МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій

«До захисту допущено»
В.о. завідувача кафедри

_____ Світлана ВАЩЕНКО

____ _____ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 «Комп'ютерні науки»,

освітньо-професійної програми «Інформаційні технології проектування»

на тему: «Мобільний додаток відстеження погодних показників»

Здобувача (ки) групи ІТ-91 Кущ Богдан Сергійович
(шифр групи) (прізвище, ім'я, по батькові)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Богдан КУЩ
(підпис) (Ім'я та ПРІЗВИЩЕ здобувача)

Керівник доц. кафедри ІТ, к. т. н, Віктор НЕНЯ _____
(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)

Суми – 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ
В. о. зав. кафедри ІТ

_____Світлана ВАЩЕНКО
«__» _____ 2023 р.

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

Кущ Богдану Сергійовичу

1 Тема роботи *Мобільний додаток відстеження погодних показників*

керівник роботи *Неня Віктор Григорович, к.т.н., доцент*,

затверджені наказом по університету від « 29 » травня 2023 р. №0588-VI

2 Строк подання студентом роботи « 7 » червня 2023 р.

3 Вхідні дані до роботи Технічне завдання на розробку мобільного додатку відстеження погодних показників

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Аналіз предметної області, моделювання та проектування мобільного додатку, розробка мобільного додатку

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____

6. Консультанти розділів роботи:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7.Дата видачі завдання 08.02.2023

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Постановка задачі проекту	16.02.23-25.02.23	
2	Проектування структури додатка	26.02.23-09.03.23	
3	Розробка дизайну додатка	10.03.23-17.03.23	
4	Розробка бази даних	18.03.23-03.04.23	
5	Розробка функціональності	04.04.23-24.04.23	
6	Оформлення документації	25.04.23-03.05.23	
7	Захист	15.06.23-19.06.23	

Студент _____ Богдан КУЩ
(підпис)

Керівник роботи _____ к.т.н., доц. Віктор НЕНЯ
(підпис)

РЕФЕРАТ

Тема кваліфікаційної роботи бакалавра «Мобільний додаток відстеження погодних показників».

Пояснювальна записка складається зі вступу, розділу «Аналіз предметної області», розділу «Моделювання та проектування мобільного додатку відстеження погодних показників», розділу «Розробка мобільного додатку», висновків, списку використаних джерел із 20 найменувань, додатків. Загальний обсяг роботи − 94 сторінки у тому числі 49 сторінок основного тексту, 3 сторінки списку використаних джерел, 42 сторінки додатків.

Кваліфікаційну роботу бакалавра присвячено розробці мобільного додатка відстеження погодних показників. В роботі над проектом, проведено аналіз предметної області та виявлено недоліки, що існують в додатках-аналогах. На основі проведеного аналізу, було обрано засоби реалізації, що відповідають потребам проекту. Виконано моделювання та проектування основних процесів роботи додатку, застосовуючи нотацію IDEF0.

Останній етап роботи над проектом присвячений розробці додатку. З використанням обраних засобів реалізації було розроблено два додатка, відповідно для звичайних користувачів, який дозволяє перегляд погодної інформації та додаток для оновлення погодних даних, адміністраторами.

Процес програмної реалізації включав створення макетів екрану для взаємодії з користувачами та реалізацію передачі погодних даних між додатками, з використанням системи управління базами даних Firebase.

Ключові слова: мобільний додаток, відстеження погодних показників, Android Studio, kotlin.

ЗМІСТ

ВСТУП

Протягом останнього десятиліття спостерігається помітний ріст глобальної цифрової трансформації. Продажі смартфонів стрімко зростають, а компанії-виробники активно інвестують у мобільні технології та їх розвиток. Цей розквіт мобільних пристроїв відкриває нові можливості для розробників програмного забезпечення, зокрема в сфері мобільних додатків [1].

Одним з найактуальніших напрямків у мобільній розробці є створення додатків, що відстежують різноманітні показники, що цікавлять користувачів. Один з таких показників, який має значення для багатьох людей, - це погода. Інформація про погоду не лише допомагає нам планувати ділові та особисті справи, але також впливає на настрій, здоров'я та безпеку [2].

У зв'язку зі скасуванням обмежень, пов'язаних з пандемією COVID-19, люди знову активно виходять на вулицю, планують подорожі та займаються різноманітними заняттями на природі [3]. Отже, зростає потреба в зручному і надійному способі отримання точних та актуальних погодних даних. У цьому контексті створення мобільного додатка для відстеження погодних показників стає невід'ємною складовою, яка може задовольнити потреби користувачів.

В цій області багато додатків стикаються з типовими недоліками, особливо щодо їхнього інтерфейсу. Хоча вони можуть мати значну кількість функцій, взаємодія з користувачем часто виявляється незрозумілою та складною у використанні. Це може відлякувати потенційних користувачів та обмежувати ефективність таких додатків.

З метою вирішення цих проблем та задоволення потреб користувачів у надійних погодних даних, пропонується створити новий мобільний додаток, який поєднуватиме зручність, точність та простоту використання. Цей додаток буде

базуватись на найсучасніших технологіях збору погодних даних і матиме інтуїтивно зрозумілий інтерфейс, що дозволить користувачам швидко та легко отримувати актуальну інформацію про погоду.

Щоб створити додаток для відстеження погоди, необхідно вирішити наступні задачі :

- Дослідження предметної області;
- Аналіз існуючих аналогів;
- Формування технічного завдання;
- Вибір необхідних технологій розробки;
- Проектування мобільного додатка;
- Програмна реалізація.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1Огляд останніх досліджень і публікацій

Для розробки мобільних додатків під платформу Android існує декілька типів архітектури, кожен з яких має свої переваги і недоліки.

