаток ReactApp, реалізований з використанням HTML/CSS/TypeScript, який взаємодіє з API сервісом OpenWeatherMap для отримання метеорологічних даних та з Firebase Backend для автентифікації користувачів та зберігання персоналізованих налаштувань.

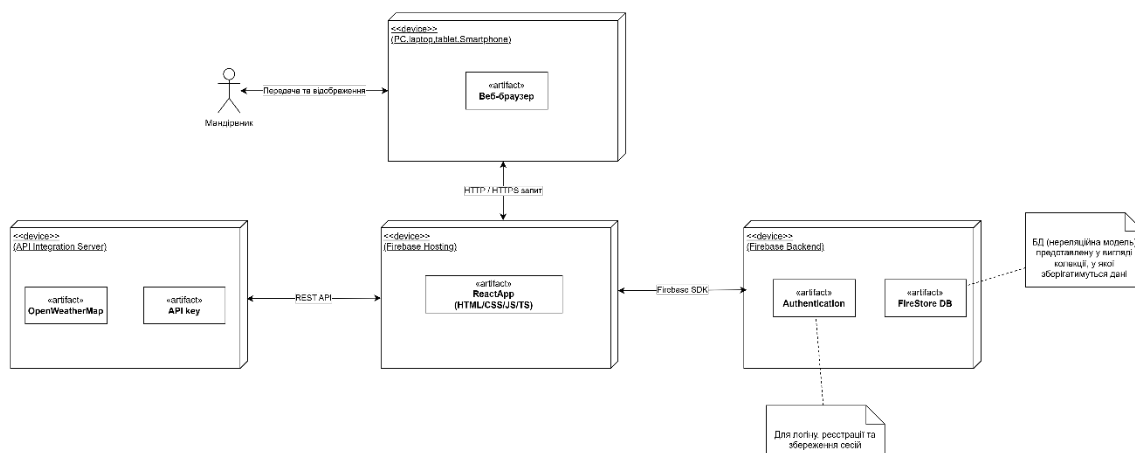


Рис. 4.11 Діаграма розгортання

Представлена на діаграмі архітектура розгортання системи моніторингу погодних умов демонструє взаємодію між основними компонентами програмного забезпечення. Центральним елементом є веб-додаток ReactApp, реалізований з використанням HTML/CSS/TypeScript, який взаємодіє з API сервісом OpenWeatherMap для отримання метеорологічних даних та з Firebase Backend для автентифікації користувачів та зберігання персоналізованих налаштувань.

**Клієнтська частина**

**Апаратне забезпечення (мінімальні вимоги)**

**Для настільних комп'ютерів та ноутбуків:**

- Процесор: двоядерний з частотою від 1.8 ГГц;
- Оперативна пам'ять: від 4 ГБ;
- Графічний адаптер з підтримкою WebGL;
- Вільний простір на диску: від 200 МБ;
- Роздільна здатність екрану: від 1280x720 пікселів;
- Підключення до мережі Інтернет зі швидкістю від 30 Мбіт/с.

**Для планшетів:**

- Процесор: двоядерний з частотою від 1.5 ГГц;
- Оперативна пам'ять: від 4 ГБ;
- Вільний простір для кешування: від 100 МБ;
- Роздільна здатність екрану: від 1024x768 пікселів;
- Підключення до мережі Інтернет (Wi-Fi або мобільний інтернет).

**Для смартфонів:**

- Процесор: від 1.2 ГГц;
- Оперативна пам'ять: від 4 ГБ;
- Вільний простір для кешування: від 50 МБ;
- Роздільна здатність екрану: від 320dp (density-independent pixels);
- Підключення до мережі Інтернет (Wi-Fi або мобільний інтернет).

**Програмне забезпечення**

**Для настільних операційних систем:**

- Windows 10/11;
- macOS 10.13+;
- Linux з графічним інтерфейсом (Ubuntu 18.04+, Fedora 30+, тощо).

