НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри

Інформаційних систем і технологій
(назва кафедри)

Швиденко М.З, к.е.н., проф.
(підпис) (ПІБ)

“16” травня 2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему

Автоматизована система моніторингу погоди

Спеціальність 126 “Інформаційні системи та технології”

Гарант освітньої програми

к.е.н., доцент _____ Мокрієв Максим Володимирович
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник кваліфікаційної роботи

доктор тех наук, проф. _____ Смолій Вікторія Миколаївна
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ Каплун Богдан Васильович
(підпис) (ПІБ)

Київ 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Завідувач кафедри

інформаційних систем і технологій

Швиденко М.З, К.Е.Н , ПРОФ

Підпис ініціали та прізвище_____202_ р.

ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи

студенту(ці) Каплуну Богдану Васильовичу

Спеціальності 126 “Інформаційні системи та технології”

Тема роботи: **Автоматизована система моніторингу погоди**

Затверджена наказом ректора від 16.12.2024 р. № 2245-С

1. Термін подання завершеної роботи на кафедру – *10.06.2025*
2. Вихідні данні Методичні вказівки кафедри, діаграми, аналітика з SQL Server, макети інтерфейсу в Figma
3. Перелік питань, що розглядаються:

Аналіз предметної області та визначення актуальності теми?;
Проєктування архітектури інформаційної системи моніторингу погоди?;
Розробка бази даних у SQL Server?;
Формування SQL-запитів для аналізу погодних даних?;
Розробка прототипу інтерфейсу користувача в середовищі Figma?.

Календарний план:

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної області та формулювання цілей і завдань	16.12.2024 – 15.01.2025	Виконано
2	Розробка архітектури системи, створення UML-діаграм	16.01.2025 – 10.02.2025	Виконано
3	Проєктування та реалізація бази даних у SQL Server	11.02.2025 – 01.03.2025	Виконано
4	Розробка інтерфейсу користувача в Figma	02.03.2025 – 20.03.2025	Виконано
5	Тестування, оформлення документації та написання пояснювальної записки	21.03.2025 – 10.04.2025	Виконано

Керівник кваліфікаційної роботи _____/ доктор тех наук, проф. Смолій В.М_____
підпис ПІБ, вчене звання та ступінь

Завдання прийняв до виконання _____/ Каплун Богдан Васильович _____
підпис ПІБ

Дата отримання завдання 16.12.2024

РЕФЕРАТ

Тема бакалаврської кваліфікаційної роботи: “ Автоматизована система моніторингу погоди ”.

Автор роботи: Каплун Богдан Васильович .

Керівник роботи: Смолій Вікторія Миколаївна.

Метою роботи є розробити автоматизовану систему моніторингу погоди , яка забезпечує збір, збереження, обробку та аналіз метеорологічних даних з подальшим представленням інформації через зручний інтерфейс користувача

Під час виконання кваліфікаційної роботи були використані сучасні підходи до розробки програмного забезпечення, включаючи створення структурованої бази даних, побудову діаграм відповідно до стандартів UML та BPMN, а також макетування інтерфейсу користувача в інструменті Figma. Система дозволяє автоматично збирати, аналізувати і візуалізувати дані про погодні умови. Це особливо важливо для застосування в аграрному секторі, екологічному моніторингу або в освітніх цілях. Реалізація роботи продемонструвала мою здатність до аналітичного мислення, самоорганізації, а також уміння працювати з базами даних і візуальними інструментами проєктування. Отримані результати можуть бути основою для подальшої розробки повноцінного програмного продукту.

ЗМІСТ

ВСТУП

Розвиток цифрових технологій створює нові можливості для вдосконалення процесів спостереження за природними явищами. Однією з важливих складових є оперативне та точне інформування про стан погодних умов у реальному часі. Ця інформація є критично важливою для багатьох галузей: сільського господарства, транспорту, енергетики, будівництва та екологічного моніторингу. З огляду на це, підвищується попит на ефективні автоматизовані системи моніторингу погоди, здатні забезпечити своєчасний збір, зберігання, обробку та подання метеорологічних даних.

Сучасні інформаційні технології дозволяють створювати такі системи з використанням мікроконтролерів, датчиків, бездротових каналів передачі даних, хмарного зберігання інформації та візуалізації через вебінтерфейси або мобільні додатки. Це забезпечує високу точність, масштабованість та зручність у використанні подібних рішень.

Метою даної кваліфікаційної роботи є розробка автоматизованої системи моніторингу погодних умов, яка дозволяє збирати дані з контрольних точок, обробляти їх, зберігати у базі даних та надавати доступ користувачам через зручний інтерфейс.

Для досягнення поставленої мети необхідно виконати наступні завдання:

провести аналіз предметної області та огляд існуючих рішень;

сформувати функціональні та нефункціональні вимоги до системи;

побудувати систему діаграм, що моделюють логіку і архітектуру системи;

спроєктувати реляційну базу даних та реалізувати її у середовищі MySQL;

створити SQL-запити для вибірки та аналізу даних;

розробити макет користувацького інтерфейсу за допомогою Figma;

задокументувати усі етапи проєктування та реалізації системи.

Об’єктом дослідження є процес автоматизованого збору, зберігання та подання метеорологічних даних.

Предметом дослідження є розробка інформаційної системи для моніторингу погодних умов.

У роботі використано методи системного та об'єктно-орієнтованого аналізу, структурного моделювання (UML, BPMN, DFD), проєктування баз даних, SQL-аналітики та прототипування користувацького інтерфейсу.

Практична значущість розробки полягає у можливості її застосування для оперативного отримання погодної інформації, що може бути використано в системах управління міською інфраструктурою, сільськогосподарських підприємствах, транспортній логістиці та службах цивільного захисту.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Особливості моніторингу погодних умов в інформаційних системах

Моніторинг погодних умов є критично важливою складовою багатьох галузей, включаючи сільське господарство, транспорт, енергетику, будівництво, екологію та охорону здоров'я. Точне й своєчасне отримання даних про погодні умови дозволяє не лише планувати діяльність, а й запобігати можливим ризикам, що виникають через несприятливі метеорологічні явища. У зв'язку з цим автоматизація процесу моніторингу погодних умов стає все більш актуальною, оскільки дозволяє збирати, обробляти та аналізувати великі обсяги даних у режимі реального часу.

Інформаційні системи моніторингу погоди зазвичай складаються з таких основних компонентів: апаратної частини, що включає сенсори та метеостанції; програмного забезпечення для збору, збереження, обробки та візуалізації даних; баз даних, що забезпечують збереження історичних показників; користувацького інтерфейсу, який дозволяє переглядати й аналізувати інформацію.

Однією з ключових особливостей погодного моніторингу є різноманітність джерел даних. Показники можуть надходити як із локальних сенсорів, так і з відкритих зовнішніх API, супутникових систем, державних метеослужб. Це потребує від інформаційної системи здатності до інтеграції з різними протоколами передачі даних та до адаптації під різні формати вхідної інформації.

Сучасні погодні інформаційні системи мають функціонал не лише для відображення поточних показників, але й для аналізу історичних даних, побудови графіків змін, прогнозування тенденцій, формування сповіщень про критичні значення. Це досягається завдяки застосуванню баз даних, SQL-запитів, модулів аналітики та спеціалізованих алгоритмів.

Інформаційні системи моніторингу погоди можуть бути як вузькоспеціалізованими (наприклад, для аграрного бізнесу), так і універсальними. Універсальні рішення зазвичай передбачають масштабовану архітектуру, можливість налаштування під конкретні потреби користувача,

модульну структуру, що дозволяє додавати нові сенсори або підключати додаткові зовнішні сервіси.

Ще однією особливістю є потреба у високій надійності та безперервності роботи. Погодні умови змінюються постійно, тому будь-які збої в системі можуть призвести до втрати важливих даних. Тому такі системи мають передбачати механізми резервного копіювання, журналювання подій, повторну передачу інформації у разі збою зв'язку.

Інформаційні системи моніторингу погоди також мають відповідати сучасним вимогам до безпеки: автентифікація користувачів, контроль доступу до інформації, шифрування конфіденційних даних. У випадку використання хмарних рішень слід враховувати питання захисту персональних даних відповідно до чинного законодавства.[1]

У контексті кліматичних змін і зростаючої кількості екстремальних погодних явищ зростає роль локалізованих систем моніторингу, які дозволяють отримувати дані з високою просторовою роздільною здатністю. Такі системи є особливо корисними в гірських районах, прибережних зонах, а також у містах, де погодні умови можуть суттєво відрізнятися на коротких відстанях.

Інформаційні системи погодного моніторингу використовуються як державними службами, так і приватними підприємствами. Вони можуть бути частиною більших комплексів управління, наприклад, смарт-міст або інтелектуальних сільськогосподарських систем. Інтеграція з мобільними додатками, картографічними сервісами, системами оповіщення дозволяє зробити такі рішення максимально корисними для кінцевих користувачів.

Окрему увагу слід приділити ролі штучного інтелекту в аналізі погодних даних. Сучасні алгоритми машинного навчання дозволяють виявляти закономірності, будувати коротко- та довгострокові прогнози на основі великих масивів даних. Це відкриває нові можливості для розвитку інформаційних систем у сфері моніторингу.[2]

Підсумовуючи, можна зазначити, що особливості моніторингу погодних умов в інформаційних системах полягають у багатофункціональності,

необхідності надійної передачі та зберігання даних, інтеграції з різними джерелами, дотриманні вимог безпеки, масштабованості та адаптивності до потреб користувачів. Реалізація таких систем вимагає поєднання знань у галузях IT, метеорології, баз даних та проєктування користувацьких інтерфейсів.[3]

1.2. Аналіз існуючих рішень та підходів до побудови погодних сервісів

У сучасному цифровому середовищі існує велика кількість погодних сервісів, які дозволяють отримувати метеорологічні дані в режимі реального часу, формувати прогнози, аналізувати тенденції та будувати кліматичні моделі. Дані сервіси використовуються як великими компаніями, так і приватними користувачами для прийняття щоденних рішень. Найбільш відомими серед них є OpenWeather, AccuWeather, Weather.com, Weather Underground, Visual Crossing, Climacell (нині Tomorrow.io), Stormglass, MetaWeather та ін.

OpenWeather є одним з найпопулярніших погодних API, що надає доступ до даних про поточну погоду, прогнозів на 7-16 днів, історичних даних, індексів забруднення повітря та УФ-індексів. Його API дозволяє використовувати як координати, так і назву міста, підтримує кілька форматів (JSON, XML), що робить його універсальним інструментом для розробників. Проте, у безкоштовній версії існують суттєві обмеження щодо частоти запитів, а історичні дані обмежені певним періодом.[4]

AccuWeather забезпечує високоточні прогнози погоди, включаючи погодні умови по годинах, попередження про надзвичайні явища, графічне представлення температурних змін. Інтерфейс API є платним, з чітким розмежуванням по функціональності. Основною перевагою є якість прогнозів, однак це робить його менш привабливим для студентських або освітніх проєктів.

Weather.com та Weather Underground, які належать компанії IBM, використовують супутникові дані та власну метеорологічну мережу. Їхній API дозволяє створювати аналітичні платформи з деталізацією до 1 км. Вони забезпечують не лише поточну погоду, а й метеорологічну історію та кліматичні

моделі, інтеграцію з бізнес-аналізом. Недоліком є обмежена доступність для нових користувачів.

Visual Crossing пропонує інструменти для роботи з погодними даними у форматах CSV, JSON, Excel, що дозволяє будувати інтерактивні звіти. Його API дозволяє здійснювати пошук даних за адресами, координатами або поштовими кодами. Доступна інтеграція з Power BI, Excel, R, Python. Це рішення орієнтоване на бізнес-аналітику, однак його точність у деяких регіонах залишає бажати кращого.

Climacell, який змінив назву на Tomorrow.io, спеціалізується на мікрокліматичних прогнозах і підтримує моделі, адаптовані до інфраструктури клієнта. Цей сервіс має унікальні модулі прогнозування для будівництва, авіації, сільського господарства. Його особливість — можливість налаштування критичних значень, при досягненні яких система надсилає сповіщення.

Stormglass - приклад сервісу, який фокусується на морських погодних умовах. Він збирає дані з морських буїв, портів, супутників і надає прогноз для координат океану. Сервіс активно використовується в морській логістиці, навігації, рибальстві. Він має відкритий API і підтримує інтеграцію з мобільними додатками.

MetaWeather - повністю безкоштовний API, який не потребує ключа доступу. Він ідеально підходить для тестування та прототипування, однак кількість доступних параметрів значно менша у порівнянні з комерційними рішеннями. Перевагою є простота використання та низький поріг входження для новачків.[5]

Окремим напрямом є використання погодних даних у складі інших екосистем -наприклад : Google Weather API (через Android SDK), Apple WeatherKit або Microsoft Weather Services. Вони не надають відкритого доступу, однак активно використовуються в екосистемах смартфонів, смарт-годинників, автонавігації.

Загальні підходи до побудови погодних сервісів включають:

Збір даних із сенсорів, супутників, метеостанцій;

Централізовану обробку даних на серверах або в хмарних середовищах;

Формування інтерфейсів API (REST, GraphQL) для сторонніх розробників;

Інтеграцію з візуалізаційними та аналітичними платформами (наприклад, Tableau, Power BI);

Створення мобільних і веб застосунків для користувачів.

Усі ці сервіси мають одну спільну рису - вони здебільшого не належать користувачам. Це означає, що при використанні зовнішніх погодних API ми залежимо від їхніх умов, тарифів, точності та доступності. Це може створювати ризики, особливо у випадках критичної інфраструктури або бізнес-процесів, де переривання доступу до даних є неприйнятним.

Розробка власної системи моніторингу погоди дозволяє уникнути цих обмежень. Власні сенсори, локальне зберігання даних, адаптований інтерфейс - усе це дає повний контроль над функціональністю. Це важливо у сільському господарстві, де локальна точність має велике значення, в освітніх проєктах де важливе самостійне налаштування компонентів, у дослідницькій діяльності - де потрібна гнучкість аналізу.

Отже, аналіз існуючих погодних сервісів дозволяє зробити висновок, що хоча на ринку існує велика кількість інструментів, жоден з них не поєднує повну гнучкість, безкоштовність, локалізацію та простоту налаштування одночасно. Саме тому доцільно розробити власну інформаційну систему моніторингу погоди, яка буде враховувати специфіку регіону, потреби користувача та забезпечувати стабільну, незалежну роботу навіть без доступу до Інтернету. [6]

1.3. Визначення вимог до автоматизованої системи моніторингу погоди

Розробка будь-якої інформаційної системи починається з етапу формалізації вимог, які визначають її функціональність, надійність, технічні характеристики та інтерфейсні особливості. Вимоги є основою для проєктування, реалізації, тестування та подальшої експлуатації програмного забезпечення. У випадку створення автоматизованої системи моніторингу погодних умов особливо важливо врахувати специфіку роботи з сенсорними даними, часовою залежністю інформації, вимогами до точності та оперативності обробки.

Функціональні вимоги визначають, яку саме роботу повинна виконувати система. Вони охоплюють основну функціональність, яку очікує кінцевий користувач. До функціональних вимог автоматизованої системи моніторингу погоди можна віднести:

реєстрацію та авторизацію користувачів з різними ролями (адміністратор, спостерігач, гість);

підключення та обробку даних з метеорологічних сенсорів (температура, вологість, тиск, швидкість вітру);

збереження даних у базі для подальшої обробки та аналізу;

формування графіків, діаграм і звітів за вказаний період;

експорт інформації у формати CSV, PDF або XLSX;

створення сповіщень про досягнення критичних значень погодних показників;

відображення поточних даних в режимі реального часу;

підтримку декількох мов інтерфейсу (українська, англійська).

Нефункціональні вимоги не описують конкретних функцій, але визначають характеристики системи. У контексті погодного моніторингу це такі вимоги:

надійність - система повинна стабільно працювати упродовж тривалого часу без збоїв;

масштабованість - можливість додавання нових сенсорів, модулів або функцій без зміни архітектури;

безпека - автентифікація користувачів, шифрування конфіденційних даних, захист від несанкціонованого доступу;

продуктивність - система повинна забезпечувати обробку великої кількості записів без суттєвих затримок;

зручність користування - інтуїтивно зрозумілий інтерфейс, логічна структура елементів управління.

Також слід врахувати вимоги до технічного забезпечення:

- наявність сенсорів температури, вологості, атмосферного тиску (наприклад, DHT22, BME280);
- контролер для збору даних (наприклад, Arduino, Raspberry Pi, ESP32);
- модуль зв’язку (Wi-Fi, Ethernet або GSM) для передачі даних;
- сервер або хмарне сховище для бази даних (наприклад, Microsoft SQL Server, PostgreSQL);
- вебсервіс для обробки запитів (Node.js, Python Flask, ASP.NET);
- клієнтський інтерфейс (React, Vue або прості HTML+CSS макети).

Особливу увагу слід приділити вимогам до бази даних. Вона має підтримувати:

реляційну структуру з таблицями сенсорів, показників, користувачів, сеансів з’єднання;

індексацію по часових мітках для пришвидшення запитів;

резервне копіювання;

можливість масштабування без втрати даних. [7]

Щодо вимог до користувацького інтерфейсу, то він має бути адаптивним, легким для сприйняття, з мінімальною кількістю переходів. Наприклад, головна сторінка може містити поточні дані, графік за добу та меню для вибору історичних даних або налаштувань.