Один з найпоширеніших – Model-View-Controller (MVC). У MVC архітектурі додаток розділяється на три компоненти: модель, представлення та контролер. Модель відповідає за бізнес-логіку та обробку даних, представлення відображає графічний інтерфейс користувача, а контролер забезпечує взаємодію між моделлю та представленням. MVC має переваги в легкості тестування, швидкості розробки та чіткому розділенні обов'язків. Проте, у складних додатках можуть виникнути проблеми з підтримкою та розширенням функціоналу [4].

Ще один популярний підхід – Model-View-Presenter різних підходів до (MVP). У MVP архітектурі модель та представлення виконують ті самі завдання, що й у MVC, але контролер змінений представник. Він відповідає за обробку подій від користувача та взаємодію з моделлю та представленням. MVP спрощує тестування та забезпечує розділення обов'язків між компонентами, але може призвести до збільшення кількості класів та складності розробки [5].

Останнім часом широко використовуваною архітектурою є Model-View-ViewModel (MVVM). У цій архітектурі модель відповідає за бізнес-логіку, представлення відображає графічний інтерфейс, а ViewModel забезпечує зв'язок між моделлю та представленням. ViewModel керує станом додатку та забезпечує потік даних між моделлю та представленням. MVVM дозволяє декларативну розробку, спрощує тестування та полегшує розширення функціоналу додатку. Проте, використання MVVM може призвести до збільшення складності коду та навантаження на пам'ять [6].

Крім вищезазначених підходів, існує також Clean Architecture, яка базується на принципах залежності і чистоти коду. Clean Architecture розділяє додаток на шари, такі як презентаційний, доменний та джерела даних, так система забезпечує слабку залежність між ними. Це дозволяє досягти високої модульності, тестованості та розширюваності додатку. Але застосування Clean Architecture може спричинити підвищення рівня складності у процесі розробки та збільшення витрат часу [7].

Кожна з наведених архітектур має свої переваги та недоліки. Вибір кращої залежить від специфіки проекту та вимог розробки. Правильний вибір архітектури дозволить створити легкий для тестування та зручний у підтримці мобільний додаток під операційну систему Android.

Вирішено обрати архітектуру MVVM для проекту, адже це сприятиме полегшенню розширення функціоналу додатку в подальшому.

1.2 Аналіз програмних продуктів – аналогів

На сьогоднішній день існує багато мобільних додатків для моніторингу погоди, але більшість з них має недоліки пов’язані з нагромадженням інтерфейсу, що може викликати складності з взаємодією.

Розглянемо перший аналог – «Weather Underground»[8]. Додаток має великий функціонал: можна дізнатися поточну температуру повітря, швидкість вітру, ймовірність опадів, індекс якості повітря та час світанку. Дизайн додатку характеризується зручністю та легкістю сприйняття, що досягається шляхом організації інформації у відокремлених блоках. Головний екран аналогу представлений на рисунках 1.1-1.2.