**Для мобільних операційних систем:**

- Android 10.0+;
- iOS 12+;

**Підтримувані веб-браузери:**

- Google Chrome (версія 90+)
- Mozilla Firefox (версія 88+)
- Safari (версія 14+)
- Microsoft Edge (версія 90+)
- Chrome для Android (версія 90+)
- Safari для iOS (версія 14+)

**Додаткові вимоги:**

- Підтримка JavaScript: увімкнена
- Підтримка файлів cookie: увімкнена

Додаток розроблено з використанням принципів адаптивного дизайну та прогресивного веб-додатку (PWA), що забезпечує оптимальний користувацький досвід на всіх типах пристроїв від смартфонів до настільних комп'ютерів. Інтерфейс автоматично адаптується до розміру екрану, орієнтації пристрою та типу взаємодії (сенсорний ввід або миша/клавіатура).

**Серверна частина**

Система не потребує власного серверного апаратного забезпечення для роботи, оскільки використовує зовнішні сервіси:

**Firebase Backend**

- Firebase Authentication для управління користувачами та автентифікації;
- Firebase Firestore для зберігання даних користувачів та їх налаштувань;
- Firebase Hosting для розміщення веб-додатку.

**Зовнішні API**

OpenWeatherMap API для отримання метеорологічних даних з використанням API-ключа.

**Особливості розгортання**

Діаграма розгортання відображає багаторівневу архітектуру системи, де користувач (мандрівник) взаємодіє з веб-браузером, який завантажує веб-додаток з Firebase Hosting. Додаток, в свою чергу, взаємодіє з Firebase Authentication для автентифікації користувачів, Firebase Firestore для зберігання та отримання даних, та з API OpenWeatherMap для отримання актуальної метеорологічної інформації.

Використання хмарних сервісів Firebase забезпечує високу доступність, автоматичне масштабування та безпеку даних без необхідності розгортання та підтримки власної серверної інфраструктури. Це дозволяє зосередитись на розробці функціональності додатку та покращенні користувацького досвіду.

Для розгортання нових версій додатку використовується CI/CD конвеєр, що забезпечує автоматичну збірку, оновлення веб-додатку на Firebase Hosting після внесення змін до репозиторію коду.

## 4.3 Склад інсталяційного пакету

Інсталяційний пакет системи моніторингу погодних умов складається з оптимізованих файлів збірки веб-додатку, які розгортаються на платформі Firebase Hosting. Після процесу збірки з використанням Vite, проект компілюється у набір статичних файлів, готових до розгортання.

**Основні компоненти збірки**

**HTML-файли**:

- index.html — головний HTML-файл, що служить точкою входу для додатку.

**JavaScript-файли**:

- Основний бандл JS з мінімізованим кодом React-додатку;
- Розділені (chunked) JS-файли для оптимізації завантаження;
- Відкладені (lazy-loaded) компоненти для покращення продуктивності.

**CSS-файли**:

- Мінімізовані та оптимізовані стилі з SCSS-компіляції;
- Стилі компонентів Material UI та Bootstrap.

**Статичні ресурси**:

- Оптимізовані зображення для різних погодних умов;
- SVG-іконки з підтримкою адаптації до теми інтерфейсу;
- Шрифти з оптимізацією для веб.

**Конфігураційні файли**:

- Файл конфігурації Firebase (firebase.json);
- Файл налаштувань хостингу (firebase-hosting.json);
- Конфігурація PWA (Progressive Web App) з маніфестом.

**Сервіс-воркер**:

- JavaScript-файл для підтримки офлайн-функціональності;
- Конфігурація кешування для оптимізації продуктивності.

**Процес інсталяції та розгортання**

Оскільки додаток реалізовано як веб-застосунок з хостингом на Firebase, процес інсталяції відбувається автоматично при першому відвідуванні сайту користувачем. Для розробників, які бажають розгорнути додаток локально або на власному хостингу, інсталяційний пакет може бути отриманий наступним чином:

1. Клонування репозиторію проекту з системи контролю версій;
2. Встановлення залежностей за допомогою npm або yarn;
3. Налаштування змінних середовища для підключення до Firebase;
4. Запуск процесу збірки для генерації оптимізованих файлів;
5. Розгортання на цільовій платформі (Firebase Hosting або інший хостинг).