Додатково, система має враховувати умови відсутності зв’язку або живлення. У таких випадках дані мають зберігатися локально й передаватися після відновлення з’єднання. Це особливо важливо для віддалених станцій або в умовах польових вимірювань.

При визначенні вимог також необхідно враховувати потенційних користувачів:

- для аграрія - важлива деталізація даних, можливість отримувати SMS-повідомлення;
- для дослідника - зручна аналітика, порівняння між періодами;
- для студента - навчальна простота, можливість симулювати роботу системи;
- для адміністратора - контроль доступу, лог подій, налаштування системних параметрів.

Таким чином, визначення вимог є ключовим етапом, що дозволяє уникнути непорозумінь у майбутньому, забезпечити узгодженість між замовником, користувачем та розробником. Якісно сформульовані вимоги стають фундаментом для побудови технічного завдання та подальшого ефективного розвитку програмного продукту. [8]

1.4. Постановка задачі дипломного проєкту

Постановка задачі - це формулювання мети, задач і очікуваних результатів проєкту, що реалізується в межах дипломної роботи. Вона є завершальним етапом аналітичної частини і дозволяє узагальнити отриману інформацію про предметну область, вимоги до системи та технічні можливості для її реалізації.

Основна мета дипломного проєкту полягає у розробці автоматизованої інформаційної системи, яка дозволяє здійснювати моніторинг погодних умов у режимі реального часу, зберігати та аналізувати історичні дані, забезпечувати візуалізацію показників у зручному форматі, а також генерувати звіти та аналітику для користувачів.[2]

Для досягнення поставленої мети необхідно вирішити низку підзадач:

- здійснити аналіз предметної області, вивчити існуючі аналоги та технології збору погодних даних;

- визначити вимоги до функціонування інформаційної системи;
- розробити архітектуру системи та структуру її компонентів;
- побудувати логічні та фізичні моделі бази даних для збереження показників погоди;
- створити прототип інтерфейсу користувача для відображення поточних і архівних погодних даних;
- реалізувати модулі введення, збереження та виводу інформації;
- протестувати систему на відповідність функціональним і нефункціональним вимогам;
- сформувати висновки щодо ефективності створеної системи та можливості її подальшого розширення.[9]

Описані задачі охоплюють повний цикл життєвого циклу створення інформаційної системи: від аналізу потреб до тестування і висновків. Важливою частиною є проєктування бази даних, що дозволяє зберігати значні обсяги даних з урахуванням часових міток, сенсорних джерел та параметрів. Дані повинні бути структурованими, доступними для фільтрації, сортування та агрегації.

Наступною задачею є розробка користувацького інтерфейсу. Його необхідно спроєктувати так, щоб користувачі могли легко отримати необхідну інформацію без спеціальної підготовки. Прототипування інтерфейсу буде здійснюватися за допомогою Figma, що дозволяє створювати інтерактивні моделі для десктопної та мобільної версій.

Технічно система має підтримувати реляційну модель даних, SQL-запити до бази, а також мати змогу інтегруватися з сенсорними пристроями або зовнішніми API для підключення реальних погодних джерел. Це дозволить адаптувати рішення як до лабораторного середовища, так і до практичного використання.

З точки зору життєвого циклу програмного забезпечення, запропонована система проходить усі класичні фази: аналіз вимог, проєктування, реалізація, тестування та супровід. Постановка задачі дозволяє чітко сформулювати цілі на кожному етапі та забезпечити логічний перехід між етапами.

Визначення задач дає змогу уникнути перевантаження проєкту зайвими функціями, сконцентруватися на ключових елементах, які забезпечують цінність для користувача. Водночас, правильно сформульовані задачі слугують базою для перевірки досягнутих результатів і виступають критеріями успішності виконаної роботи.

Таким чином, постановка задачі дипломного проєкту є важливим етапом, який визначає рамки майбутньої реалізації системи, формулює очікувану функціональність, технічні параметри та вимоги до результату. Чітка, структурована постановка задач дозволяє досягти послідовності, логічності та ефективності в реалізації інформаційної системи моніторингу погодних умов.

У результаті аналізу предметної області було з’ясовано, що існуючі погодні сервіси мають низку обмежень, зокрема недостатню адаптивність та інтеграцію з локальними пристроями. Це обґрунтовує доцільність створення власної автоматизованої системи моніторингу, яка відповідає сучасним вимогам точності, гнучкості та доступності.[10]

Розділ 2. ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Загальна архітектура та функціональність системи

Для ефективного функціонування автоматизованої системи моніторингу погодних умов необхідно розробити чітку архітектуру, яка визначає компоненти системи, їхню взаємодію та відповідальність. Архітектура є фундаментом проєкту, що забезпечує масштабованість, підтримку, гнучкість та можливість модернізації.

Система передбачає клієнт-серверну модель з розділенням функціональних блоків на декілька логічних рівнів. Це дозволяє ізолювати обробку даних, зберігання інформації та візуалізацію для забезпечення безпеки, продуктивності та підтримки.

Основними компонентами системи є:

- сенсорний рівень, який включає фізичні пристрої для збору даних: датчики температури, вологості, атмосферного тиску, вітру тощо;
- мікроконтролер (наприклад, ESP32 або Arduino), що отримує сигнали з сенсорів і передає їх на сервер через Wi-Fi або GSM-модуль;
- серверна частина, відповідальна за прийом, обробку та зберігання інформації в базі даних (на прикладі Microsoft SQL Server);
- клієнтський інтерфейс, який надає користувачам змогу переглядати актуальні та архівні дані, формувати запити, аналізувати інформацію, будувати графіки.

Принципова схема архітектури передбачає три основні рівні:

1. Рівень збору даних (сенсори та мікроконтролери);
2. Рівень обробки та зберігання (сервер + база даних);
3. Рівень презентації (вебінтерфейс/мобільний додаток).

Кожен з цих рівнів виконує окремі задачі, що дозволяє масштабувати систему незалежно: додавати нові сенсори, змінювати базу даних, оновлювати інтерфейс без потреби повної перебудови. [11]

Функціонально система реалізує:

реєстрацію та авторизацію користувачів;

підключення до сенсорів для збору даних;

автоматичний запис показників у базу кожні X хвилин;

візуалізацію поточних і архівних значень за добу/тиждень/місяць;

формування та експорт звітів;

налаштування критичних порогів із відправкою сповіщень;

адміністрування користувачів, перегляд журналу подій.

Архітектура системи має враховувати й можливість віддаленого доступу через веб, тому сервер має бути захищеним, підтримувати авторизацію, шифрування з'єднання (HTTPS) та ведення логів.

У якості платформи для реалізації обрано Microsoft SQL Server, оскільки він забезпечує високу продуктивність, гнучкі засоби для роботи з реляційними структурами, розвинуту систему прав доступу та резервного копіювання. Для клієнтського інтерфейсу створюється прототип у Figma, що дозволяє визначити логіку взаємодії користувача з системою ще до її реалізації.

Система орієнтована на декілька типів користувачів:

адміністратор має повний доступ до налаштувань, перегляду логів, створення нових користувачів;

звичайний користувач переглядає погодні дані, формує звіти;

гість має обмежений доступ до поточних даних без можливості їх змінювати.

Крім того, важливо враховувати відмовостійкість. У разі втрати зв'язку або відключення живлення дані мають зберігатися локально на пристрої і передаватися пізніше. Це дозволить уникнути втрат інформації.

Функціональність системи може бути розширена в майбутньому за рахунок:

підключення зовнішніх погодних API (OpenWeather тощо);

додавання нових типів сенсорів (ультрафіолет, дощоміри);

інтеграції з іншими інформаційними системами (наприклад, системами керування сільським господарством);

розширення можливостей аналітики та прогнозування на основі машинного навчання. [12]

Отже, загальна архітектура системи охоплює всі ключові компоненти від фізичного збору даних до візуалізації результатів. Вона забезпечує гнучкість, безпеку, ефективність та можливість подальшого розвитку. Функціональність відповідає поставленим задачам та дозволяє досягти мети дипломного проєкту.

2.2. Діаграма прецедентів (Use Case Diagram)

Діаграма прецедентів (Use Case Diagram) є однією з ключових складових моделювання інформаційної системи в нотації UML. Вона призначена для візуалізації функціональних вимог до системи з позиції зовнішніх користувачів акторів (Actors), які взаємодіють з нею. Діаграма дозволяє ідентифікувати основні сценарії використання, встановити взаємозв’язки між користувачами та функціональними можливостями, що реалізуються у системі.

У випадку автоматизованої системи моніторингу погодних умов передбачається, що існує декілька типів користувачів. Серед них адміністратор, зареєстрований користувач та гість. Кожен з них має власний набір прав доступу до функціоналу системи.

Основні прецеденти, які реалізовані в межах проєкту:

- Авторизація користувача;
- Перегляд поточних погодних умов;
- Аналіз історичних даних;
- Формування звітів;
- Налаштування параметрів відображення;
- Отримання сповіщень про критичні зміни погодних умов;
- Управління обліковими записами користувачів (доступно лише адміністратору);
- Вивантаження даних в CSV або PDF;
- Перегляд діаграм та графіків динаміки змін параметрів. [13]

Діаграма прецедентів дозволяє чітко окреслити, які функції є загальнодоступними, які вимагають авторизації, а також які зарезервовані виключно для адміністративного рівня. Це дає змогу структурувати логіку взаємодії системи з різними категоріями користувачів.

Наприклад, гість може переглядати лише загальну інформацію про погоду в реальному часі без доступу до історичних даних. Зареєстрований користувач має доступ до розширених функцій: аналітики, звітності, персональних налаштувань. Адміністратор має найвищий рівень доступу і може редагувати базу, додавати або блокувати користувачів, змінювати параметри з’єднання з сенсорами, встановлювати порогові значення для оповіщення.

На діаграмі актори позначаються у вигляді людських піктограм, а сценарії у вигляді овалів з назвами операцій. Стрілки демонструють взаємозв’язок між акторами і прецедентами. Іноді використовуються залежності типу «include» або «extend», які відображають вкладеність чи додаткові варіанти реалізації сценаріїв.

Use Case діаграма також є базисом для подальшого створення діаграм активностей, послідовностей, класів. Вона відіграє роль точки відліку при декомпозиції функціоналу на технічні завдання. [14]

Таким чином, діаграма прецедентів - це не лише інструмент для візуалізації функцій, але й основа для архітектурного планування, оцінки складності системи, розподілу обов’язків між компонентами програмного забезпечення.

Нижче наведено приклад діаграми прецедентів, створеної на основі описаних вище функціональних вимог системи моніторингу погодних умов [Рис2.1].

Рисунок 2.1 – Діаграма прецедентів автоматизованої системи моніторингу погодних умов

2.3. Моделювання бізнес-процесів (BPMN)

Для побудови логіки роботи автоматизованої системи моніторингу погодних умов необхідно описати бізнес-процеси, які забезпечують функціонування системи на різних рівнях. Для цього застосовується нотація BPMN (Business Process Model and Notation), яка є стандартом де-факто для моделювання процесів у сучасних інформаційних системах.

BPMN дозволяє створювати графічні моделі, зрозумілі як технічним спеціалістам, так і кінцевим користувачам. У межах цієї нотації процеси поділяються на логічні блоки: події, дії, рішення, потоки, учасники процесу (Pools/Lanes). Це дозволяє не тільки візуалізувати послідовність виконання дій, а й чітко окреслити відповідальність за кожну з них.

У межах даного проєкту було змодельовано декілька бізнес-процесів, серед яких основним є процес отримання погодної інформації користувачем. Учасниками (акторами) процесу є: Сенсор, Система моніторингу, Користувач, Адміністратор.

Типовий бізнес-процес виглядає так:

1. Сенсор фіксує показники температури, вологості, тиску в контрольній точці.
2. Мікроконтролер зчитує значення і передає їх у серверну частину.
3. Система перевіряє коректність отриманих даних.
4. Дані записуються у базу даних.
5. Користувач через інтерфейс надсилає запит на перегляд погодних даних.
6. Система формує відповідь на запит.
7. Результати виводяться у вигляді таблиці або графіка.

Окремо змодельовано бізнес-процес адміністрування системи, що включає:

перевірку статусу підключених сенсорів;

додавання нових точок збору даних;

модифікацію параметрів збору;

створення облікових записів користувачів.

Усі процеси представлені у вигляді BPMN-діаграм. Основними елементами є:

Start Event (початкова подія) -ініціація процесу, наприклад, запит користувача;

Task (завдання) - конкретна дія, наприклад, “Зчитати дані з сенсора”;

Gateway (умовний розгалужувач) - перевірка, наприклад, “Чи дані коректні?”;

End Event - завершення процесу, наприклад, “Відповідь виведено”;

Pools and Lanes - окремі блоки для кожного учасника процесу. [15]

Побудова таких діаграм дозволяє бачити всі можливі варіанти проходження процесу, передбачити відмови або помилки, внести поліпшення у логіку взаємодії між компонентами.

BPMN також підтримує розширення у вигляді повідомлень між процесами (Message Flow), таймерів (Timer Events), зовнішніх подій (Signal Events), що робить її універсальним інструментом для побудови складних сценаріїв.

У розробленій системі, що описується у цій дипломній роботі, основний BPMN-процес реалізує сценарій “Отримання погодної інформації за запитом”. Він включає п'ять основних учасників: користувача, інтерфейс, обробник запитів, базу даних та систему збору даних. Це дозволяє відстежити шлях кожного елемента від ініціації до завершення.

Крім основного сценарію, були змодельовані додаткові процеси:

оновлення даних у випадку відсутності сигналу;

обробка запиту на звіт;

видалення застарілих даних з бази;

контроль активності користувачів. [16]

Варто зазначити, що використання BPMN-моделей дозволяє уникнути непорозумінь між розробником, замовником і кінцевим користувачем. У процесі розробки системи вони можуть використовуватись як живий документ: змінюватися, адаптуватися, деталізуватися за потреби [Рис 2.2].

Таким чином, моделювання бізнес-процесів у вигляді BPMN-діаграм є ключовим етапом у побудові сучасної автоматизованої інформаційної системи. Воно забезпечує прозорість, прогнозованість і контроль кожного кроку взаємодії користувача з системою [Рис2.3].