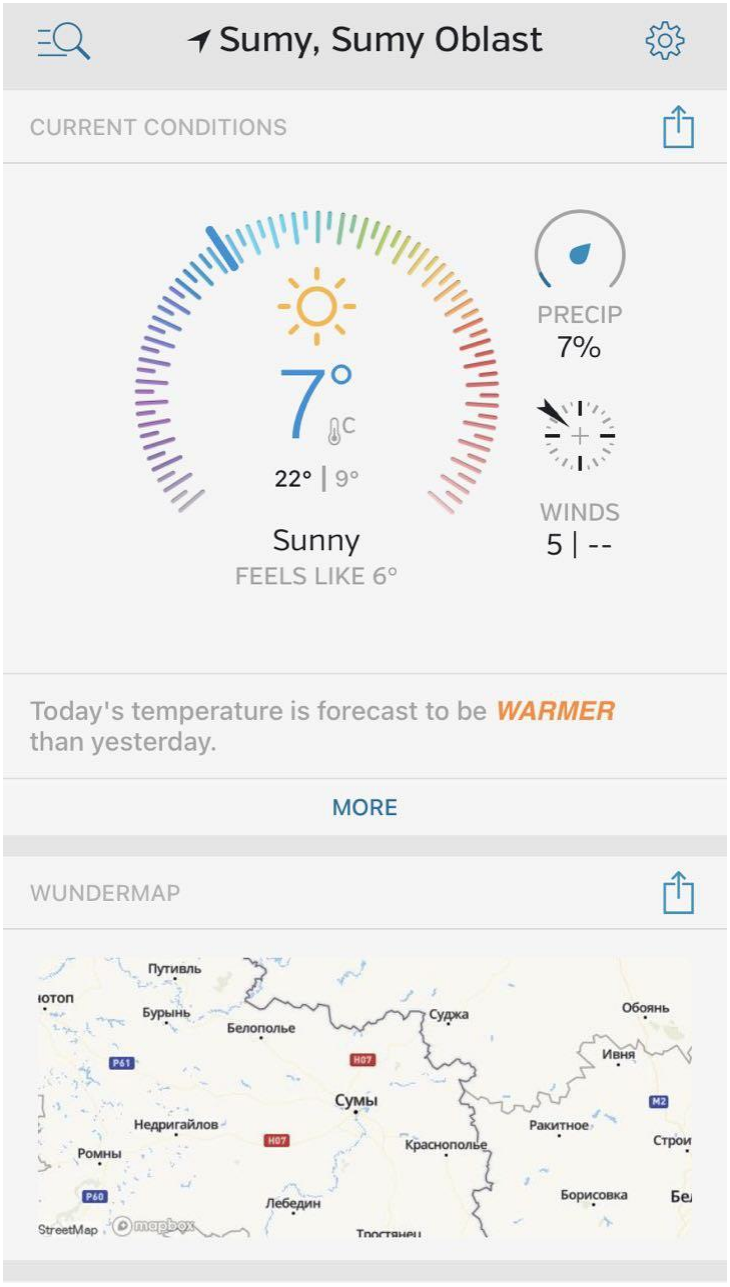

Рисунок 1.1 – Головний екран додатку «Weather Underground» частина 1

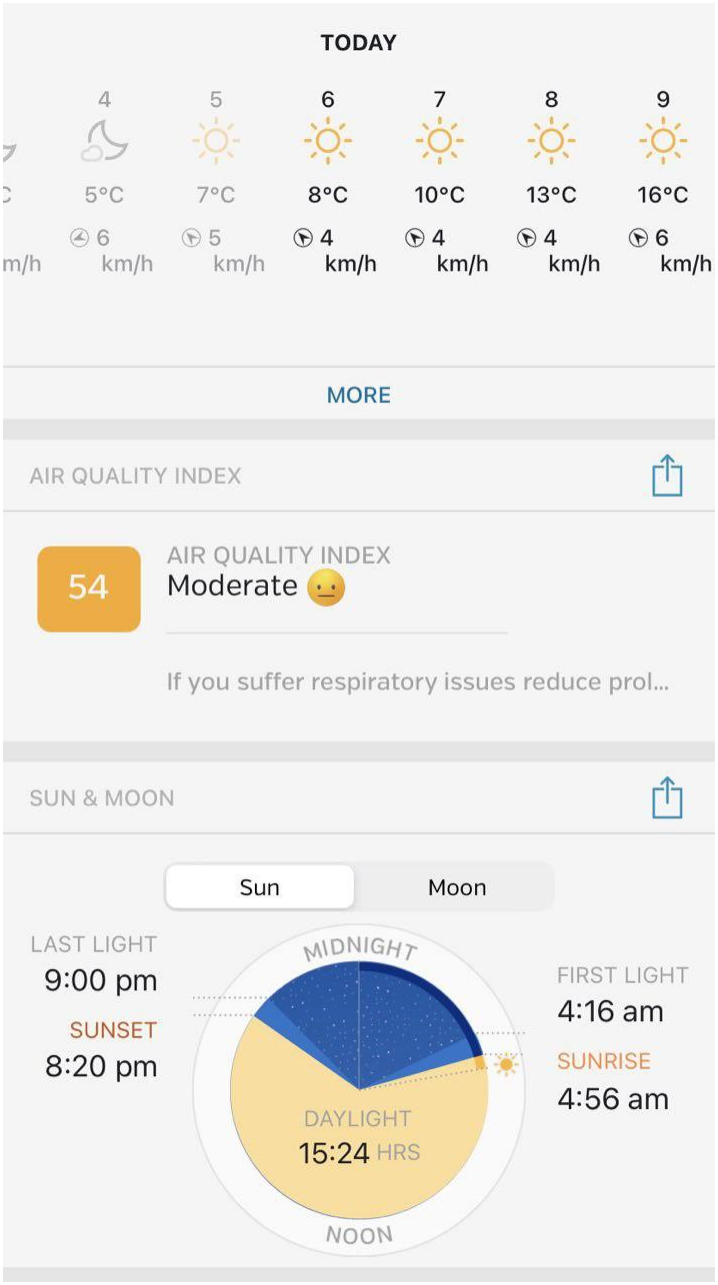

Рисунок 1.2 – Головний екран додатку «Weather Underground» частина 2

До мінусів даного додатку можна віднести рекламні оголошення та вкорочений функціонал. Головні привади «Weather Underground» - інтерактивна карта з погодними умовами та налаштування для спорту доступні лише після придбання підписки.

Наступний продукт-аналог – це «Daily Weather»[9]. Даний додаток призначений для перегляду погодних умов. На рисунку 1.3 продемонстровано головний екран додатку. До переваг можна віднести простий та незавантажений інтерфейс та можливість вибору міста шляхом пошуку або згідно геоданих.

Рисунок 1.3 – Головний екран додатку «Daily Weather»

Додаток "Daily Weather" має проблеми з оновленням інформації та стабільністю роботи. Під час користування було зафіксовано критичні помилки,

які спричиняють збої в роботі додатку. Дані недоліки значно впливають на досвід користувача і є неприйнятними.

Останнім аналогом було обрано додаток – «WeatherBug»[10]. Інтерфейс додатку виглядає непогано, але виникли сумніви до правдивості наданих даних, а саме до якості повітря. Також, геопозиція була визначена з похибкою. Вигляд головного екрану додатка наведено на рисунках 1.4-1.5.

Рисунок 1.4 – Головний екран додатку «WeatherBug» частина 1

Рисунок 1.5 – Головний екран додатку «WeatherBug» частина 2

Було проведено аналіз мобільних додатків-аналогів та визначено їх переваги та недоліки. Результати аналізу наведено у таблиці 1.1.