**Реалізація Progressive Web App**

Важливою частиною інсталяційного пакету є підтримка технології Progressive Web App (PWA), що дозволяє користувачам встановлювати додаток безпосередньо з браузера на свої пристрої. Компоненти PWA включають:

- Web App Manifest з описом додатку та налаштуваннями відображення;
- Service Worker для кешування та офлайн-функціональності;
- Іконки різних розмірів для різних платформ та пристроїв.

**Конфігурація Firebase**

Для повноцінного функціонування додатку після розгортання необхідно налаштувати відповідні сервіси Firebase:

- Firebase Authentication з увімкненими методами автентифікації (Email/Password, Google);
- Firebase Firestore з відповідною структурою колекцій та правилами безпеки;
- Firebase Hosting з налаштуваннями кешування та перенаправленнями для SPA (Single Page Application).

Інсталяційний пакет розроблено з урахуванням оптимізації завантаження та максимальної продуктивності на різних пристроях, що забезпечує швидкий старт додатку та економне використання мобільного трафіку.

# ВИСНОВКИ

У ході виконання кваліфікаційної роботи було розроблено програмне забезпечення системи моніторингу погодних умов, що надає користувачам зручний інструмент для отримання актуальної метеорологічної інформації. Проект був реалізований з використанням сучасних технологій веб-розробки та хмарних сервісів, що забезпечило створення надійної, масштабованої та доступної системи.

На етапі аналізу предметної області було досліджено специфіку метеорологічних даних та потреби користувачів, зокрема туристів та мандрівників, у актуальній інформації про погодні умови. Сформульовані вимоги до програмної системи визначили ключові функціональні та нефункціональні характеристики, що стали основою для подальшої розробки.

Моделювання предметної області дозволило сформувати чітке розуміння структури даних та взаємозв'язків між основними сутностями системи. За допомогою діаграм різних типів (ER-діаграми, діаграми класів, пакетів та компонентів) було спроектовано архітектуру, що забезпечує модульність, масштабованість та зручність підтримки програмного забезпечення.

Для зберігання даних обрано Firebase Firestore як документо-орієнтовану NoSQL базу даних, що оптимально відповідає вимогам до гнучкості структури даних, підтримки реального часу та можливості офлайн-роботи. Розроблена система управління інформаційною базою забезпечує надійний доступ до даних з належними рівнями безпеки.

У процесі розробки програмних модулів було використано React з TypeScript в поєднанні з SCSS для стилізації компонентів, що дозволило створити сучасний, інтуїтивно зрозумілий та адаптивний інтерфейс користувача. Впроваджено контекстний підхід до управління станом додатку, що забезпечило чітке розділення відповідальності та оптимізацію передачі даних між компонентами.

Особливу увагу було приділено персоналізації користувацького досвіду

через реалізацію системи тем оформлення, музичного супроводу та збереження індивідуальних налаштувань. Впроваджено механізми ізольованої обробки помилок, що забезпечують стабільність роботи системи навіть при виникненні збоїв в окремих компонентах.

Тестування системи з залученням реальних користувачів дозволило виявити та усунути потенційні проблеми, оптимізувати інтерфейс та підтвердити відповідність розробленого програмного забезпечення потребам цільової аудиторії.

Результатом роботи стала функціональна система моніторингу погодних умов, яка забезпечує:

- Надійну автентифікацію та персоналізацію користувацького досвіду;
- Отримання актуальних метеорологічних даних з відомого сервісу OpenWeather;
- Зручне відображення погоди у різних містах з можливістю управління списком локацій;
- Адаптивний інтерфейс, що коректно відображається на різних пристроях;
- Можливість встановлення як Progressive Web App для швидкого доступу.

Розроблене програмне забезпечення може бути розгорнуте на Firebase Hosting без необхідності додаткової серверної інфраструктури, що спрощує процес впровадження та експлуатації системи.