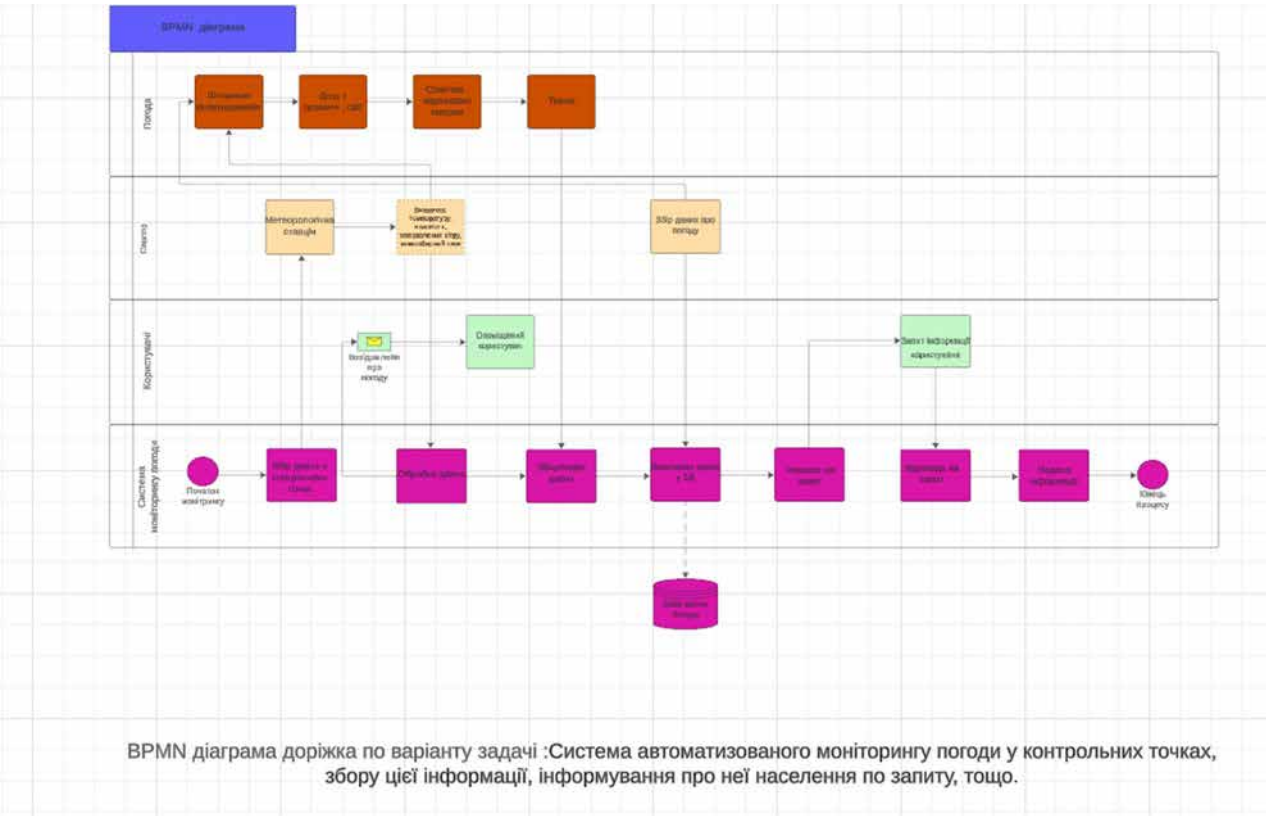

Рисунок. 2.2 – BPMN-діаграма бізнес-процесу моніторингу погодних умов

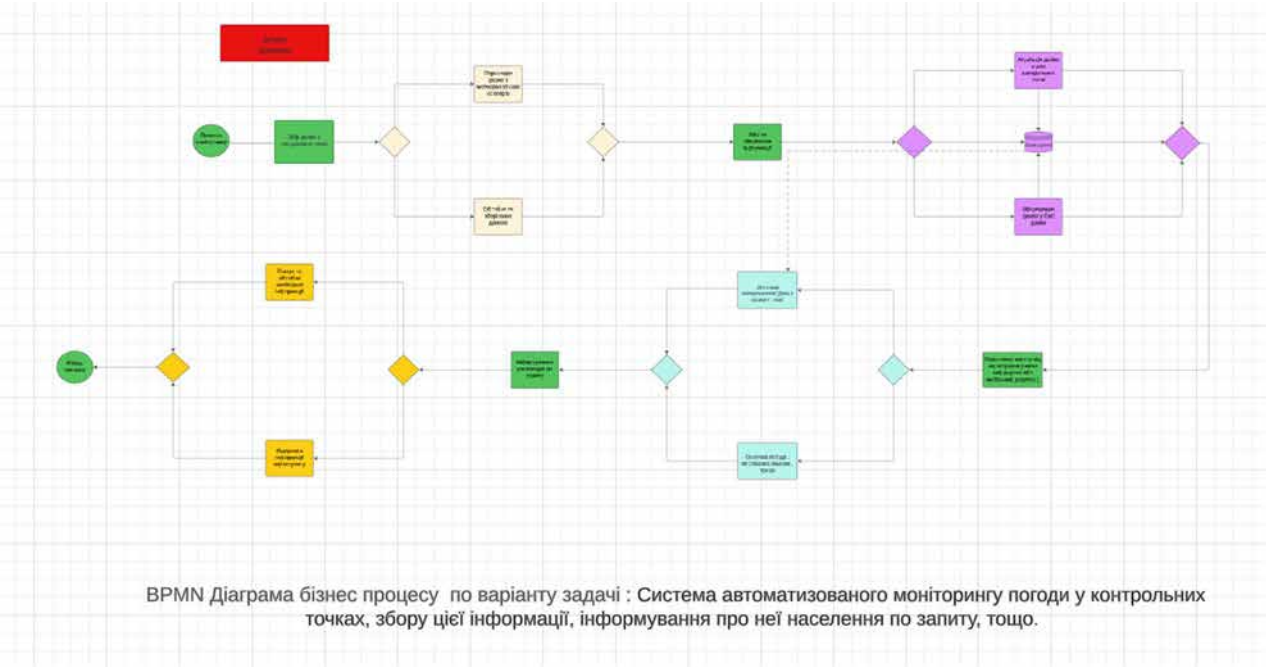

Рисунок. 2.3 – BPMN-діаграма бізнес-процесу моніторингу погодних умов

2.4. Діаграма потоків даних (DFD)

Для детального моделювання структури обробки даних у системі моніторингу погоди було використано діаграму потоків даних (Data Flow Diagram, DFD). Цей тип діаграми дозволяє відобразити, як інформація передається між різними елементами системи: зовнішніми джерелами, процесами, базами даних і результатами обробки.

DFD є особливо корисною на етапі проєктування, оскільки дозволяє побачити логіку обміну даними ще до написання коду. У межах дипломного проєкту була побудована DFD-діаграма першого рівня, яка показує основні потоки інформації між ключовими компонентами.

До елементів, що були враховані у діаграмі, належать:

Сенсори - надсилають погодні дані (температура, вологість, тиск тощо).

Система збору даних - отримує вхідні дані, перевіряє їх на коректність та передає у сховище. [17]

База даних - містить усі зібрані метеорологічні показники.

Користувач - взаємодіє з інтерфейсом і формує запити на перегляд інформації.

Модуль обробки запитів - шукає потрібні дані у базі, формує відповідь та відображає її у зрозумілому вигляді.

У центрі діаграми розташовано основний процес - обробка запиту користувача, яка включає:

1. Прийом запиту через інтерфейс;
2. Звернення до бази даних;
3. Формування відповіді;
4. Виведення результату на екран. [18]

Окремо також показано потік вхідних даних від сенсорів, що надходять до системи автоматично і зберігаються в реальному часі.[Рис2.4]

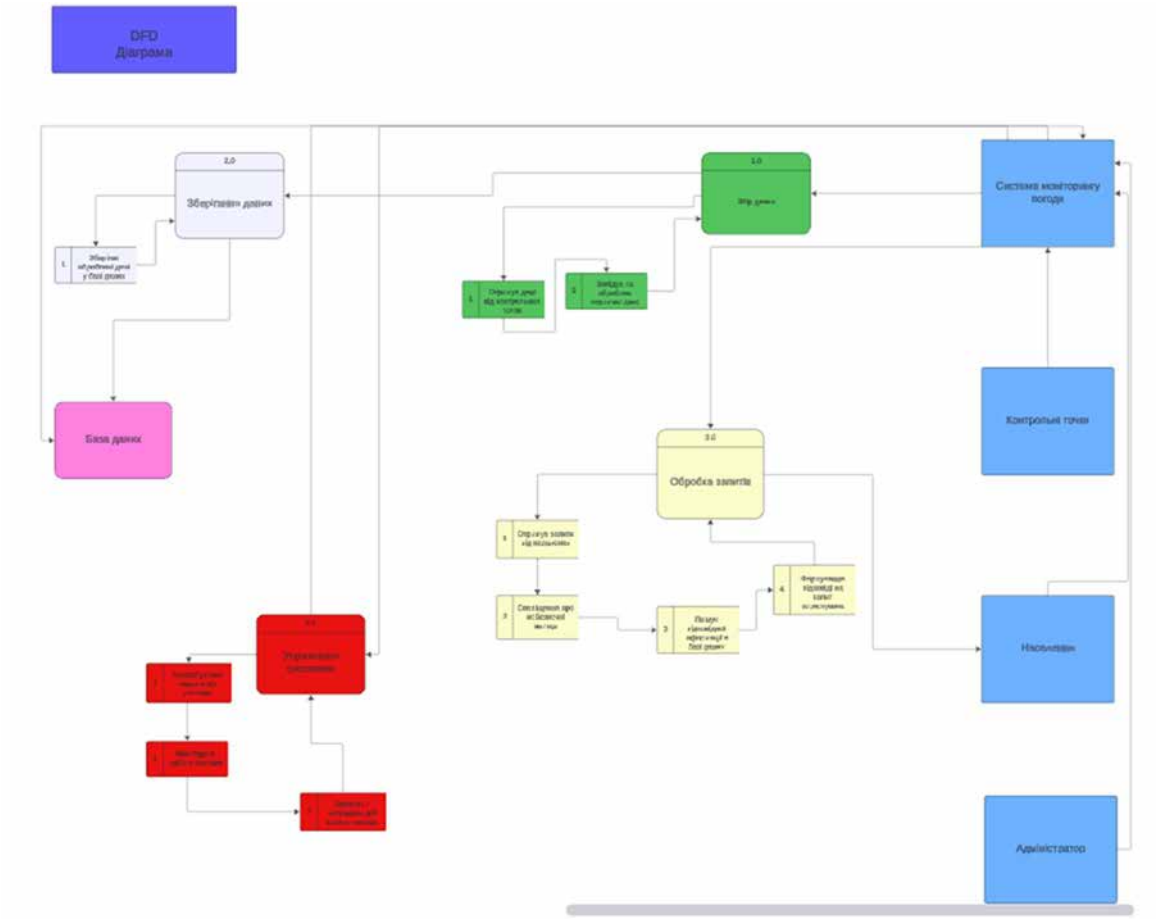

Рисунок. 2.4 – Діаграма потоків даних (DFD) автоматизованої системи моніторингу погоди

2.5. Діаграма класів (Class Diagram)

Діаграма класів є однією з основних складових моделювання об’єктно-орієнтованих систем. Вона дозволяє представити структуру об’єктів, які будуть використовуватись у системі, зв’язки між ними, їхні атрибути і методи. У дипломному проєкті було створено діаграму класів, яка відображає основні елементи системи моніторингу погодних умов та взаємодію між ними.

Кожен клас на діаграмі відповідає певному логічному об’єкту, який виконує конкретну функцію в системі. Наприклад, клас User описує користувача, клас Sensor - погодний сенсор, а WeatherData - набір погодних показників.

Клас User містить атрибути: UserID, Name, Email, і метод для отримання погодних даних. Він є важливим, бо саме через цього об'єкта користувач взаємодіє із системою. З ним пов’язаний клас Request, який відображає запит користувача на дані - у ньому зберігаються ID запиту, дата, місце і самі параметри.

Клас Sensor містить такі поля, як SensorID, тип сенсора, його статус і час передачі даних. Цей клас є основним джерелом погодної інформації, яку система потім обробляє.

Клас WeatherStation поєднує в собі кілька сенсорів. Тут є атрибути: StationID, місце розташування, список сенсорів. Він також має методи для запиту погодних даних від усіх підключених сенсорів.

Клас WeatherData включає такі параметри як температура, вологість, тиск, швидкість вітру, час фіксації. Цей клас використовується при формуванні відповіді користувачеві, а також при збереженні в базу.

Ще один важливий клас Notification. Він відповідає за повідомлення користувача в разі певних погодних умов, які вийшли за допустимі межі. Тут є атрибути NotificationID, дата й час, текст повідомлення. [19]

Класи DataStorage та Database описують, як дані зберігаються. Перший із них має методи збереження і запиту з бази, другий уже ближчий до фізичної реалізації бази даних. Вони важливі для того, щоб погодна інформація не втрачалась і її можна було отримати в будь-який момент.

Клас DataProcessing реалізує обробку: фільтрацію, перетворення одиниць вимірювання, формування звіту. Він взаємодіє з WeatherData, Request та Database.

Зв’язки між класами відображені за допомогою стрілок. Наприклад, User пов’язаний з Request, Sensor з WeatherStation, а DataStorage з базою даних. Усе це дає змогу зрозуміти, як інформація рухається по системі.

Особливу увагу варто звернути на методи. Наприклад, у класі Notification є метод для збереження сповіщення, а в класі Request метод для обробки запиту.

Такі методи дозволяють реалізувати логіку системи й описати функціональність в об’єктному вигляді. [20]

Загалом, діаграма класів допомогла структурувати систему, поділити її на логічні частини, кожна з яких має власну зону відповідальності. Це полегшило подальше проєктування бази даних, розробку інтерфейсу та написання SQL-запитів.

На [Рис.2.5] нижче наведена створена діаграма класів, яка демонструє всі основні класи, атрибути, методи та зв’язки між ними.

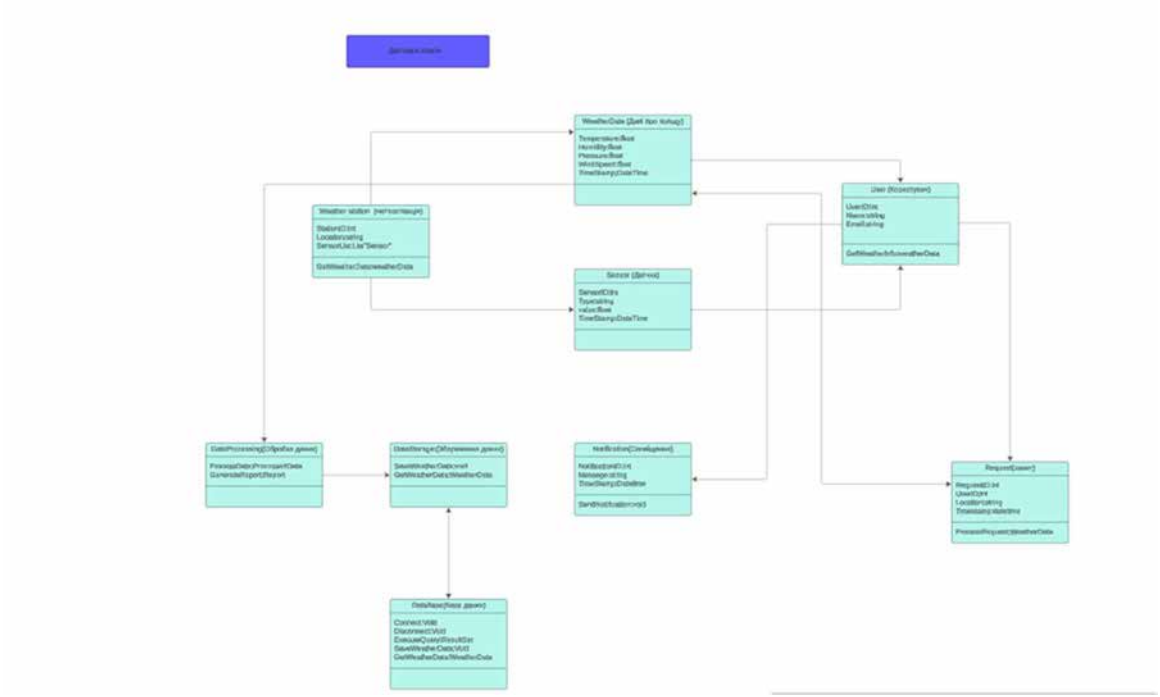

Рисунок. 2.5 – Діаграма класів автоматизованої системи моніторингу погодних умов

2.6. Діаграма послідовностей (Sequence Diagram)

Діаграма послідовностей - це один з основних типів діаграм UML, яка використовується для моделювання взаємодії між об’єктами в часі. Вона показує, які саме повідомлення передаються між об’єктами та в якому порядку це відбувається. На відміну від діаграми класів, де акцент зроблено на структурі системи, діаграма послідовностей дозволяє розкрити динаміку її роботи.

У контексті автоматизованої системи моніторингу погодних умов ця діаграма демонструє, як саме відбувається взаємодія між користувачем, системою, базою даних, сенсорними модулями та іншими компонентами. Вона є особливо корисною для розуміння процесу обробки запитів, передачі даних та відповіді системи.

На діаграмі чітко відображені основні учасники процесу:

Користувач (ініціює запит на погодні дані);

Система моніторингу (приймає запит і керує взаємодією);

Контрольна точка (віртуальний сенсор);

Інформаційний модуль (відповідає за логіку відповіді);

База даних (джерело історичних даних);

Модуль сповіщення (надсилає повідомлення користувачеві).

Основний сценарій виглядає наступним чином:

1. Користувач надсилає запит на перегляд погодних даних.
2. Система моніторингу отримує запит і звертається до контрольної точки.
3. Контрольна точка виконує зчитування поточних даних.
4. Дані з контрольної точки передаються назад у систему.
5. Система звертається до бази даних для отримання додаткових даних.
6. База надсилає дані назад до інформаційного модуля.
7. Інформаційний модуль обробляє дані, формує відповідь.
8. Система надсилає готову відповідь користувачеві.
9. При потребі активується модуль сповіщення.

Такий сценарій є типовим для погодних сервісів, особливо коли важлива не лише поточна інформація, але й аналітика за минулі періоди.
Кожна стрілка на діаграмі позначає повідомлення (message) між об’єктами.

Вертикальні лінії - це життєвий цикл (lifeline) об’єкта. Порядок викликів демонструється зверху вниз, що дозволяє легко простежити послідовність дій.

У нашому випадку повідомлення мають такі приклади: "Зчитати дані", "Запит у базу", "Сформувати відповідь", "Відправити сповіщення". Усі ці дії відбуваються послідовно, і кожна з них має логічне завершення.

Особливістю діаграми послідовностей у цьому проєкті є те, що вона охоплює як користувацький інтерфейс, так і внутрішню серверну логіку. Це дозволяє краще зрозуміти, як компоненти пов’язані між собою і як забезпечується надійність передачі даних. [21]

Також важливо, що дана діаграма може бути використана на етапі тестування для перевірки, чи працює система згідно з очікуваним сценарієм. Вона допомагає побачити, де можуть виникати затримки або помилки.

Ще одним важливим моментом є можливість масштабування. Якщо у майбутньому до системи буде додано, наприклад, мобільний додаток або зовнішній погодний API, то ця діаграма дозволить легко оновити послідовність без повної перебудови логіки. [22]

У висновку варто зазначити, що діаграма послідовностей відіграє важливу роль у візуалізації процесів, де бере участь багато компонентів. Вона не лише допомагає на етапі проєктування, але й є корисною для презентації роботи системи на захисті.

Нижче на [Рис.2.6] наведено створену в рамках дипломного проєкту діаграму послідовностей, яка демонструє ключовий сценарій взаємодії між користувачем і основними компонентами системи.