Таблиця 1.1 – Порівняльна таблиця характеристик мобільних додатків-аналогів

Додаток / Характеристика	«Weather Underground»	«Daily Weather»	«WeatherBug»	«WeatherApp»
Сучасність дизайну	+	+/-	+/-	+/-
Прогнозування на різні періоди часу	+	+	+	+
Зручність інтерфейсу	+	+/-	+/-	+
Відображення швидкості вітру	+	+	+	+
Точність наведених погодних даних	+	+/-	+/-	+
Стабільність роботи	+	+/-	+	+

Визначення наведених показників:

– Сучасність дизайну: Вказує на ступінь сучасності дизайну додатку. Оцінюється за шкалою з трьох позначень: "+" означає високу сучасність та естетику дизайну, "-" вказує на низький рівень, а "+-" вказує на проміжний рівень.

– Прогнозування на різні періоди часу: Вказує, на можливість прогнозування погоди на різні періоди часу, включаючи години та дні.

– Зручність інтерфейсу: Відображає, наскільки легко користувачам орієнтуватися та взаємодіяти з інтерфейсом додатку. Оцінюється за шкалою з трьох позначень: "+" показує високу зручність та легкість використання, "-" вказує на незручність, а "+/-" вказує на проміжний рівень.

– Відображення швидкості вітру: Вказує, чи додаток відображає інформацію про швидкість вітру.

– Точність наведених погодних даних: Вказує на точність погодних даних, наданих додатком. Оцінюється за шкалою з трьох позначень: "+" вказує

на високу точність, "-" вказує на низьку точність, а "+-" вказує на проміжний рівень.

– Стабільність роботи: Вказує, наскільки стабільно працює додаток та чи має він проблеми з функціональністю. Оцінюється за шкалою з трьох позначень: "+" вказує на високу стабільність, "-" вказує на низьку стабільність, а "+-" вказує на проміжний рівень.

Аргументи для оцінки кожного показника включають:

– Сучасність дизайну: оцінка базується на використанні сучасних дизайнерських тенденцій, естетичності та відповідності дизайну сучасним стандартам[11].

– Прогнозування на різні періоди часу: полягає у наданні детального прогнозу погоди на різні періоди часу, включаючи години та дні

– Зручність інтерфейсу: оцінка засновується на легкості навігації та інтуїтивному розміщенні елементів інтерфейсу[12].

– Історія погоди: оцінка враховує можливість відображення погоди минулих днів.

– Відображення швидкості вітру: полягає у наданні додатком інформації про швидкість вітру.

– Точність наведених погодних даних: оцінка враховує відповідність наведених даних реальним умовам.

– Стабільність роботи: оцінка базується на частоті виникнення помилок та збоїв в роботі додатку.

Таблиця 1.1 надає можливість при розробці звернути увагу на цікаві функціональні доповнення, які можуть бути використані, а також на недоліки, які слід уникнути. При створенні мобільного додатка важливо забезпечити сучасний дизайн і зручний інтерфейс. Серед функціональних доповнень варто відзначити

можливість вибору міста для перегляду погодних показників та додавання інформації про якість повітря і час світанку.

1.3 Постановка задачі

Метою даного дослідження є розробка мобільного додатку, який надає можливість користувачам відстежувати погодні показники. Додаток повинен забезпечити користувачів актуальною інформацією про погоду, включаючи температуру, швидкість вітру, опади.

З погляду розробника, для досягнення мети дослідження, необхідно виконати наступні задачі:

- Провести аналіз інформаційних потреб користувачів відносно погодних показників. Вивчення типових запитів користувачів, їх пріоритетів та очікувань від додатку.
- Забезпечити регулярне оновлення погодних даних.
- Розробити зручний інтерфейс користувача для мобільного додатку. Він повинен бути інтуїтивно зрозумілим, зручним у використанні і привабливим для користувачів.
- Розробити функціонал для відстеження температури. Додаток повинен надавати актуальні дані про температуру в реальному часі, а також можливість перегляду прогнозу.

Основні функціональні вимоги до додатку:

- Надання погодних даних: додаток повинен забезпечувати можливість отримання актуальних погодних даних.
- Відображення погоди: додаток повинен надавати зручний інтерфейс для відображення погодних показників.

– Прогнозування погоди: додаток повинен мати функцію прогнозу погоди, яка дає користувачам можливість здійснювати свою діяльність на основі передбачених погодних умов.

Вимоги до проекту в цілому, структури мобільного додатку, видів забезпечення та функціонування системи описані у технічному завданні на розробку проекту (додаток А).

Для реалізації мобільного додатку буде використана мова програмування Kotlin [13]. Kotlin є сучасною і міцно типізованою мовою, яка працює на віртуальній машині Java (JVM) і має підтримку для розробки мобільних додатків на платформі Android. Вона пропонує чистий синтаксис, безпечну роботу з нульовими значеннями, функціональні можливості та інші покращення порівняно з Java. Kotlin є офіційною мовою для розробки Android-додатків.

Для отримання погодних даних у додатку буде використана Firebase Realtime Database [14]. Ця база даних є хмарною базою даних від Google і надає реальний час синхронізації даних між різними пристроями. Firebase Realtime Database дозволяє безпосередньо спілкуватися з базою даних з мобільних пристроїв, забезпечуючи швидкий та ефективний доступ до даних. Оновлення даних відбувається в реальному часі, яка буде оновлюватись адміністраторами. Середовище розробки: Android Studio [15].Воно надає потужні інструменти для розробки, тестування та налагодження додатків, а також інтегровану підтримку для мови Kotlin.

Вибір Kotlin, Firebase Realtime Database та Android Studio дозволить забезпечити швидку та ефективну розробку мобільного додатку з підтримкою актуальних погодних даних та зручним інтерфейсом користувача.