У перспективі розвитку проекту можливе розширення функціональності системи через додавання детальних погодних прогнозів, інтеграцію з додатковими джерелами метеорологічних даних, впровадження аналітичних інструментів та персоналізованих рекомендацій на основі погодних умов.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. AccuWeather. URL: https://uk.wikipedia.org/wiki/AccuWeather (дата звернення: 03.02.2025).
2. Bootstrap. ITMaster. URL: https://itmaster.biz.ua/programming/web-prohramuvannia/bootstrap.html (дата звернення: 10.02.2025).
3. Firebase. Avada Media. URL: https://avada-media.ua/blog/firebase/#scho-take-firebase (дата звернення: 15.02.2025).
4. Material UI. URL: https://mui.com/material-ui/getting-started/ (дата звернення: 18.02.2025).
5. React. URL: https://uk.wikipedia.org/wiki/React (дата звернення: 23.02.2025).
6. React Router. FPM DNU. URL: http://fpm.dnu.dp.ua/2019/12/04/react-router/ (дата звернення: 28.02.2025).
7. SCSS. Highload.Tech. URL: https://highload.tech/uk/scss/ (дата звернення: 24.03.2025).
8. The Weather Channel. URL: https://www.linkedin.com/company/the-weather-channel/ (дата звернення: 28.03.2025).
9. Windy. URL: https://uk.wikipedia.org/wiki/Windy (дата звернення: 30.03.2025).
10. Архипов В. А. Програмне забезпечення системи моніторингу погодних умов. Збірник наукових праць за матеріалами VII Всеукраїнської науково-практичної конференції студентів і аспірантів «Теоретичні та прикладні аспекти розробки комп’ютерних систем 2025», м. Київ, 24 квітня. 2025 р. НУБІП України. Київ, 2025..
11. Діаграма класів. URL: https://surl.lu/bkjbqi (дата звернення: 05.04.2025).
12. Діаграма компонентів. URL: https://surli.cc/ponsoi (дата звернення: 10.04.2025).

13. Діаграма кооперацій. URL: https://studfile.net/preview/9828823/ (дата звернення: 11.04.2025).

14. Діаграма пакетів. URL: https://surli.cc/gbpcce (дата звернення: 13.04.2025).

15. Діаграма послідовності. URL: https://surl.lu/qwfmma (дата звернення: 17.04.2025).

16. Діаграма прецедентів. URL: https://surl.li/qdrfay (дата звернення: 23.04.2025).

17. Діаграма розгортання. URL: https://surl.li/jgovwl (дата звернення: 29.04.2025).

18. Діаграма діяльності. URL: https://surl.li/nljzhv (дата звернення: 13.05.2025).

19. Модель «сутність — зв'язок». URL: https://surl.li/pnaave (дата звернення: 17.05.2025).

20. Модель предметної області. URL: https://surli.cc/vyzkzn (дата звернення: 20.05.2025).

21. Що таке СУБД і для чого вони потрібні. Foxminded. URL: https://foxminded.ua/systema-upravlinnia-bazamy-danykh/ (дата звернення: 26.05.2025).