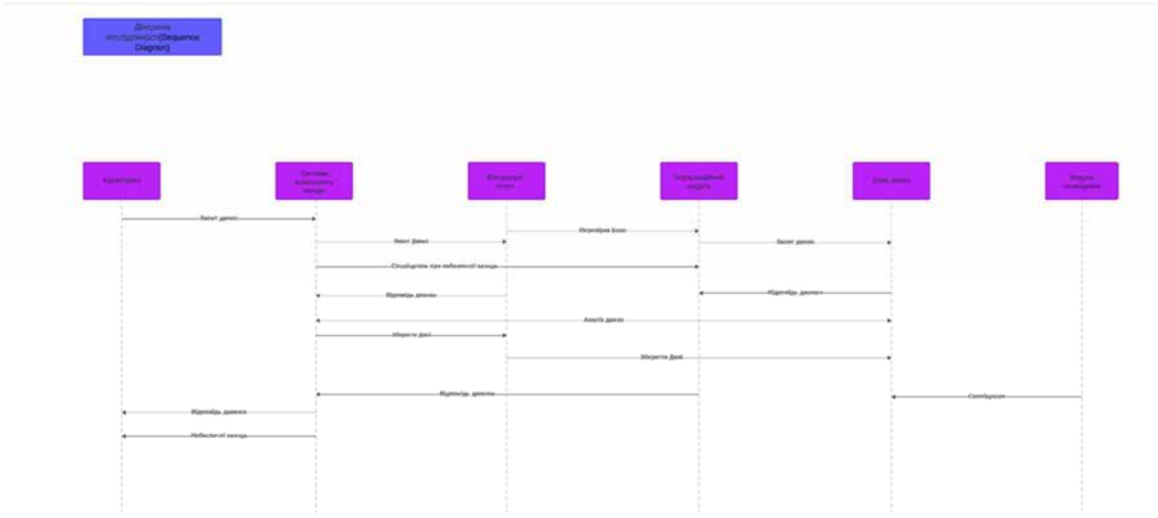

Рисунок. 2.6 – Діаграма послідовностей взаємодії в системі моніторингу погоди

2.7. ER-модель бази даних

ER-модель (Entity-Relationship Diagram) є важливим інструментом при проєктуванні структури бази даних. Вона дозволяє графічно відобразити сутності, їхні атрибути та зв’язки між ними. Це полегшує розуміння логіки зберігання даних, дозволяє уникнути помилок у структурі та забезпечує узгодженість даних у процесі розробки.

У межах даної дипломної роботи було створено ER-модель, яка охоплює основні компоненти системи моніторингу погодних умов. Вона сформована на основі логіки взаємодії між користувачем, сенсорами, точками спостереження, базою даних, а також запитами й сповіщеннями.

Головні сутності в ER-моделі:

Користувач (User): має такі поля як ID, ім’я, тип користувача, електронна пошта. Користувач може надсилати запити, отримувати дані та отримувати сповіщення. Його зв’язок з іншими таблицями через поля UserID.

Контрольні точки (MonitoringPoint): місця, де безпосередньо збираються погодні дані. Атрибути включають ID точки, назву, геолокацію, опис.

Погодні дані (WeatherData): містять температуру, вологість, тиск, швидкість вітру та інші показники. Кожен запис прив’язаний до конкретної точки і має часову мітку.

Зберігання даних (DataStorage): описує структуру і тип збереження погодної інформації, включаючи спосіб обробки та збереження в базі.

Запит (Request): описує взаємодію користувача із системою. Включає ID запиту, дату, користувача, контрольну точку.

Сповіщення (Alert): формується у випадку досягнення критичних параметрів. Зберігає ID сповіщення, тип, опис, час і ID користувача.

Обробка даних (DataProcessing): сутність, що описує логіку перетворення, обчислення та аналізу даних.

База даних (Database): центральне сховище інформації, з ідентифікатором, назвою, типом і загальним описом.

Кожна сутність у моделі має первинні ключі (Primary Key), які забезпечують унікальність записів, а також зовнішні ключі (Foreign Key), які встановлюють зв’язки з іншими таблицями. Наприклад, поле UserID у таблиці Request є зовнішнім ключем до таблиці User, що дозволяє об’єднати інформацію про користувача із його запитом. [23]

Типи зв’язків у моделі:

Один-до - багатьох: один користувач може мати багато запитів або сповіщень.

Один-до-одного: наприклад, кожен запис у таблиці DataProcessing може відповідати одному запису у WeatherData.

Багато-до-багатьох: такі зв’язки зазвичай уникаються у фізичній реалізації, але можуть бути промодельовані через проміжні таблиці.

ER-модель є зручною базою для створення фізичної структури бази даних у Microsoft SQL Server. Саме за цією моделлю формувались SQL-таблиці, прописувались запити та забезпечувалась цілісність даних.

Переваги цієї моделі:

Чітке розділення даних по сутностях;

Уникнення надмірного дублювання інформації;

Можливість масштабування (додавання нових сутностей);

Зрозумілий зв’язок між різними компонентами системи.

ER-модель також зручна для пояснення структури бази під час захисту, оскільки дозволяє наочно продемонструвати, як логічно побудована система і як усі елементи взаємодіють між собою. [24]

На рисунку нижче представлена ER-модель, створена в рамках цього дипломного проєкту, яка відображає всі основні сутності та зв’язки, що реалізовані у базі даних.[Рис.2.7]

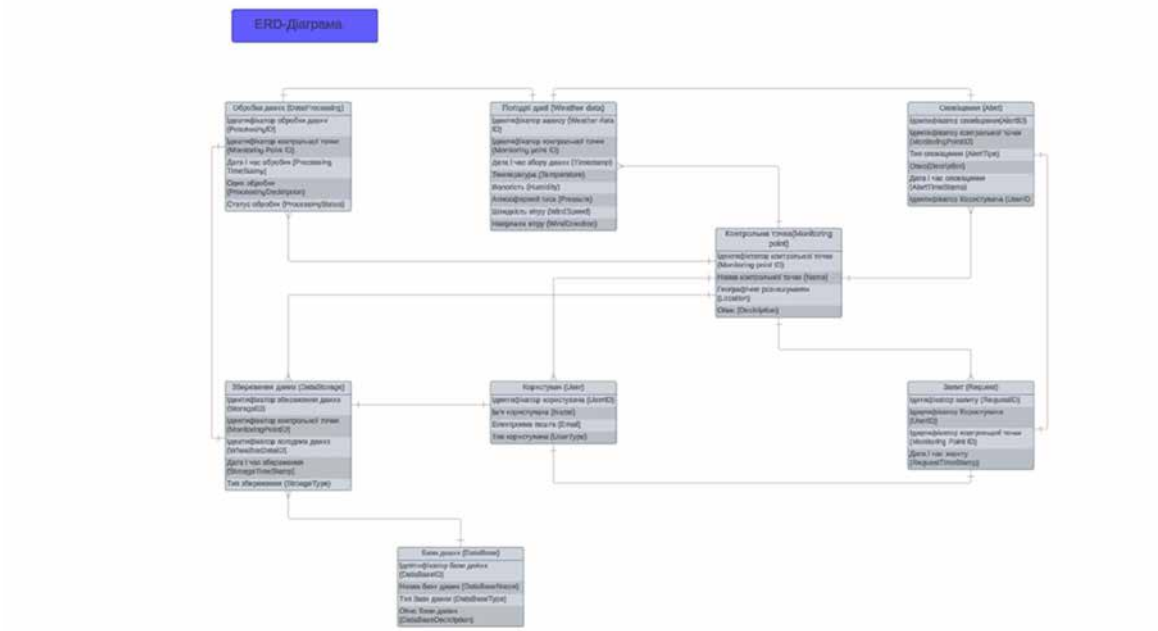

Рисунок. 2.7 – ER-модель бази даних системи моніторингу погодних умов

Під час проєктування інформаційної системи було побудовано логічну архітектуру, створено діаграми UML, BPMN, DFD, а також ER-модель бази даних. Ці результати дозволили чітко структурувати систему, визначити її компоненти та їхню взаємодію, що є важливим етапом для подальшої реалізації та тестування.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1. Створення реляційної бази даних у SQL Server

Для реалізації автоматизованої системи моніторингу погодних умов було розроблено реляційну базу даних, що забезпечує централізоване зберігання, обробку та доступ до погодної інформації. В якості платформи для реалізації обрано Microsoft SQL Server, оскільки ця система керування базами даних підтримує складні SQL-запити, процедури, зв’язки, індекси та має зручне графічне середовище адміністрування.

Процес створення бази розпочався з проєктування структури, відповідно до ER-моделі, яка попередньо була побудована. Кожна сутність моделі відображена у вигляді окремої таблиці, з чітко визначеними атрибутами та зв’язками між таблицями.

На початковому етапі була створена база даних з назвою WeatherOB2. Далі поетапно створювалися таблиці: користувачі, запити, сповіщення, погодні дані, точки моніторингу, зберігання, обробка, події.

Кожна таблиця має первинний ключ, що забезпечує унікальність записів, а також зовнішні ключі, які формують зв’язки типу один-до-багатьох між таблицями. Наприклад, таблиця Alert містить зовнішній ключ UserID, що пов’язує її з таблицею користувачів, а також MonitoringPointID, що посилається на точку збору даних.

Також передбачено таблицю Request - журнал звернень користувачів. Таблиця WeatherData пов’язана з MonitoringPoint і зберігає мітки часу та погодні параметри: температура, вологість, тиск, швидкість вітру тощо.

У таблиці DataStorage зберігається інформація про спосіб збереження погодних даних. Вона пов’язана із WeatherData, MonitoringPoint і таблицею DataBase, яка описує саму структуру бази даних. [25]

Таблиця ExtremeWeatherEvent описує події, пов’язані з аномальними погодними умовами. Вона включає тип події, опис, рівень загрози та прив’язується до конкретного запису в WeatherData.

Щоб уникнути конфліктів під час повторного запуску скриптів, попередньо реалізовано DROP-інструкції для кожної таблиці.

Створення таблиць відбувається через стандартні SQL-оператори CREATE TABLE.

Ця структура дозволяє:

швидко виконувати запити на отримання погодних даних;

фільтрувати інформацію по датах, точках, користувачах;

аналізувати історію сповіщень;

відстежувати аномальні події.

Завдяки використанню Microsoft SQL Server, база підтримує створення індексів, обмежень цілісності, каскадні оновлення та обмеження на видалення. Це підвищує надійність збереження та цілісність даних. [26]

На завершення можна сказати, що база даних повністю відповідає функціональним вимогам системи та готова до використання в аналітичних запитах, що описані в наступному підрозділі.

Код програмної реалізації форми наведено в лістингу 3.1:

Лістинг 3.1

```
DROP TABLE IF EXISTS ExtremeWeatherEvent;
DROP TABLE IF EXISTS Alert;
DROP TABLE IF EXISTS DataStorage;
DROP TABLE IF EXISTS DataProcessing;
DROP TABLE IF EXISTS WeatherData;
DROP TABLE IF EXISTS Request;
DROP TABLE IF EXISTS MonitoringPoint;
DROP TABLE IF EXISTS [User];
DROP TABLE IF EXISTS [DataBase];
```

```
CREATE TABLE [User] (
UserID INT PRIMARY KEY,
Name VARCHAR(255),
Email VARCHAR(255),
UserType VARCHAR(50)
);
CREATE TABLE MonitoringPoint (
MonitoringPointID INT PRIMARY KEY,
Name VARCHAR(255),
Location VARCHAR(255),
Description TEXT
);
CREATE TABLE [DataBase] (
DataBaseID INT PRIMARY KEY,
DataBaseName VARCHAR(255),
DataBaseType VARCHAR(50),
DataBaseDescription TEXT
);
CREATE TABLE WeatherData (
WeatherDataID INT PRIMARY KEY,
MonitoringPointID INT,
Timestamp DATETIME,
Temperature FLOAT,
Humidity FLOAT,
Pressure FLOAT,
WindSpeed FLOAT,
WindDirection VARCHAR(50),
FOREIGN KEY (MonitoringPointID) REFERENCES
MonitoringPoint(MonitoringPointID)
);
```

```
CREATE TABLE DataProcessing (
ProcessingID INT PRIMARY KEY,
MonitoringPointID INT,
ProcessingTimestamp DATETIME,
ProcessingDescription TEXT,
ProcessingStatus VARCHAR(50),
FOREIGN KEY (MonitoringPointID) REFERENCES
MonitoringPoint(MonitoringPointID)
);
CREATE TABLE Alert (
AlertID INT PRIMARY KEY,
MonitoringPointID INT,
AlertType VARCHAR(50),
Description TEXT,
AlertTimestamp DATETIME,
UserID INT,
FOREIGN KEY (MonitoringPointID) REFERENCES
MonitoringPoint(MonitoringPointID),
FOREIGN KEY (UserID) REFERENCES [User](UserID)
);
CREATE TABLE DataStorage (
StorageID INT PRIMARY KEY,
MonitoringPointID INT,
WeatherDataID INT,
StorageTimestamp DATETIME,
StorageType VARCHAR(50),
DataBaseID INT,
        FOREIGN KEY (MonitoringPointID) REFERENCES
MonitoringPoint(MonitoringPointID),
```

```
FOREIGN KEY (WeatherDataID) REFERENCES
WeatherData(WeatherDataID),
FOREIGN KEY (DataBaseID) REFERENCES [DataBase](DataBaseID)
);
CREATE TABLE Request (
RequestID INT PRIMARY KEY,
UserID INT,
MonitoringPointID INT,
RequestTimestamp DATETIME,
FOREIGN KEY (UserID) REFERENCES [User](UserID),
FOREIGN KEY (MonitoringPointID) REFERENCES
MonitoringPoint(MonitoringPointID)
);

CREATE TABLE ExtremeWeatherEvent (
EventID INT PRIMARY KEY,
WeatherDataID INT,
EventType VARCHAR(50),
EventDescription TEXT,
EventTimestamp DATETIME,
SeverityLevel VARCHAR(50),
        FOREIGN KEY (WeatherDataID) REFERENCES
        WeatherData(WeatherDataID)
        );
```

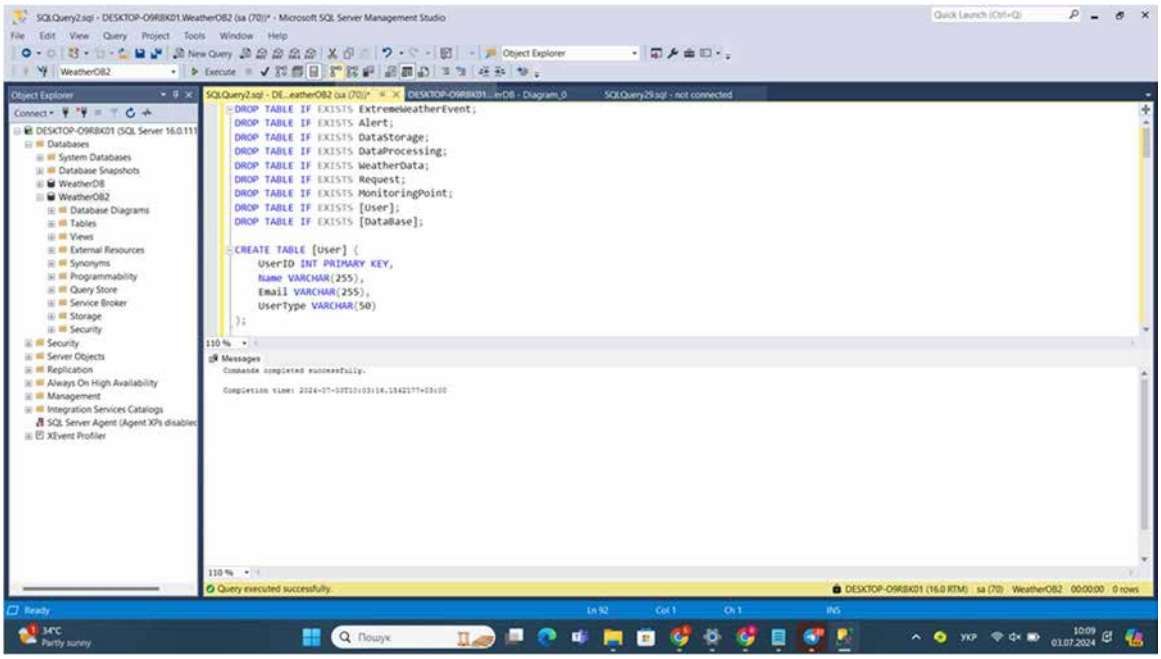

Рисунок. 3.1 – Створена база даних у середовищі SQL Server

3.2. Формування основних SQL-запитів

Після створення структури бази даних наступним етапом є написання SQL-запитів, які дозволяють здійснювати вибірку, фільтрацію, агрегацію та аналіз збережених погодних даних. У межах дипломного проєкту були реалізовані запити, які виконують логічні аналітичні дії над таблицею WeatherData.

Для прикладу візьмемо два основних завдання, які мають практичну значущість для аналізу кліматичних змін у контрольних точках.

Завдання 1. Визначити кількість днів у зимові місяці, коли температура була вище 0°C ?