2 МОДЕЛЮВАННЯ ТА ПРОЕКТУВАННЯ МОБІЛЬНОГО ДОДАТКУ ВІДСТЕЖЕННЯ ПОГОДНИХ ПОКАЗНИКІВ

2.1 Функціональне моделювання мобільного додатку

Функціональне моделювання полягає у аналізі та визначенні функції, яка виконується в додатку. Одним з інструментів функціонального моделювання є IDEF0,він надає структурований підхід до представлення функцій та їх взаємодії в системі. Це сприяє охопленню більшої кількості спеціалістів в окремій галузі та полегшенню прийняття рішень відносно системи.

На рисунках 2.1-2.2 наведені діаграми функціонального моделювання з використанням IDEF0 відносно користувацького та адміністративного додатків. Вони включають вхідні та вихідні данні, функціональний блок, обмеження та механізми для виконання функції.

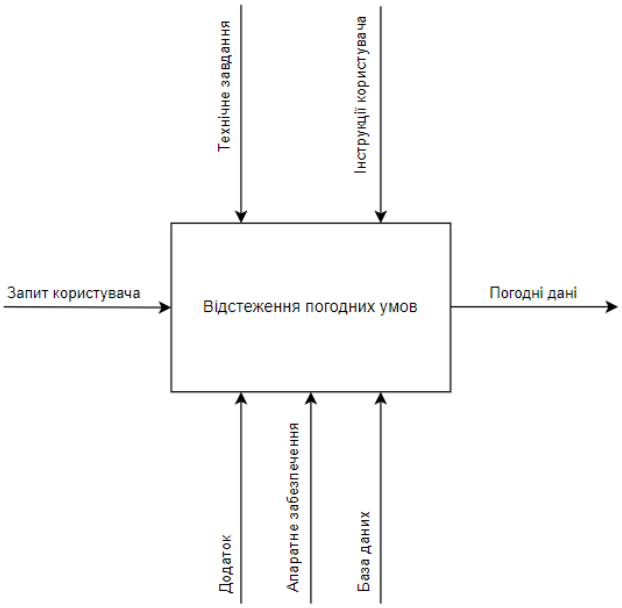

Рисунок 2.1 – Діаграма IDEF0 для користувацького додатку

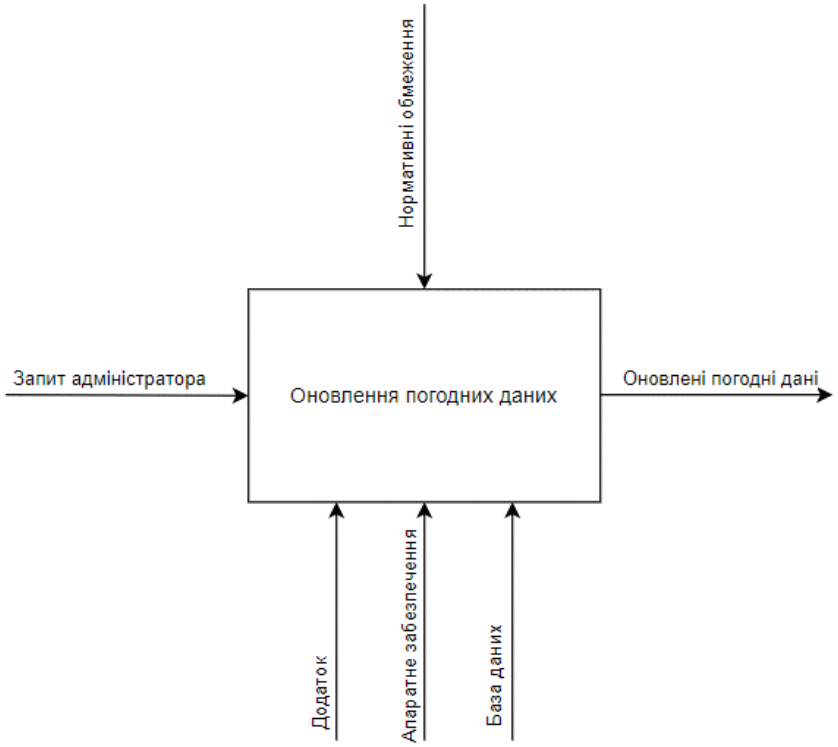

Рисунок 2.2 – Діаграма IDEF0 для додатку адміністратора

Вхідні дані:

- Запит користувача (відображення погодинного прогнозу на теперішній день або денного на наступні 7 днів)
- Запит адміністратора (оновлення погодних даних)

Контролюючі та обмежуючі механізми:

- Інтерфейс WeatherApp (обмеження у вигляді доступних в інтерфейсі можливостей для перегляду погодних даних)
- Інтерфейс WeatherAdminApp (обмеження у вигляді доступних в інтерфейсі можливостей для оновлення погодних даних)

Механізми, що використовуються для виконання роботи:

- WeatherApp (користувацький мобільний додаток, який отримує данні з Firebase DB)

– WeatherAdminApp (додаток для адміністрування даних в Firebase DB)
– Firebase DB (система управління даними, використовується для обміну даними між WeatherAdminApp та WeatherApp)

Вихідні дані:

– Погодні дані (отримані данні відносно запиту користувача)
– Оновлені погодні дані (в Firebase DB)

З метою отримання детальної інформації про функції додатків, було змодельовано розгорнуті моделі їх роботи.

Декомпозиція функціональних моделей мобільних додатків представлена на рисунках 2.3-2.4.