# Додаток А

### Авторизація

```
import { createContext, useContext, useEffect, useState } from "react";
import {
  getAuth,
  onAuthStateChanged,
  signInWithPopup,
  signOut,
  GoogleAuthProvider,
  createUserWithEmailAndPassword,
  signInWithEmailAndPassword,
  sendPasswordResetEmail,
  User,
} from "firebase/auth";
import { initializeApp } from "firebase/app";
import { getFirestore } from "firebase/firestore";

const firebaseConfig = {
  apiKey: import.meta.env.VITE_FIREBASE_API_KEY,
  authDomain: import.meta.env.VITE_FIREBASE_AUTH_DOMAIN,
  projectId: import.meta.env.VITE_FIREBASE_PROJECT_ID,
  storageBucket: import.meta.env.VITE_FIREBASE_STORAGE_BUCKET,
  messagingSenderId: import.meta.env.VITE_FIREBASE_MESSAGING_SENDER_ID,
  appId: import.meta.env.VITE_FIREBASE_APP_ID,
};

const app = initializeApp(firebaseConfig);
export const auth = getAuth(app);
export const db = getFirestore(app);
interface AuthContextType {
  user: User | null;
  loading: boolean;
  signInWithGoogle: () => Promise<void>;
  signInWithEmail: (email: string, password: string) => Promise<void>;
  signUpWithEmail: (email: string, password: string) => Promise<void>;
  logout: () => Promise<void>;
  resetPassword: (email: string) => Promise<void>;
  error: string | null;
  clearError: () => void;
}

const AuthContext = createContext<AuthContextType | null>(null);

export function AuthProvider({ children }: { children: React.ReactNode }) {
  const [user, setUser] = useState<User | null>(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState<string | null>(null);
```

```
useEffect(() => {
  const unsubscribe = onAuthStateChanged(auth, (user) => {
    setUser(user);
    setLoading(false);
  });

  return unsubscribe;
}, []);

const signInWithGoogle = async () => {
  try {
    const provider = new GoogleAuthProvider();
    await signInWithPopup(auth, provider);
    setError(null);
  } catch (error) {
    setError((error as Error).message);
  }
};

const signInWithEmail = async (email: string, password: string) => {
  try {
    await signInWithEmailAndPassword(auth, email, password);
    setError(null);
  } catch (error) {
    setError((error as Error).message);
  }
};

const signUpWithEmail = async (email: string, password: string) => {
  try {
    await createUserWithEmailAndPassword(auth, email, password);
    setError(null);
  } catch (error) {
    setError((error as Error).message);
  }
};

const logout = async () => {
  try {
    await signOut(auth);
    setError(null);
  } catch (error) {
    setError((error as Error).message);
  }
};

const resetPassword = async (email: string) => {
  try {
    await sendPasswordResetEmail(auth, email);
  } catch (error) {
    const errorMessage = (error as Error).message;
    setError(errorMessage);
```

```
    throw error;
  }
};

const clearError = () => setError(null);

return (
  <AuthContext.Provider
    value={{
      user,
      loading,
      signInWithGoogle,
      signInWithEmail,
      signUpWithEmail,
      logout,
      resetPassword,
      error,
      clearError,
    }}
  >
    {children}
  </AuthContext.Provider>
);
}

export const useAuth = () => {
  const context = useContext(AuthContext);
  if (!context)
```