Цей запит дозволяє дізнатись, наскільки часто траплялись теплі дні взимку. Це важливо для моніторингу змін клімату. Для цього використовується групування по роках і місяцях та фільтрація за температурою. [27]

Код програмної реалізації форми наведено в лістингу 3.2:

Лістинг 3.2

```
SELECT
YEAR(Timestamp) AS Year,
MONTH(Timestamp) AS Month,
COUNT() AS DaysWithTemperatureAboveZero
FROM WeatherData
WHERE MONTH(Timestamp) IN (12, 1, 2) -- Зимові місяці
AND Temperature > 0
GROUP BY YEAR(Timestamp), MONTH(Timestamp)
ORDER BY Year, Month;
```

Запит повертає кількість днів з температурою > 0 для кожного зимового місяця у різні роки.

Завдання 2. Знайти кількість днів у кожному місяці, коли вологість повітря була нижче 30% ?

Це дає змогу аналізувати періоди з надто сухим повітрям, що є важливим для аграрного сектору та здоров’я населення.

Код програмної реалізації форми наведено в лістингу 3.3:

Лістинг 3.3

```
SELECT
YEAR(Timestamp) AS Year,
MONTH(Timestamp) AS Month,
COUNT() AS DaysWithLowHumidity
FROM WeatherData
WHERE Humidity < 30
GROUP BY YEAR(Timestamp), MONTH(Timestamp)
ORDER BY Year, Month;
```

Цей запит виконує групування по роках та місяцях, з умовою відбору записів, де рівень вологості менший за 30%.

Обидва запити демонструють, як можна отримати з бази даних погодну аналітику, яка допомагає приймати рішення в реальному часі або будувати звіти.

Такі запити можуть бути використані для:

побудови графіків у веб-інтерфейсі;

формування PDF-звітів;

відправлення сповіщень при досягненні критичних умов;

довгострокового аналізу кліматичних змін. Використання SQL у системі дозволяє реалізувати логіку запитів до даних, які постійно оновлюються, і дає змогу легко масштабувати запити під нові вимоги. [28]

На [Рис.3.2] наведені скріншоти виконання вищезазначених запитів у середовищі SQL Server Management Studio.

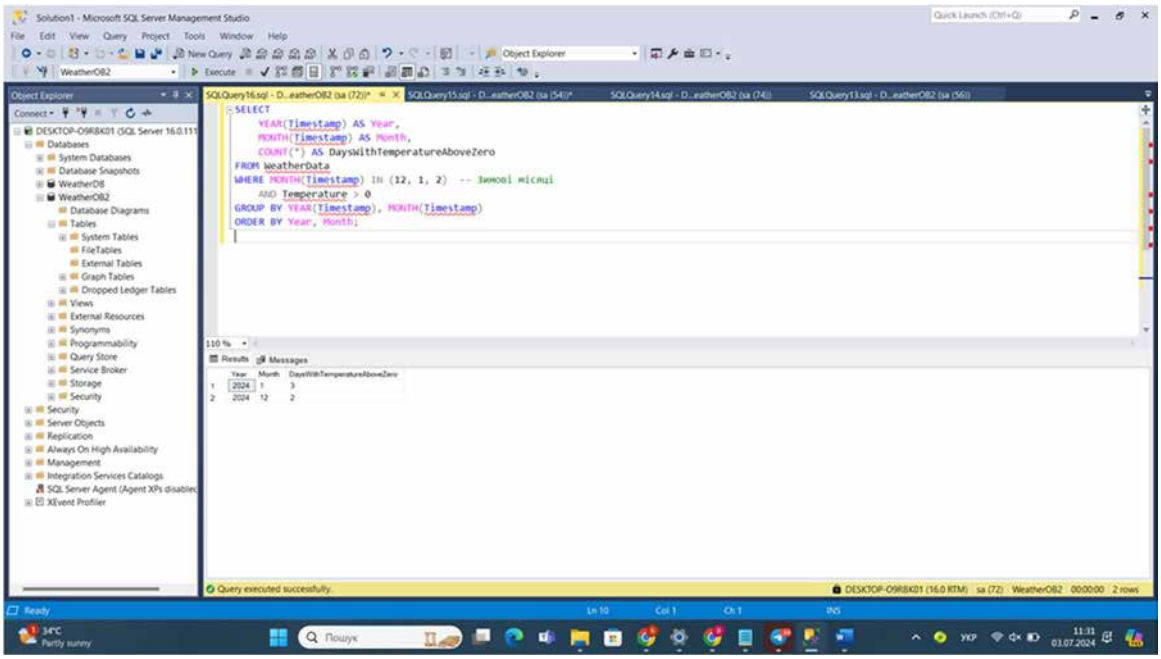

Рисунок. 3.2 – Результат запиту: дні з температурою вище 0°C у зимові місяці

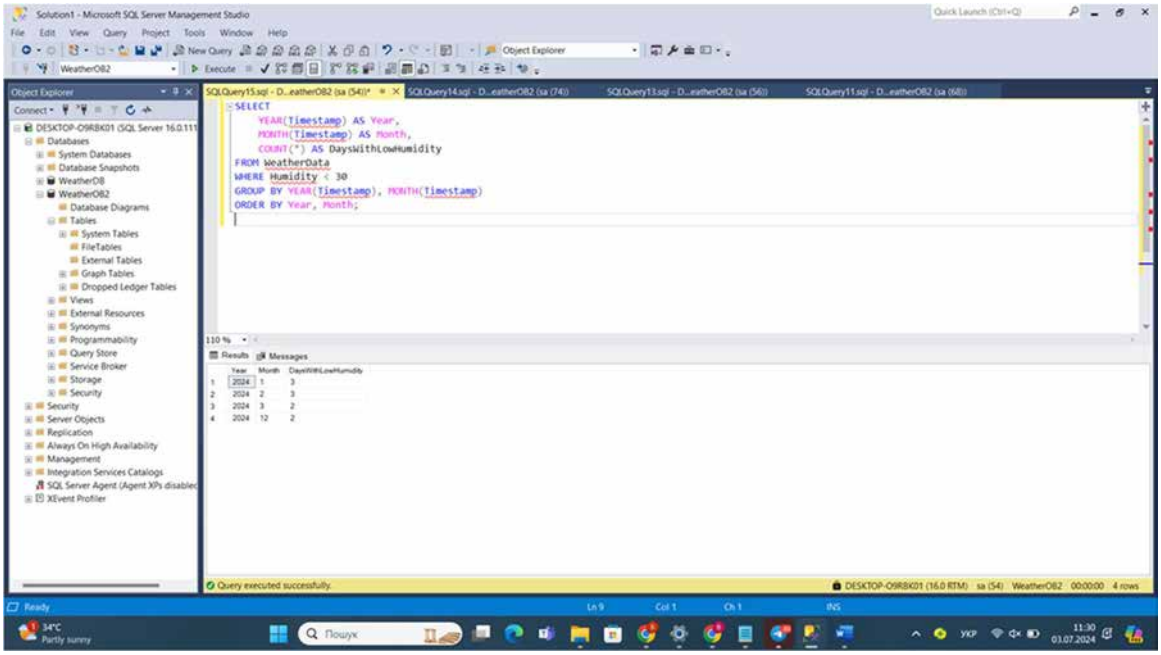

Рисунок. 3.3 – Результат запиту: дні з вологістю нижче 30%

3.3. Аналіз погодних даних за допомогою SQL

Після формування структури бази даних та реалізації базових SQL-запитів, важливим етапом стало проведення глибшого аналізу зібраних погодних даних. Основною метою цього підрозділу є демонстрація можливостей використання SQL-запитів для аналітики та візуалізації інформації, що накопичується в системі.

SQL дає можливість здійснювати як прості запити, так і складні аналітичні операції обчислення середніх значень, знаходження максимумів і мінімумів, фільтрацію, групування за місяцями, роками, сезонами тощо. У межах дипломного проєкту було сформовано кілька запитів, які дозволяють оцінити погодні тенденції та виявити закономірності.

Аналіз середньої температури по місяцях за всі роки [29]

Код програмної реалізації форми наведено в лістингу 3.4:

Лістинг 3.4

```
SELECT
YEAR(Timestamp) AS Year,
MONTH(Timestamp) AS Month,
AVG(Temperature) AS AverageTemperature
FROM WeatherData
GROUP BY YEAR(Timestamp), MONTH(Timestamp)
ORDER BY Year, Month;
```

Цей запит дозволяє отримати середню температуру по кожному місяцю в кожному році. Він є корисним для аналізу сезонних коливань температури та спостереження за змінами клімату.

Код програмної реалізації форми наведено в лістингу 3.5:

Лістинг 3.5

```
SELECT
MAX(Pressure) AS MaxPressure,
MIN(Pressure) AS MinPressure
FROM WeatherData;
```

Цей запит допомагає швидко визначити екстремальні значення атмосферного тиску, що може бути важливим для виявлення аномальних погодних явищ.

Середня вологість по контрольних точках

Код програмної реалізації форми наведено в лістингу 3.6:

Лістинг 3.6

```
SELECT
MonitoringPointID,
AVG(Humidity) AS AverageHumidity
FROM WeatherData
GROUP BY MonitoringPointID;
```

Такий запит дозволяє побачити, в яких точках спостереження найвища чи найнижча середня вологість. Це корисно для локального аналізу.

Кількість аномальних погодних подій

Код програмної реалізації форми наведено в лістингу 3.7:

Лістинг 3.7

```
SELECT
EventType,
COUNT() AS EventCount
FROM ExtremeWeatherEvent
GROUP BY EventType;
```

Цей запит дає змогу порахувати кількість різних типів аномалій: сильний вітер, буря, град, спека тощо. Такі дані важливі для оцінки загроз і трендів.

Середня швидкість вітру по сезонах

Для аналізу по сезонах потрібно групувати дані за місяцями і відповідно об'єднувати у сезони.

Код програмної реалізації форми наведено в лістингу 3.8:

Лістинг 3.8

```
SELECT
CASE
WHEN MONTH(Timestamp) IN (12,1,2) THEN 'Winter'
WHEN MONTH(Timestamp) IN (3,4,5) THEN 'Spring'
WHEN MONTH(Timestamp) IN (6,7,8) THEN 'Summer'
WHEN MONTH(Timestamp) IN (9,10,11) THEN 'Autumn'
END AS Season,
AVG(WindSpeed) AS AverageWindSpeed
FROM WeatherData
```

```
GROUP BY
CASE
WHEN MONTH(Timestamp) IN (12,1,2) THEN 'Winter'
WHEN MONTH(Timestamp) IN (3,4,5) THEN 'Spring'
WHEN MONTH(Timestamp) IN (6,7,8) THEN 'Summer'
WHEN MONTH(Timestamp) IN (9,10,11) THEN 'Autumn'
END;
```

Цей запит показує, в яку пору року в середньому фіксувалась найсильніша швидкість вітру.

Вивід останніх записів про погоду

Код програмної реалізації форми наведено в лістингу 3.9:

Лістинг 3.9

```
SELECT TOP 10
FROM WeatherData
ORDER BY Timestamp DESC;
```

Завдяки цьому запиту можна вивести найсвіжіші дані, що дозволяє перевірити актуальність та справність сенсорів.

Аналіз по днях тижня

Код програмної реалізації форми наведено в лістингу 3.10:

Лістинг 3.10

```
SELECT
DATENAME(WEEKDAY, Timestamp) AS DayOfWeek,
COUNT(*) AS EntryCount
FROM WeatherData
GROUP BY DATENAME(WEEKDAY, Timestamp);
```

Цей запит дозволяє визначити, в які дні тижня фіксується найбільше записів - це корисно для пошуку системних збоїв або періодичності роботи сенсорів.

Усі наведені запити є прикладом використання SQL для аналітики даних у реальному проєкті.

За їх допомогою можна:

оцінити погодні умови у динаміці;

порівняти ситуацію між роками або точками;

виявити критичні та аномальні події;

сформувати звіти для користувачів або керівництва;

оптимізувати інтерфейс користувача для швидкого доступу до інформації.

Аналітика на основі SQL є ефективним інструментом для розуміння структури погодних змін, особливо в умовах змін клімату та нестабільних метеоумов. [30]

3.4 . Розробка інтерфейсу користувача в Figma

Інтерфейс користувача - це візуальний прошарок між користувачем та функціональною частиною інформаційної системи. У випадку автоматизованої системи моніторингу погодних умов саме через інтерфейс здійснюється основна взаємодія з кінцевим користувачем: перегляд актуальної інформації, отримання прогнозу, перегляд попереджень та виконання пошукових дій. Саме тому важливо було не лише реалізувати функціональну базу, а й створити зручний, зрозумілий і привабливий інтерфейс.

Для створення макетів було використано онлайн-сервіс Figma, який є одним з найпопулярніших інструментів сучасного UI/UX-дизайну. Перевагами Figma є простота у використанні, можливість одночасної командної роботи, підтримка адаптивного дизайну, а також зручний інтерфейс для побудови прототипів та дизайну інтерфейсів як для десктопних, так і для мобільних пристроїв.

Розробка інтерфейсу складалась з кількох логічних етапів.

Аналіз цільової аудиторії

Перш ніж перейти до створення елементів дизайну, було визначено основні групи користувачів:

- звичайні користувачі, які хочуть дізнатись поточну погоду або прогноз;
- аграрії та фермери, які стежать за погодою для прийняття рішень;
- науковці, що аналізують зміни клімату;
- працівники служб надзвичайних ситуацій, яким потрібна оперативна інформація.

Для кожної групи важливо, щоб інтерфейс був інтуїтивно зрозумілий, інформація подавалась стисло, але інформативно, а також була можливість отримати швидкий доступ до даних без перевантаження екрана другорядною інформацією. [31]

Визначення структури інтерфейсу

Було створено логічну структуру інтерфейсу, яка передбачає такі елементи:

головна сторінка (поточна погода, попередження);

сторінка прогнозу (на 5-7 днів вперед);

аналітика (графіки температур, вологості, тиску);

вхід/реєстрація (авторизація користувача);

мобільна версія з адаптивною верстkою та спрощеною навігацією.

Для кожного розділу передбачені окремі фрейми у Figma, що дозволяє гнучко опрацьовувати компоненти, налаштовувати відступи, кольори та типографіку.

Розробка ПК-версії інтерфейсу

Головний макет для десктопної версії включає повноекранний блок з відображенням поточної погоди, блоку попередження (при наявності аномалій), а також панелі зліва, де розміщені кнопки доступу до розділів: температура, тиск, вологість, вітер тощо. Внизу виводиться прогноз на кілька днів.

Використано плавну градієнтну заливку з фоном, що нагадує погодні умови (небо, хмари, сніг або море). Шрифти обрано читабельні, з достатнім контрастом. Іконки мінімалістичні, але інформативні.

Розробка мобільної версії

Для мобільної версії дизайн було адаптовано під вертикальну орієнтацію та з урахуванням обмеженого простору екрана. Весь контент розміщено вертикально, блоки скроляться вниз. Головна інформація - поточна температура, погода, попередження - знаходиться вгорі.

Кнопки збільшено, щоб ними було зручно користуватись на сенсорному екрані. Всі елементи вирівняно для збереження зручності при різних розмірах пристроїв.

UX-рішення

Було прийнято низку важливих рішень:

кольори: основні - сині та блакитні відтінки для асоціації з погодою;

контрастні елементи: червоні попередження привертають увагу;

простота: інтерфейс не перевантажено зайвими деталями;

адаптивність: однаковий вигляд на різних розширеннях;

швидкий доступ до основних функцій.

Також особливу увагу приділено станам інтерфейсу при помилках (наприклад, відсутність даних), а також роботі при слабкому інтернет-з'єднанні передбачено показ спінерів та повідомлень про помилки. [32]

Підсумок

Використання Figma дозволило швидко створити як прототип, так і повноцінний дизайн інтерфейсу. Готові макети можуть бути реалізовані у вигляді HTML+CSS-інтерфейсу або адаптовані у мобільний застосунок.

Отже, інтерфейс системи повністю відповідає цілям:

простий і зрозумілий;

сучасний і привабливий;

адаптивний до пристрою користувача;

інформативний і функціональний.

У наступному підрозділі наведено макети з Figma для десктопної та мобільної версій, що демонструють реалізований дизайн.

3.5. Прототипування додатку для ПК та мобільної версії

Після завершення етапу проєктування інтерфейсу користувача в середовищі Figma, наступним логічним кроком стало створення прототипу програмного забезпечення. Прототип дозволяє не лише побачити візуальний вигляд майбутнього застосунку, але й оцінити логіку переходів, розташування елементів, адаптивність та функціональність. У даному підрозділі описано процес створення та аналізу прототипу для настільної (ПК) та мобільної версії додатку.

Мета прототипування

Прототипування - це процес створення інтерактивної моделі інтерфейсу до його реалізації. Основна мета полягає у перевірці структури, логіки та зручності використання інтерфейсу. Це дозволяє заздалегідь побачити можливі недоліки, протестувати зручність користування та за потреби внести зміни до реальної розробки.

Особливості прототипу для ПК

Прототип настільної версії системи створено з урахуванням повного розміру екрана користувача. Основні блоки розміщені горизонтально та

вертикально так, щоб користувач бачив актуальну інформацію без потреби прокручування. До особливостей входять:

широка центральна панель з інформацією про поточну погоду;

правий блок для навігаційних кнопок: температура, вологість, тиск тощо;

інформаційне попередження (за потреби);

блок прогнозу на декілька днів;

логотип та кнопка входу у верхній частині екрана.