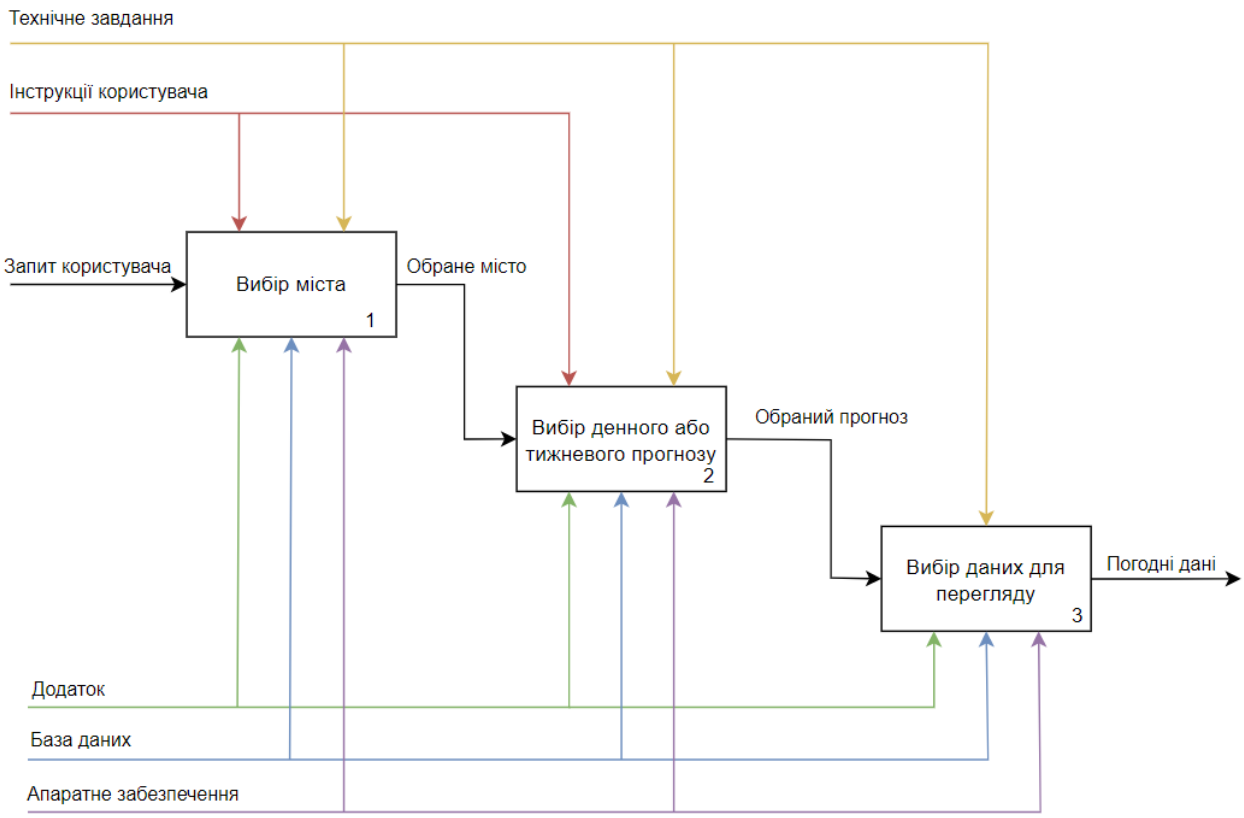

Рисунок 2.3 – Декомпозиція бізнес процесу користувацького додатку

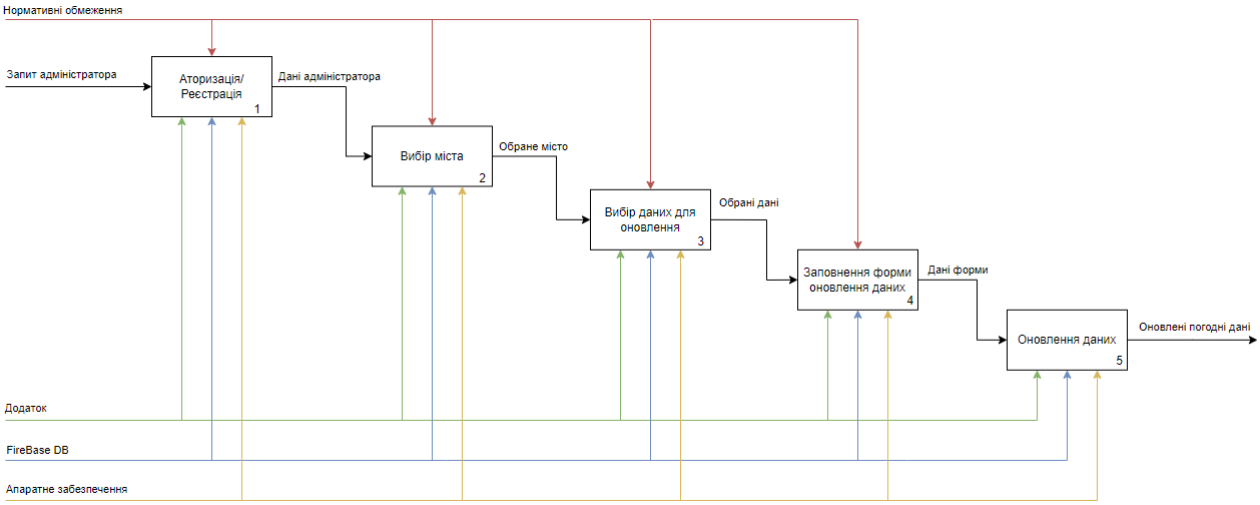

Рисунок 2.4 – Декомпозиція бізнес процесу додатку адміністратора

2.2 Проектування мобільного додатка

2.2.1 Проектування діаграми варіантів використання

Для опису функціонального призначення, було створено Use-Case діаграму, яка відображає моделювання варіантів використання з точки зору кінцевих споживачів.