# Додаток Б

## CRUD логіка для керування погоди

```
import React, { createContext, useState, useContext, useEffect, ReactNode } from "react";
import { useAuth } from "./AuthContext";
import { useRefetch } from "./RefetchContext";
import {
  collection,
  addDoc,
  query,
  where,
  onSnapshot,
  updateDoc,
  getDocs,
  doc,
  deleteDoc,
} from "firebase/firestore";
import { db } from "./AuthContext";
import configuration from "../configuration";
import { WeatherData } from "../types/weather.types";


export interface City {
  id: string;
  createdAt: string;
  weatherData?: WeatherData;
  index: number;
}

interface WeatherContextType {
  cities: City[];
  loading: boolean;
  error: string;
  isInitialView: boolean;
  isInitialLoading: boolean;
  addCity: (cityName: string, index: number) => Promise<string | void>;
  deleteCity: (cityId: string) => Promise<void>;
  updateCity: (cityId: string, newCityName: string) => Promise<string | void>;
  setError: React.Dispatch<React.SetStateAction<string>>;
}


const WeatherContext = createContext<WeatherContextType | undefined>(undefined);


export function WeatherProvider({ children }: { children: ReactNode }) {
  const { user } = useAuth();
  const { isToggled } = useRefetch();
  const [loading, setLoading] = useState<boolean>(false);
  const [error, setError] = useState<string>("");
```

```
const [cities, setCities] = useState<City[]>([]);
const [isInitialView, setIsInitialView] = useState<boolean>(true);
const [isInitialLoading, setIsInitialLoading] = useState<boolean>(true);
async function ApiWeather(name: string, api: string) {
const request = await fetch(`https://api.openweathermap.org/data/2.5/forecast?q=${name.trim().toLowerCase()}&appid=${api}&units=metric`)
  return request;
}

useEffect(() => {
  if (!user) return;

  const fetchCities = async () => {
    try {
      setIsInitialLoading(true);
      const citiesRef = collection(db, "cities");
      const q = query(citiesRef, where("uid", "==", user.uid));


      const updateWeatherData = async () => {
        const snapshot = await getDocs(q);
        for (const doc of snapshot.docs) {
          const data = doc.data();
          try {
            const weatherResponse = await ApiWeather(data.name, configuration.apiToken);
            if (weatherResponse.ok) {
              const weatherData: WeatherData = await weatherResponse.json();
              await updateDoc(doc.ref, { weatherData });
            }
          } catch (error) {
            console.error(`Error updating weather for ${data.name}:`, error);
          }
        }
      };


      await updateWeatherData();


      const unsubscribe = onSnapshot(q, (snapshot) => {
        const citiesList: City[] = [];
        snapshot.forEach((doc) => {
          const data = doc.data();
          citiesList.push({
            id: doc.id,
            createdAt: data.createdAt,
            weatherData: data.weatherData,
            index: data.index,
          });
        });
        setCities(citiesList);
```

```
      const isNewInitialView = citiesList.length === 0;
      setIsInitialView(isNewInitialView);
      if (isNewInitialView) {
        setError("");
      }
      setIsInitialLoading(false);
    });

    return unsubscribe;
  } catch (error) {
    console.error("Error in fetchCities:", error);
    setError("Failed to load your items");
    setIsInitialLoading(false);
  }
};

fetchCities();
}, [user, isToggled]);


const addCity = async (cityName: string, index: number): Promise<string | void> => {
  if (!user) {
    return "You must be logged in to add items";
  }

  try {
    setLoading(true);

    const weatherResponse = await ApiWeather(cityName, configuration.apiToken);
    const weatherData: WeatherData = await weatherResponse.json();

    if (weatherResponse.status !== 200) {
      return "Failed to fetch weather data";
    }

    const apiCityName = weatherData.city.name;
    const existingCity = cities.find(
      (c) =>
        c.weatherData?.city?.name.toLowerCase() === apiCityName.toLowerCase()
    );

    if (existingCity) {
      return `City ${apiCityName} already added!`;
    }


    const newCity = {
      uid: user.uid,
      name: apiCityName,
      createdAt: new Date().toISOString(),
      weatherData: weatherData,
      index: index,
```

```
    };

    await addDoc(collection(db, "cities"), newCity);
    setIsInitialView(false);


    return;
  } catch (err) {
    return err instanceof Error ? err.message : "Failed to add the city";
  } finally {
    setLoading(false);
  }
};


const deleteCity = async (cityId: string) => {
  if (!user) return;
  try {

    setCities(prevCities => prevCities.filter(city => city.id !== cityId));


    const cityRef = doc(db, "cities", cityId);
    await deleteDoc(cityRef);

  } catch (error) {
    console.error("Error deleting city:", error);
    setError("Failed to delete the city");


    const citiesRef = collection(db, "cities");
    const q = query(citiesRef, where("uid", "==", user.uid));
    const snapshot = await getDocs(q);
    const citiesList: City[] = [];
    snapshot.forEach((doc) => {
      const data = doc.data();
      citiesList.push({
        id: doc.id,
        createdAt: data.createdAt,
        weatherData: data.weatherData,
        index: data.index,
      });
    });
    setCities(citiesList);
  }
};


const updateCity = async (cityId: string, newCityName: string): Promise<string | void> =>
{
  if (!user) {
    return "You must be logged in to update items";
```

```
  }

  try {
    setLoading(true);


    const weatherResponse = await ApiWeather(newCityName, configuration.apiToken);

    if (weatherResponse.status !== 200) {
      return "Failed to fetch weather data";
    }

    const weatherData: WeatherData = await weatherResponse.json();
    const apiCityName = weatherData.city.name;


    const existingCity = cities.find(
      (c) =>
        c.id !== cityId &&
        c.weatherData?.city?.name.toLowerCase() === apiCityName.toLowerCase()
    );

    if (existingCity) {
      return `City ${apiCityName} already added!`;
    }


    const cityRef = doc(db, "cities", cityId);
    await updateDoc(cityRef, {
      name: apiCityName,
      weatherData: weatherData
    });


    return;
  } catch (err) {
    return err instanceof Error ? err.message : "Failed to update the city";
  } finally {
    setLoading(false);
  }
};


const value = {
  cities,
  loading,
  error,
  isInitialView,
  isInitialLoading,
  addCity,
  deleteCity,
  updateCity,
```

```
    setError,
  };

  return (
    <WeatherContext.Provider value={value}>
      {children}
    </WeatherContext.Provider>
  );
}


export function useWeather() {
  const context = useContext(WeatherContext);
  if (context === undefined) {
    throw new Error("useWeather must be used within a WeatherProvider");
  }
```