Розміщення елементів обрано таким чином, щоб навіть користувач без досвіду міг швидко орієнтуватися, знайти потрібні дані та змінити параметри.

Особливості прототипу для мобільної версії

Для мобільної версії інтерфейс адаптовано під вертикальне розташування. Основний акцент зроблено на збереження функціональності при обмеженому просторі. Було змінено:

- розмір шрифтів та елементів інтерфейсу;
- меню перетворено на вертикальний список;
- прогноз реалізовано як слайдер або перелік із прокручуванням;
- іконки збільшено для зручного натискання пальцем. [33]

Було враховано рекомендації з мобільної UX-навігації: зручність скролу, розміщення важливих елементів у зоні великого пальця, спрощення навігації.

Переваги створених прототипів

Завдяки створенню прототипів стало можливим ще до етапу розробки оцінити вигляд майбутнього застосунку. Було перевірено наступне:

чи не перевантажено екран другорядною інформацією;

чи зручно переходити між розділами;

чи зберігається єдиний стиль для різних платформ;

як реагує користувач на елементи інтерфейсу (кольори, шрифти, кнопки);

чи можлива реалізація усіх елементів у реальному коді.

Відмінності між ПК та мобільною версією

Незважаючи на загальну стилістику та логіку роботи, версії для ПК і мобільних пристроїв мають різне розташування елементів. У ПК-версії акцент

зроблено на одночасне відображення кількох блоків інформації, тоді як у мобільній - на послідовну прокрутку. Також у мобільній версії меню приховане за кнопкою, що економить місце.

Логіка взаємодії

Користувач, заходячи на головну сторінку, бачить актуальну температуру, попередження та прогноз. Він може натиснути на відповідну кнопку (наприклад, "Тиск"), щоб побачити детальний графік, або перейти до архіву. Якщо активовано попередження (наприклад, температура +13°C вище норми), воно підсвічується червоним блоком із відповідною іконкою. [34]

Результати Прототипування

Прототипування виявилось ефективним способом перевірки структури та функціональності системи. Було враховано адаптивність, сценарії взаємодії, логіку навігації. Прототипи можуть бути використані розробниками для верстки фронтенду у HTML/CSS або перенесені до мобільного застосунку за допомогою таких інструментів як Flutter, React Native, Swift тощо.

На [Рис.3.4] наведено зображення двох макетів, створених у Figma для десктопної та мобільної версій.

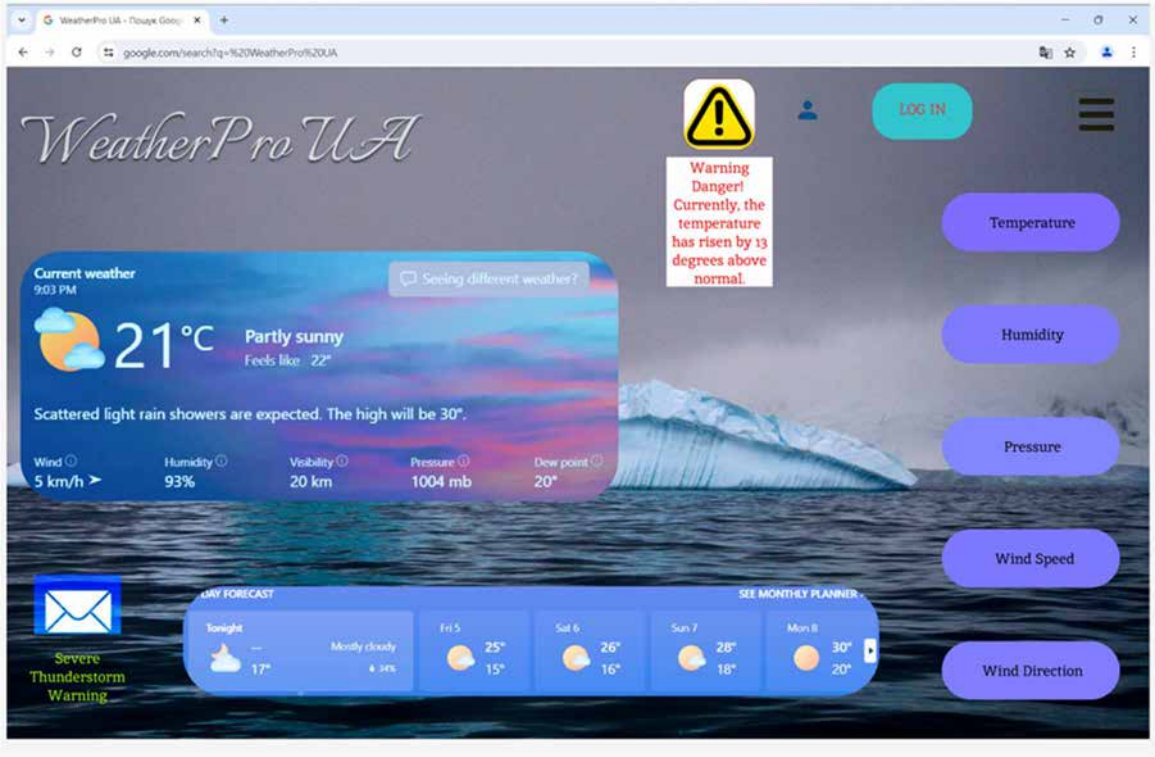

Рисунок. 3.4 – Прототип інтерфейсу для ПК, створений у Figma

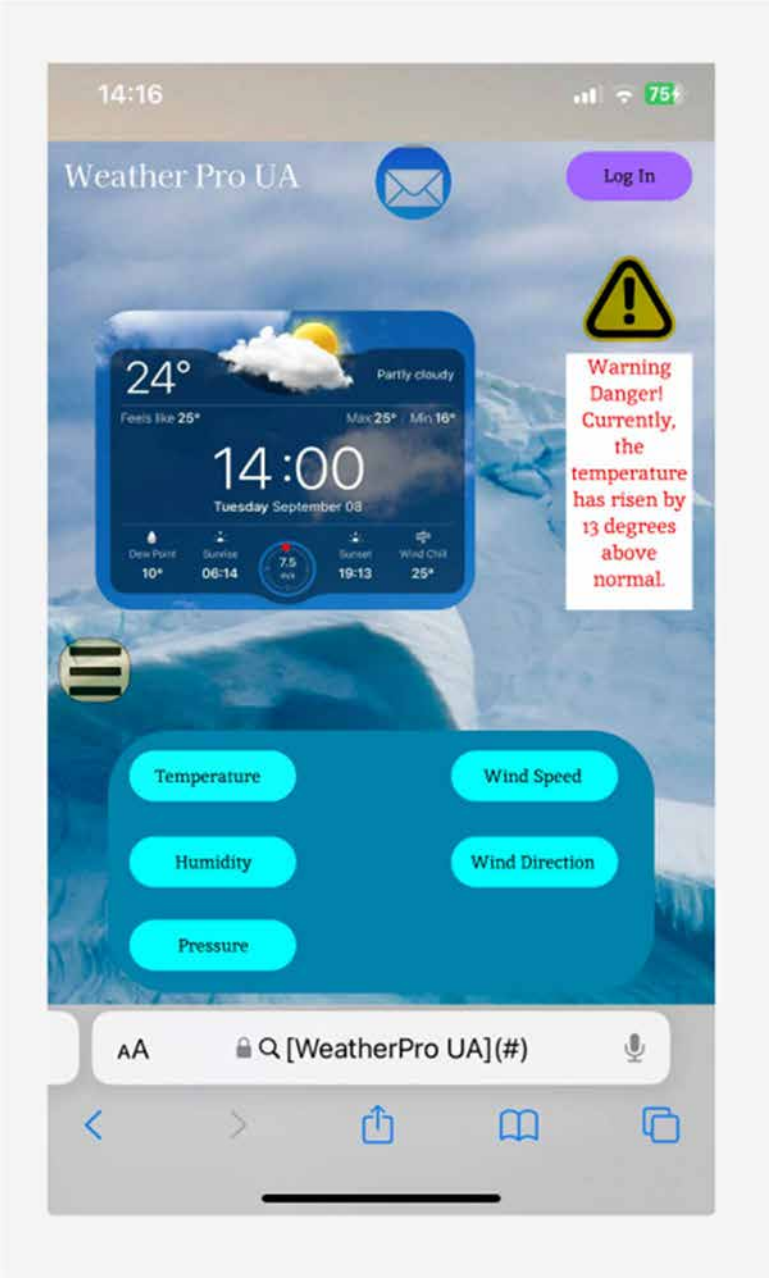

Рисунок. 3.5 – Прототип інтерфейсу для мобільної версії

На етапі реалізації було створено реляційну базу даних, розроблено SQL-запити для аналізу метеоданих, а також побудовано прототип інтерфейсу користувача у Figma. Це забезпечило практичну демонстрацію працездатності системи та підтвердило її функціональність і потенціал для подальшого впровадження.

ВИСНОВКИ

У процесі виконання бакалаврської кваліфікаційної роботи на тему "Автоматизована система моніторингу погоди" було розроблено комплексне рішення, що охоплює всі ключові етапи життєвого циклу інформаційної системи - від аналізу предметної області до створення прототипу користувацького інтерфейсу.

Проведене дослідження підтвердило актуальність теми, оскільки сучасні кліматичні зміни, потреба в оперативному прийнятті рішень у сільському господарстві, енергетиці, транспорті та надзвичайних службах вимагають наявності систем, здатних у режимі реального часу відстежувати, аналізувати та передавати інформацію про погодні умови. Наявні програмні рішення мають обмеження щодо відкритості, гнучкості та адаптації до локальних потреб, тому створення власної автоматизованої системи моніторингу є доцільним і корисним.

На першому етапі було проведено ґрунтовний аналіз предметної області. Вивчено принципи моніторингу погодних умов, особливості існуючих метеосервісів, їх архітектуру, функціональність, алгоритми збирання та обробки даних. Проаналізовано популярні рішення, такі як OpenWeather, AccuWeather, Gismeteo тощо. Визначено їхні сильні та слабкі сторони. На основі цього аналізу були сформульовані вимоги до власної системи: підтримка декількох контрольних точок, зберігання історичних даних, сповіщення про аномальні показники, зручний користувацький інтерфейс.

У другому розділі здійснено проєктування інформаційної системи. Побудовано діаграми, які дозволяють візуалізувати структуру системи та взаємодію її компонентів. Серед них: діаграма прецедентів (Use Case), яка відображає ролі та дії користувачів; BPMN-діаграма бізнес-процесів; DFD-діаграма потоків даних; діаграма класів, що моделює об'єкти системи; діаграма послідовностей; ER-модель для побудови бази даних.

Ці моделі дозволили краще структурувати майбутню реалізацію, уникнути логічних помилок, а також спростити комунікацію між розробником і користувачем. Особливу увагу приділено цілісності даних, масштабованості системи та потенційному розширенню функціоналу.

У третьому розділі реалізовано основні технічні аспекти системи. Було створено базу даних у середовищі SQL Server. Структура включає таблиці користувачів, погодних точок, погодних показників, подій, запитів, сховищ, баз даних, процесів обробки. Застосовано зовнішні ключі для підтримки зв'язків між таблицями. Реалізовані SQL-запити для вибірки актуальної інформації, аналізу історичних змін, виявлення аномалій, побудови звітів.

Було здійснено базову перевірку аналітичних запитів, зокрема: середня температура по місяцях, мінімальні та максимальні значення тиску, середня вологість по точках, кількість днів з низькою вологістю, частота аномальних подій. Результати аналізу дають змогу робити практичні висновки щодо погодних тенденцій, і можуть бути використані в агросекторі, управлінні ризиками тощо.

Особливу увагу приділено розробці інтерфейсу користувача. У середовищі Figma створено повноцінний прототип інтерфейсу для ПК та мобільної версії. Було визначено цільову аудиторію, основні сценарії використання, побудовано фрейми для головного вікна, детальної інформації, авторизації, прогнозу, архіву. Враховано принципи UX-дизайну: простота, доступність, логіка, адаптивність. Інтерфейс відповідає сучасним вимогам до зручності використання та візуальної привабливості.

Прототип продемонстрував функціональність системи ще до її реалізації у коді. Це дозволяє на ранньому етапі перевірити зручність навігації, логіку взаємодії з елементами, коректність розташування. У разі продовження розробки система може бути реалізована з використанням фреймворків (React, Angular), а мобільна версія - у Flutter або React Native.

Практична значущість дипломної роботи полягає в тому, що створена система може бути використана як основа для впровадження у невеликих господарствах, навчальних закладах, наукових установах. Вона є гнучкою, адаптивною, легко масштабується. Зібрані дані можуть використовуватись для аналізу кліматичних змін, планування сільськогосподарських робіт, оцінки ризиків.

Підсумовуючи вищесказане, можна зробити наступні висновки:

поставлену мету роботи досягнуто повністю;

виконано глибокий аналіз предметної області;

розроблено архітектуру інформаційної системи;

реалізовано базу даних і SQL-запити для аналізу даних;

створено зручний інтерфейс у Figma;

побудовано прототип, що відповідає вимогам користувача.

Отже, дипломна робота реалізує повний цикл створення інформаційної системи від ідеї до прототипу. Її результати можуть бути застосовані на практиці, а сама система вдосконалена в майбутньому, наприклад, шляхом підключення реальних сенсорів, побудови API для обміну даними або розгортання веб-версії в хмарі.

Таким чином, робота не лише розкрила теоретичні знання, але й дала змогу реалізувати практичні навички, отримані в ході навчання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. OpenWeather. https://openweathermap.org/

2. World Meteorological Organization. https://public.wmo.int/

3. Мельник А.І. Інформаційні технології в екологічному моніторингу. К.: Либідь, 2020.

4. Gismeteo. https://www.gismeteo.ua/

5. AccuWeather. https://www.accuweather.com/

6. ІТ в прогнозуванні погоди: сучасні сервіси // Науково-технічний вісник, №4, 2021.

7. Sommerville I. Software Engineering. 10th ed. — Pearson, 2016.

8. Шевчук Ю.М. Проєктування інформаційних систем. — Львів: Вид-во ЛНУ, 2021.

9. Методичні вказівки до виконання кваліфікаційної роботи НУБіП України, 2024.

10. Приклади дипломних проєктів з інформаційних технологій // Repository.nubip.edu.ua

11. Архітектура програмних систем: навч. посіб. / П.І. Когут. — К.: КНЕУ, 2020.

12. Design patterns and architectures in modern software / IEEE Software Journal, 2022.

13. UML Use Case Diagrams — Tutorial. https://www.visual-paradigm.com/

14. Бочаров А. Unified Modeling Language: навч. посіб. — Харків: ХНУРЕ, 2021.

15. Object Management Group. BPMN Specification. https://www.omg.org/spec/BPMN/

16. Савінов С.О. Бізнес-моделювання в ІТ. — К.: КНЕУ, 2021.

17. Yourdon E., DeMarco T. Structured Analysis and System Specification. — Prentice Hall, 1999.

18. Основи системного аналізу / І.І. Мельничук. — Львів: ЛНУ, 2020.

19. UML Class Diagram Reference. https://www.uml-diagrams.org/class-diagrams.html

20. Моделювання програмних систем / О.Ю. Іщенко. — К.: КПІ, 2022.

21. UML Sequence Diagrams Guide. https://www.visual-paradigm.com/

22. Глінський Я.М. Проєктування складних ІТ-систем. — Тернопіль, 2021.

23. Хорстманн К. Основи баз даних: ER-моделювання. — К.: ВНТУ, 2019.

24. Database Systems. Connolly & Begg. 6th ed. — Pearson, 2015.

25. Microsoft SQL Server Documentation. https://docs.microsoft.com/sql/

26. Робота з базами даних: SQL Server, Oracle, MySQL / Гаврилюк Л. — К.: НАУ, 2021.

27. Столяров Д. Мова SQL. Практичний посібник. — М.: Пітер, 2020.

28. SQL для початківців. https://sqlzoo.net/

29. Аналітичні запити в SQL / Підручник НТУУ «КПІ», 2021.

30. Big Data Weather Analytics: Springer Series, 2020.

31. Figma Documentation. https://help.figma.com/

32. Макетування в інтерфейсному дизайні / Назаренко О.В. — К.: НТУУ КПІ, 2021.

33. Прототипи додатків: методика та інструменти / Федорчук І.В. — К.: Університет, 2022.