Діаграма варіантів використання представлена на рисунку 2.5.

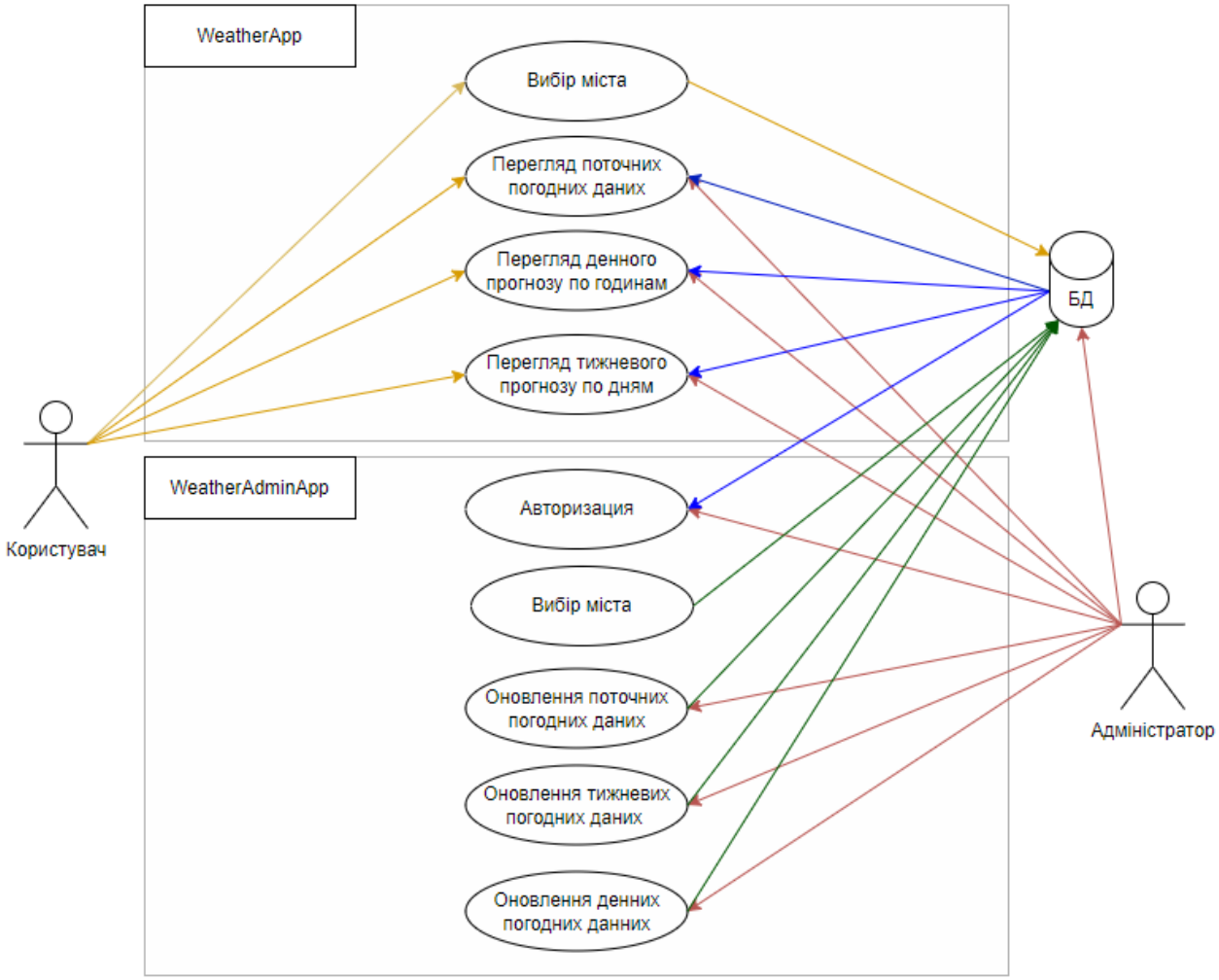

Рисунок 2.5 – Діаграма варіантів використання

На діаграмі наведено 3 актора:

– користувач;
– адміністратор;
– база даних (надає можливість зберігати, читати та редагувати дані).

2.2.2 Проектування діаграми компонентів мобільного додатку

Для фізичної реалізації конкретної системи потрібно мати відповідні матеріальні елементи, що відповідають логічному опису системи. Фізичне подання моделі використовується для цих елементів. Діаграма компонентів є

одним з графічних засобів для визначення архітектури системи, що розробляється. У діаграмі компонентів основними елементами є компоненти, інтерфейси та залежності між ними, які відображають структуру та взаємозв'язки між елементами системи.

Діаграми компонентів користувацького та адміністративного додатків представлені на рисунках 2.6-2.7.