# Додаток В

## Зміна теми інтерфейсу

```
import { createContext, ReactNode, useContext, useEffect, useState } from "react";
import { colorIcons, ColorIcon } from "../layouts/Header/ThemeSelection/color";

interface ThemeContextType {
  theme: ColorIcon | undefined;
  themeHandler: (theme: string) => void;
}


const THEME_STORAGE_KEY = 'weather-explore-theme';

export const ThemeContext = createContext<ThemeContextType | undefined>(
  undefined
);

export function ThemeProvider({ children }: { children: ReactNode }) {
  const getInitialTheme = (): ColorIcon | undefined => {
    const savedTheme = localStorage.getItem(THEME_STORAGE_KEY);
    if (savedTheme) {
      const themeIcon = colorIcons.find((icon) => icon.description === savedTheme);
      return themeIcon || colorIcons[0];
    }
    return colorIcons[0];
  };

  const [theme, setTheme] = useState<ColorIcon | undefined>(getInitialTheme);

  function themeHandler(theme: string) {
    const themeIcon = colorIcons.find((icon) => icon.description === theme);
    if (themeIcon) {
      localStorage.setItem(THEME_STORAGE_KEY, theme);
      setTheme(themeIcon);
    }
  }

  return (
    <>
      <ThemeContext.Provider value={{ theme, themeHandler }}>
        {children}
      </ThemeContext.Provider>
    </>
  );
}

export const useTheme = () => {
```

```
  const context = useContext(ThemeContext);
  if (!context) {
    throw new Error("useTheme must be used within a ThemeProvider");
  }
  return context;
};

export const ThemeHandler = () => {
  const { theme } = useTheme();

  useEffect(() => {
    document.body.classList.remove('theme-white', 'theme-black', 'theme-gray', 'theme-higgs');

    if (theme) {
      document.body.classList.add(`theme-${theme.description.toLowerCase()}`);

      if (theme.background)
```