34. UI/UX прототипування з Figma та Adobe XD. https://uxdesign.cc/

ДОДАТКИ

Додаток А

Діаграма прецедентів (Use Case Diagram)

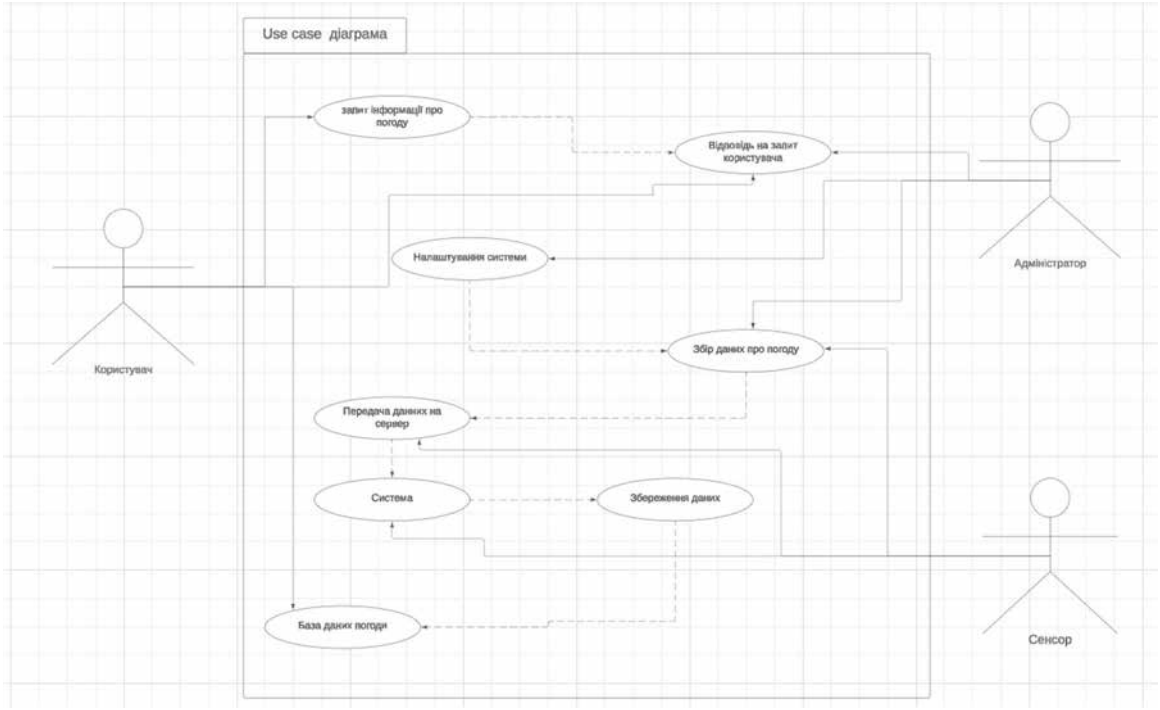

Рисунок. А.1 – Діаграма прецедентів системи моніторингу погоди

Додаток Б

BPMN-діаграма бізнес-процесу

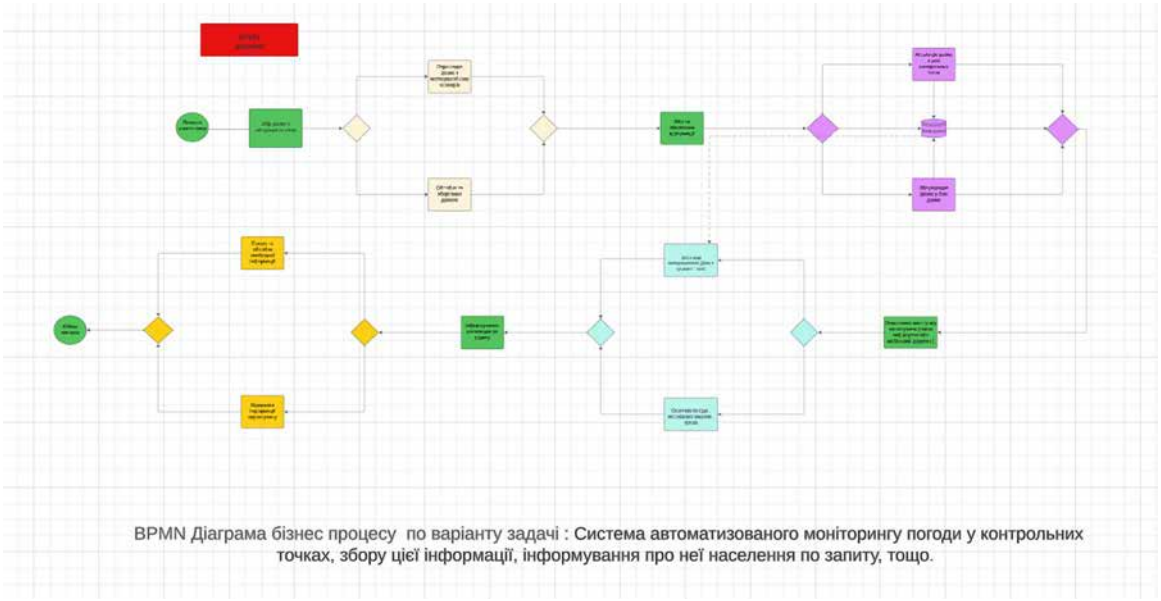

Рисунок. Б.1 – BPMN-модель збору та обробки погодних даних

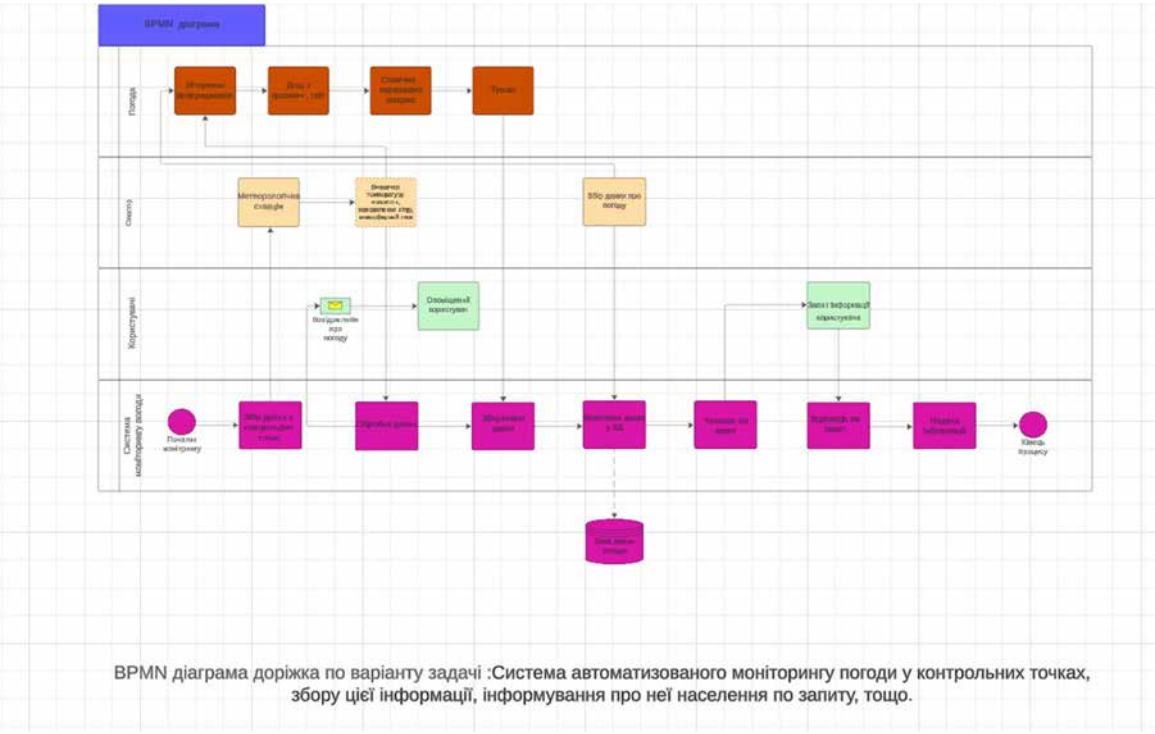

Рисунок. Б.2 – BPMN-модель збору та обробки погодних даних

Додаток В

DFD-діаграма

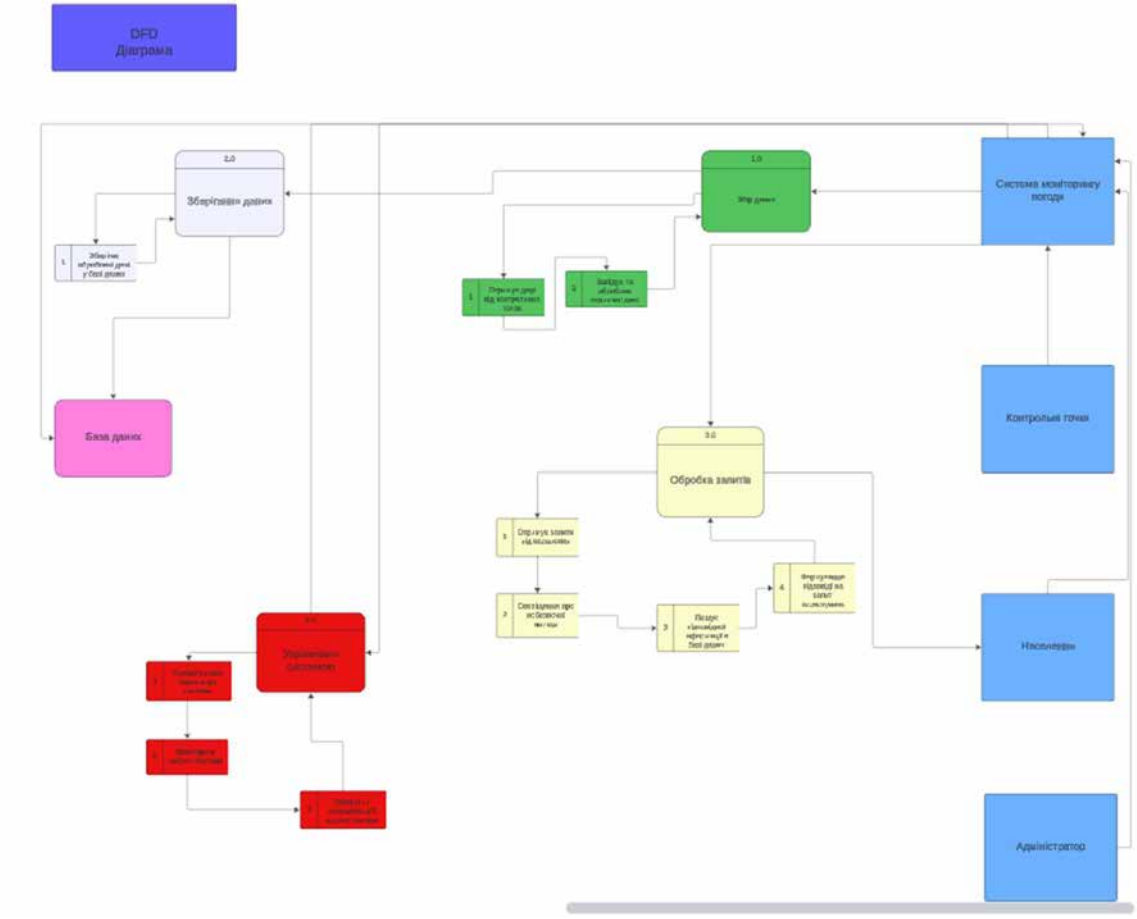

Рисунок. В.1 – Контекстна DFD-діаграма

Додаток Г

Діаграма класів

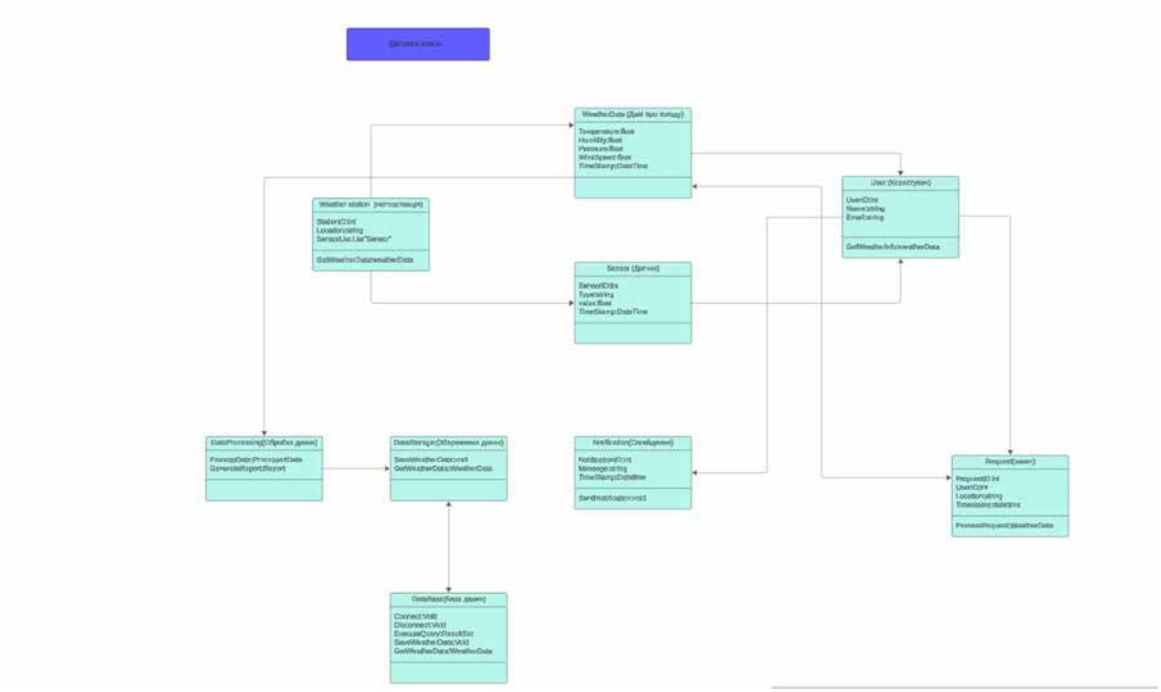

Рисунок. Г.1 – UML-діаграма класів системи

Додаток Д

Діаграма послідовностей

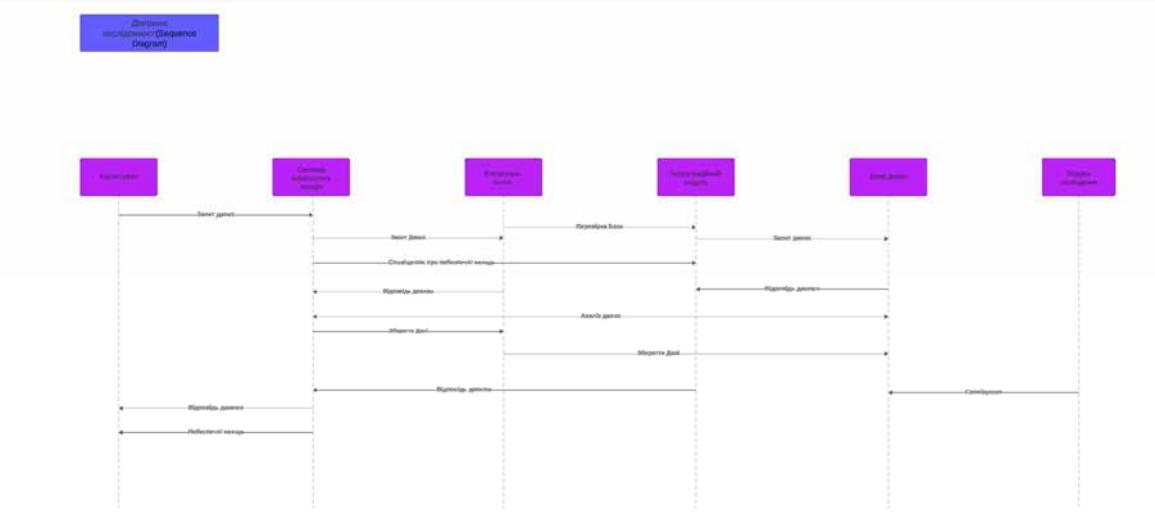

Рисунок. Д - Діаграма послідовностей системи

Додаток Е

ER-модель бази даних

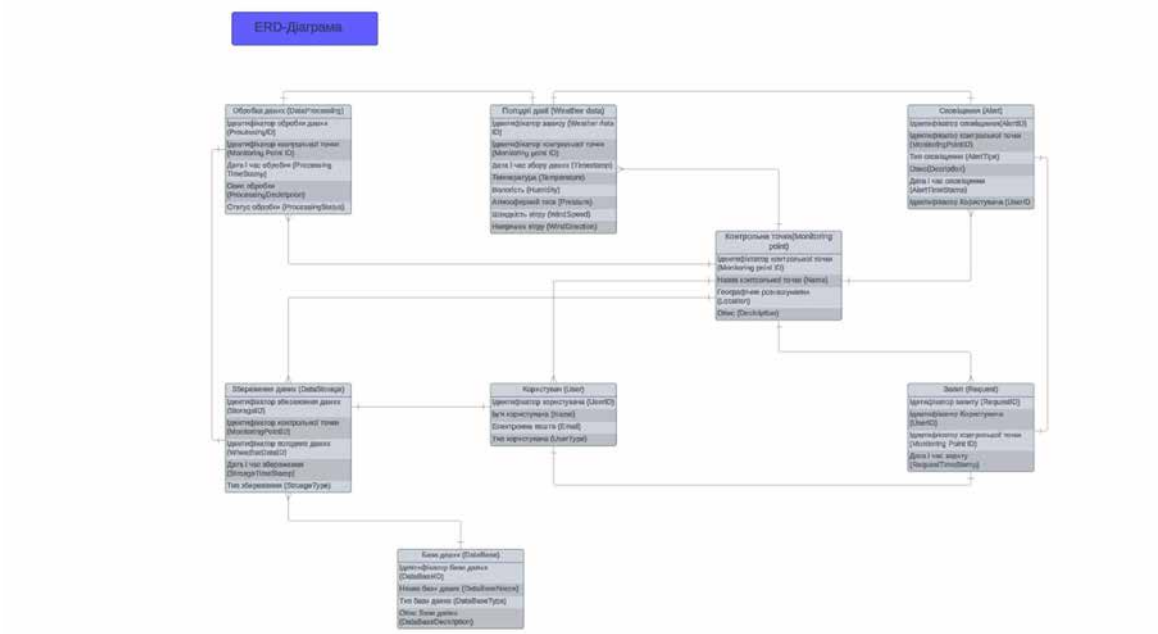

Рисунок. Е - ER-модель бази даних

Додаток Є

Діаграма потоків даних (DFD)