# Додаток Г

## Підказки для користувача

```
import React, { createContext, useContext, useEffect, useState, useLayoutEffect, useRef }
from 'react';
import Joyride, { CallBackProps, STATUS } from 'react-joyride';
import { useAuth } from './AuthContext';

interface JoyrideContextProps {
  isTouring: boolean;
  startTour: () => void;
  endTour: () => void;
}

const JoyrideContext = createContext<JoyrideContextProps>({
  isTouring: false,
  startTour: () => { },
  endTour: () => { },
});

export const useJoyride = () => useContext(JoyrideContext);

interface JoyrideProviderProps {
  children: React.ReactNode;
}

export const JoyrideProvider: React.FC<JoyrideProviderProps> = ({ children }) => {
  const [isTouring, setIsTouring] = useState(false);
  const [tourCompleted, setTourCompleted] = useState(false);
  const [isMobile, setIsMobile] = useState(false);
  const { user } = useAuth();
  const tourSteps = useRef<any[]>([]);

  // Detect screen size for responsive tour
  useLayoutEffect(() => {
    const checkScreenSize = () => {
      const mobile = window.innerWidth < 992; // Bootstrap lg breakpoint
      setIsMobile(mobile);
      // Update steps when screen size changes
      setupSteps(mobile);
    };

    checkScreenSize();
    window.addEventListener('resize', checkScreenSize);

    return () => window.removeEventListener('resize', checkScreenSize);
  }, []);

  // Function to set up the tour steps based on screen size
```

```
const setupSteps = (mobile: boolean) => {
  // Common steps for all screen sizes
  const welcomeStep = {
    target: 'body',
    content: 'Welcome to Weather Explore 2.0! We will show you the main features and capabilities of our weather service.',
    title: 'Start Your Journey',
    placement: 'center' as const,
    disableBeacon: true,
  };

  const logoStep = {
    target: '[data-tour="logo-refresh"]',
    content: 'By clicking on the Weather Explore 2.0 logo, you can quickly update the weather data.',
    title: 'Data Refresh',
  };

  const finalStep = {
    target: '[data-tour="first-card"]',
    content: 'Now you can add your first city to track the weather. Enjoy using the application!',
    title: 'Adding a City',
  };

  // Steps that depend on screen size
  if (mobile) {
    // For mobile version with requested order
    tourSteps.current = [
      welcomeStep,
      {
        target: '[data-tour="navbar-theme"]',
        content: 'Here you can change the interface theme - light, dark, gray, and higgs.',
        title: 'Theme Selection',
        disableBeacon: true,
      },
      logoStep,
      {
        target: '.navbar-toggler',
        content: 'Click here to open the menu and access all features.',
        title: 'Mobile Menu',
        spotlightClicks: true,
        disableBeacon: true,
      },
      {
        target: '#navbarSupportedContent',
        content: 'Here you can find theme settings, music controls, user profile, and feedback options.',
        title: 'Menu Features',
        disableBeacon: true,
      },
      finalStep,
    ];
```

```
  } else {
    // For desktop: show all items in the header
    tourSteps.current = [
      welcomeStep,
      {
        target: '[data-tour="navbar-theme"]',
        content: 'Here you can change the interface theme - light, dark, gray, and higgs.',
        title: 'Theme Selection',
      },
      logoStep,
      {
        target: '[data-tour="music-system"]',
        content: 'Turn on music for a more pleasant experience while using the app.',
        title: 'Background Music',
      },
      {
        target: '.auth-section',
        content: 'Your user menu. Here you can navigate to your profile or log out of the system.',
        title: 'User Menu',
      },
      {
        target: '[data-tour="feedback-icon"]',
        content: 'Click here to send us feedback about the application.',
        title: 'Send Feedback',
      },
      finalStep,
    ];
  }
};

// Initialize steps
useEffect(() => {
  setupSteps(isMobile);
}, [isMobile]);

// Check local storage on initial load to see if user has completed the tour
useEffect(() => {
  const hasCompletedTour = localStorage.getItem('tourCompleted');
  if (hasCompletedTour === 'true') {
    setTourCompleted(true);
  }
}, []);

// Start tour automatically for new users after authentication
useEffect(() => {
  if (user && !tourCompleted) {
    // Small delay to ensure all components are mounted
    const timer = setTimeout(() => {
      startTour();
    }, 1000);
    return () => clearTimeout(timer);
  }
```

```
}, [user, tourCompleted]);
const startTour = () => {
  setIsTouring(true);
};
const endTour = () => {
  setIsTouring(false);
  setTourCompleted(true);
  localStorage.setItem('tourCompleted', 'true');
};
const handleJoyrideCallback = (data: CallBackProps) => {
  const { status, action, index } = data;
  // Handle mobile menu interaction
  if (isMobile && index === 2 && action === 'next') {
    // After showing burger menu step, we need to open the menu
    const burgerButton = document.querySelector('.navbar-toggler') as HTMLElement;
 if (burgerButton &&
!document.querySelector('#navbarSupportedContent')?.classList.contains('show')) {
      burgerButton.click();
    }
  }
  // Check for completion status
  if (status === STATUS.FINISHED || status === STATUS.SKIPPED) {
    endTour();
  }
};

return (
  <JoyrideContext.Provider value={{ isTouring, startTour, endTour }}>
    <Joyride
      callback={handleJoyrideCallback}
      continuous
      hideCloseButton
      disableCloseOnEsc={true}
      disableOverlayClose={true}
      disableScrolling={false}
      run={isTouring}
      scrollToFirstStep
      showProgress
      showSkipButton
      steps={tourSteps.current}
      styles={{
        options: {
          zIndex: 10000,
          primaryColor: '#5F5F5F',
          textColor: '#000000',
        },
        tooltip: {
          fontFamily: 'Lato, sans-serif',
          fontSize: '1rem',
```

# Додаток Д

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

**ДИПЛОМ**

**III ступеню**

**НАГОРОДЖУЄТЬСЯ**

***Архипов Вадим Андрійович***

***студент групи ІПЗ-22010бск ОС «Бакалавр»***

**за доповідь «Програмне забезпечення системи моніторингу погодних умов» на VII Всеукраїнській науково-практичній конференції студентів і аспірантів «Теоретичні та прикладні аспекти розробки комп'ютерних систем '2025»**

**Завідувач кафедри комп'ютерних наук НУБіП України** **Б.Л. Голуб**

**24 квітня 2025 р.**