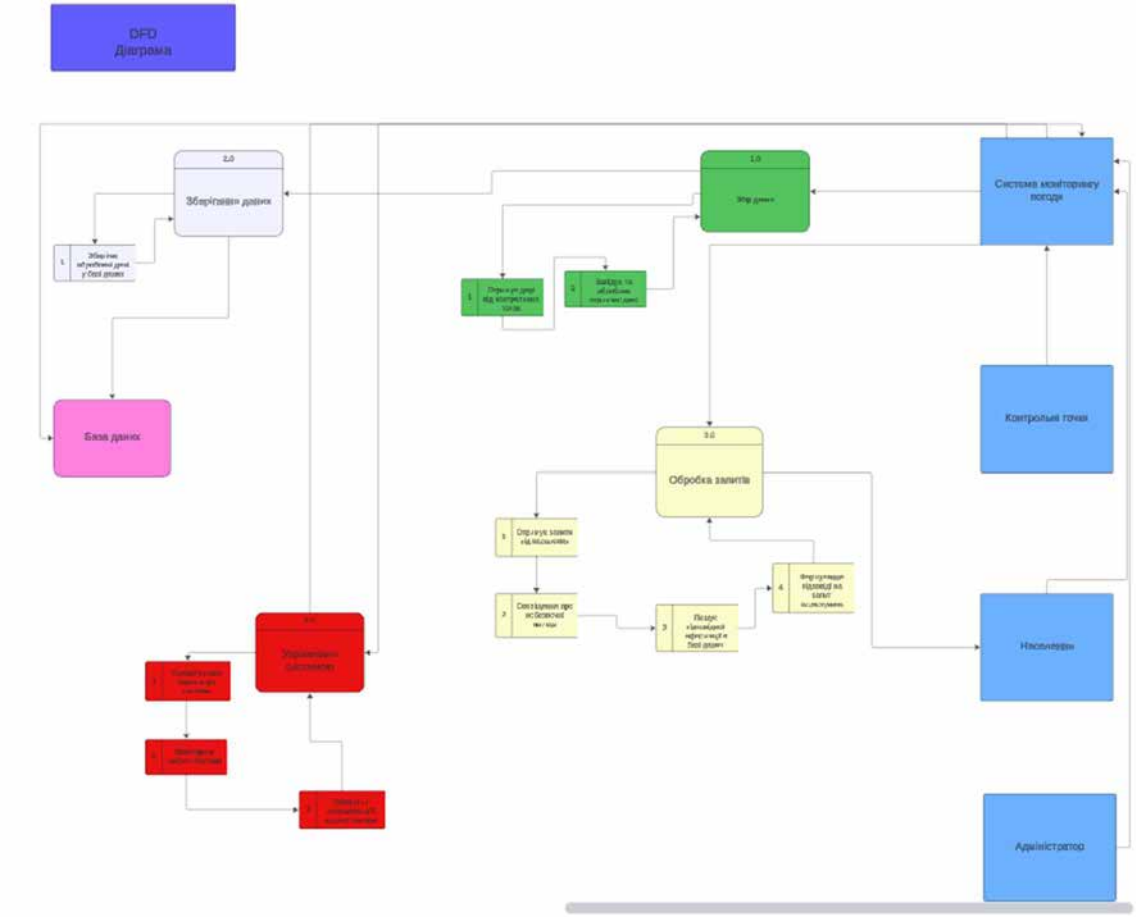

Рисунок. Є - **Діаграма потоків даних (DFD)**

Додаток Ж

SQL-код створення бази даних

Лістинг 3.1:

```
DROP TABLE IF EXISTS ExtremeWeatherEvent;
DROP TABLE IF EXISTS Alert;
DROP TABLE IF EXISTS DataStorage;
DROP TABLE IF EXISTS DataProcessing;
DROP TABLE IF EXISTS WeatherData;
DROP TABLE IF EXISTS Request;
DROP TABLE IF EXISTS MonitoringPoint;
DROP TABLE IF EXISTS [User];
DROP TABLE IF EXISTS [DataBase];

CREATE TABLE [User] (
   UserID INT PRIMARY KEY,
   Name VARCHAR(255),
   Email VARCHAR(255),
   UserType VARCHAR(50)
);

CREATE TABLE MonitoringPoint (
   MonitoringPointID INT PRIMARY KEY,
   Name VARCHAR(255),
   Location VARCHAR(255),
   Description TEXT
);

CREATE TABLE [DataBase] (
   DataBaseID INT PRIMARY KEY,
   DataBaseName VARCHAR(255),
```

```
    DataBaseType VARCHAR(50),
    DataBaseDescription TEXT
);

CREATE TABLE WeatherData (
    WeatherDataID INT PRIMARY KEY,
    MonitoringPointID INT,
    Timestamp DATETIME,
    Temperature FLOAT,
    Humidity FLOAT,
    Pressure FLOAT,
    WindSpeed FLOAT,
    WindDirection VARCHAR(50),
    FOREIGN KEY (MonitoringPointID) REFERENCES
MonitoringPoint(MonitoringPointID)
);

CREATE TABLE DataProcessing (
    ProcessingID INT PRIMARY KEY,
    MonitoringPointID INT,
    ProcessingTimestamp DATETIME,
    ProcessingDescription TEXT,
    ProcessingStatus VARCHAR(50),
    FOREIGN KEY (MonitoringPointID) REFERENCES
MonitoringPoint(MonitoringPointID)
);

CREATE TABLE Alert (
    AlertID INT PRIMARY KEY,
    MonitoringPointID INT,
```

```
    AlertType VARCHAR(50),
    Description TEXT,
    AlertTimestamp DATETIME,
    UserID INT,
    FOREIGN KEY (MonitoringPointID) REFERENCES MonitoringPoint(MonitoringPointID),
    FOREIGN KEY (UserID) REFERENCES [User](UserID)
);

CREATE TABLE DataStorage (
    StorageID INT PRIMARY KEY,
    MonitoringPointID INT,
    WeatherDataID INT,
    StorageTimestamp DATETIME,
    StorageType VARCHAR(50),
    DataBaseID INT,  -- Додаємо зовнішній ключ
    FOREIGN KEY (MonitoringPointID) REFERENCES MonitoringPoint(MonitoringPointID),
    FOREIGN KEY (WeatherDataID) REFERENCES WeatherData(WeatherDataID),
    FOREIGN KEY (DataBaseID) REFERENCES [DataBase](DataBaseID) -- Визначаємо зовнішній ключ
);

CREATE TABLE Request (
    RequestID INT PRIMARY KEY,
    UserID INT,
    MonitoringPointID INT,
    RequestTimestamp DATETIME,
    FOREIGN KEY (UserID) REFERENCES [User](UserID),
```

```
    FOREIGN KEY (MonitoringPointID) REFERENCES MonitoringPoint(MonitoringPointID)
    );

    CREATE TABLE ExtremeWeatherEvent (
        EventID INT PRIMARY KEY,
        WeatherDataID INT,
        EventType VARCHAR(50),
        EventDescription TEXT,
        EventTimestamp DATETIME,
        SeverityLevel VARCHAR(50),
        FOREIGN KEY (WeatherDataID) REFERENCES WeatherData(WeatherDataID)
    );
```

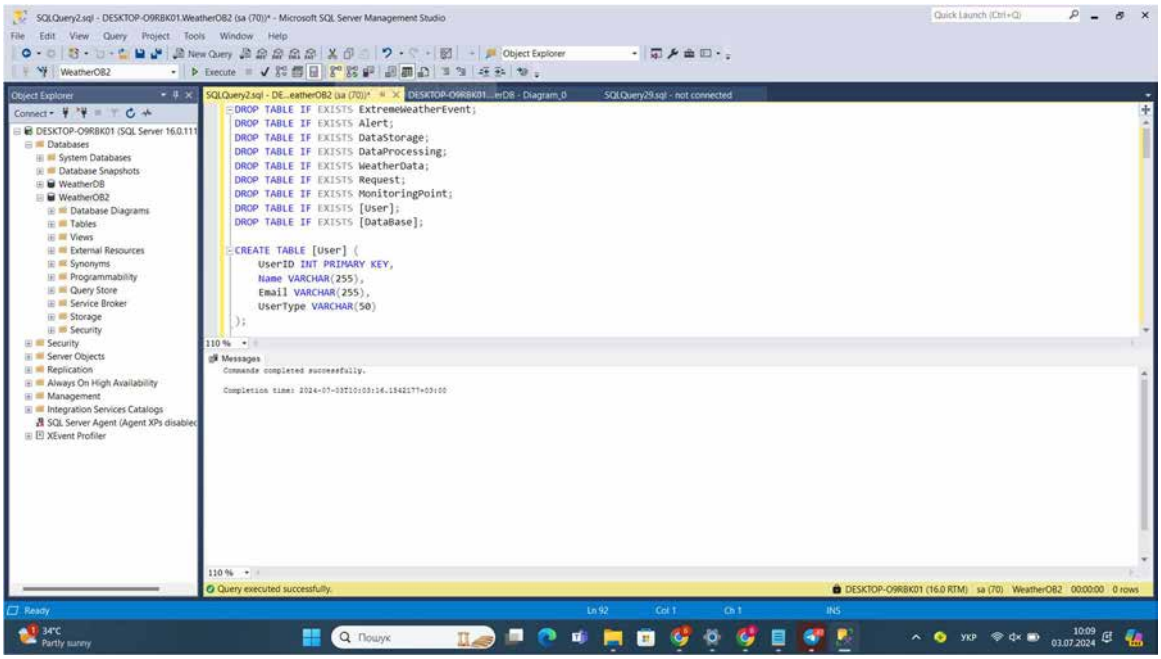

Рисунок. Ж.1 – Схема створеної бази даних у середовищі SQL Server

Додаток З

Прототип інтерфейсу в Figma

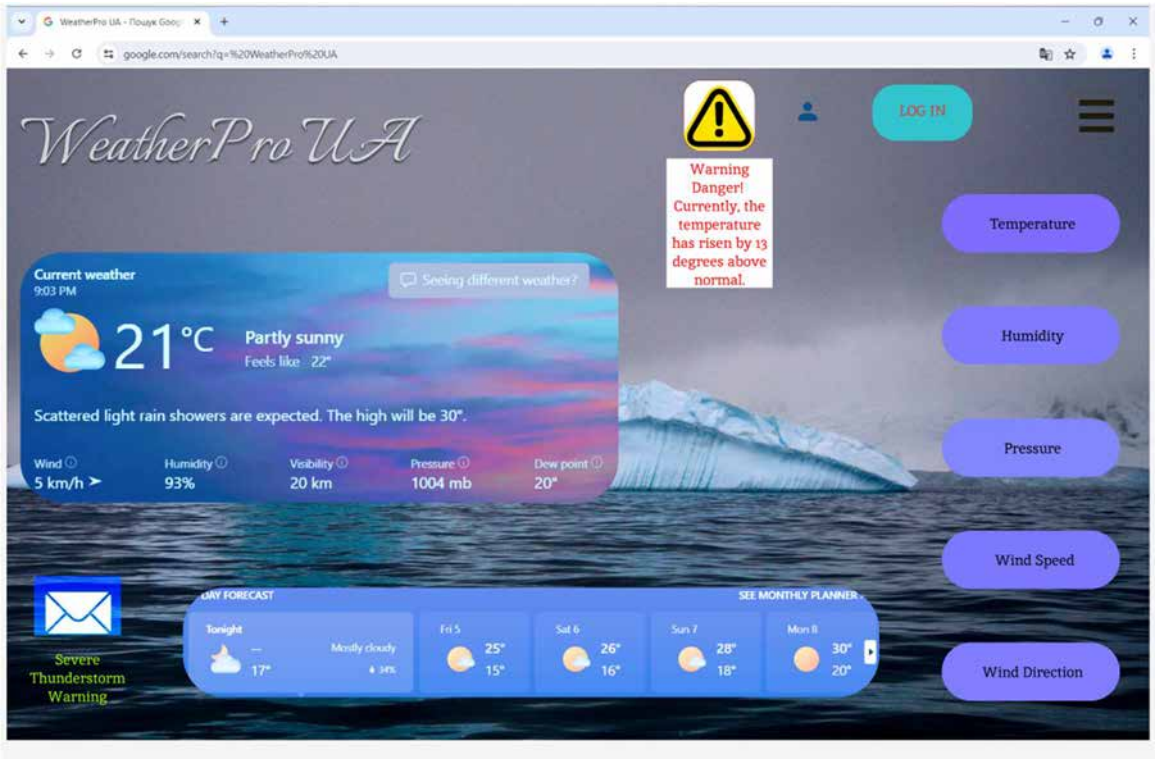

Рисунок. 3.1 – Прототип інтерфейсу для ПК

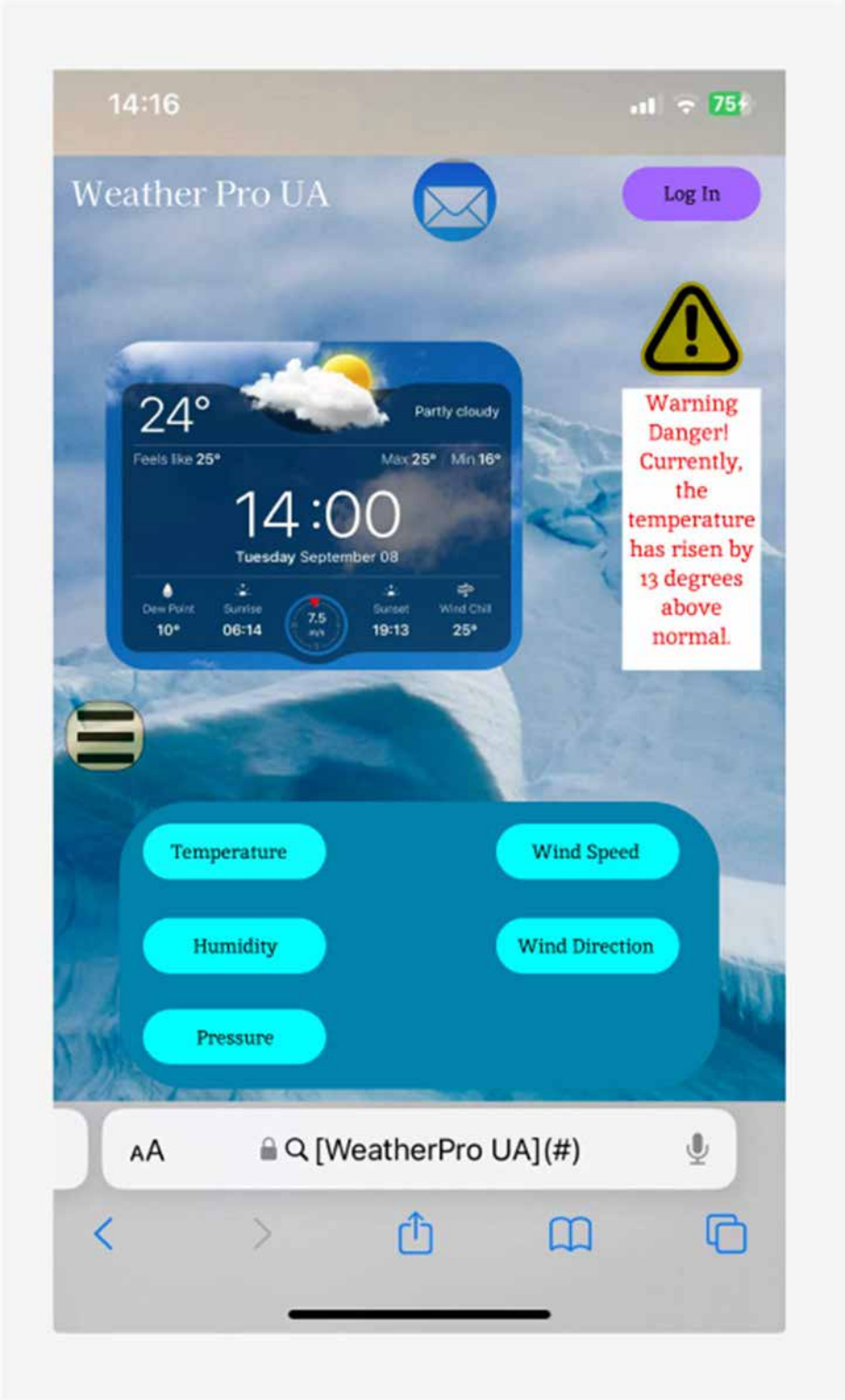

Рис. 3.2 – Прототип інтерфейсу для мобільної версії