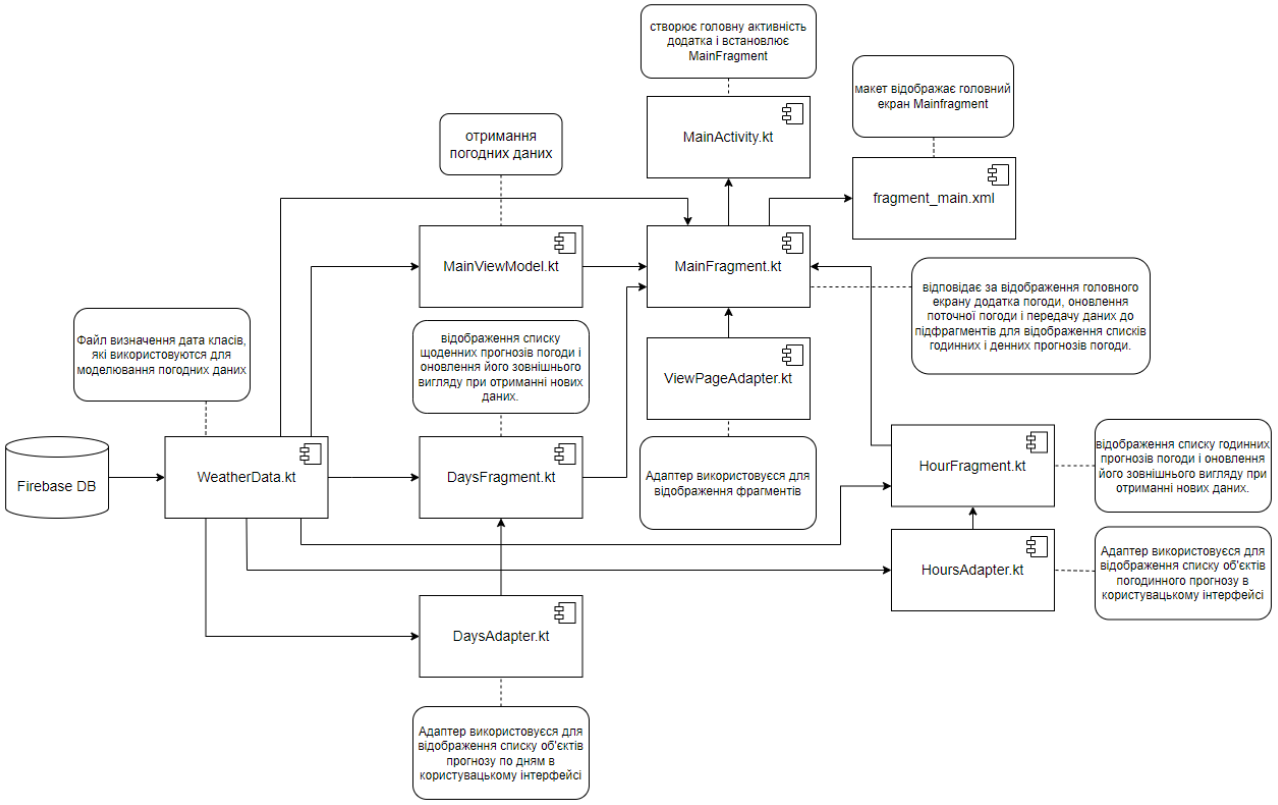

Рисунок 2.6 – Діаграма компонентів додатку користувача

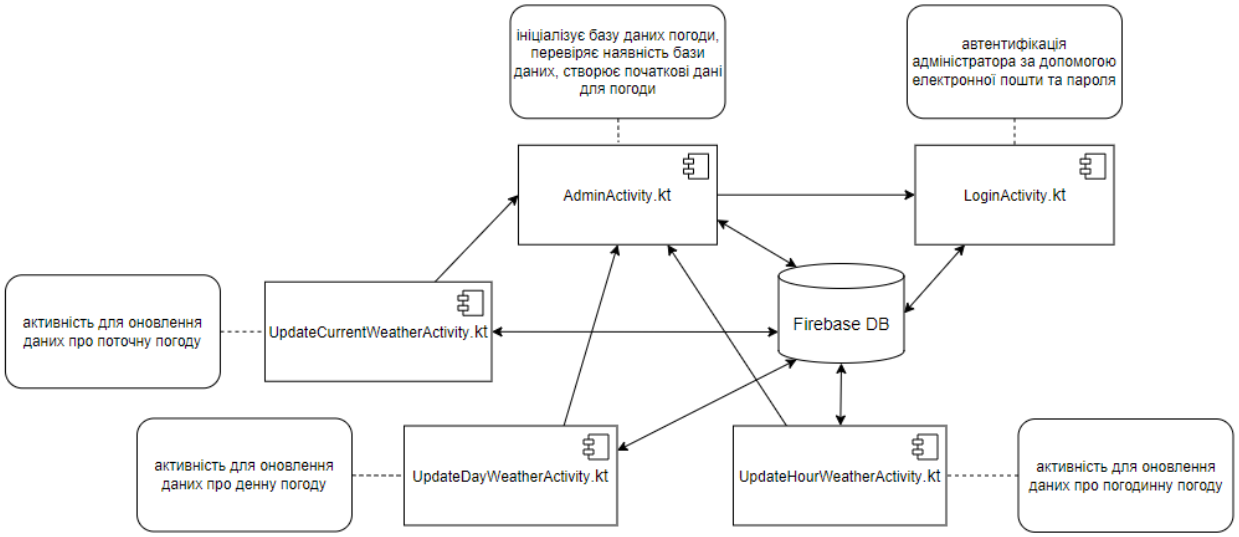

Рисунок 2.7 – Діаграма компонентів додатку адміністратора

2.3 Проектування моделі бази даних

У сучасному світі бази даних є невід'ємною частиною багатьох додатків та систем. При проектуванні бази даних одним з важливих аспектів є вибір між різними типами баз даних. Два основних типи, це SQL (реляційні бази даних) та NoSQL (не реляційні бази даних).

SQL бази даних є традиційним типом баз даних, який використовує реляційну модель для збереження даних у вигляді таблиць зі структурованими рядками та стовпцями.

NoSQL бази даних, з іншого боку, представляють нове покоління баз даних, які не використовують реляційну модель. Вони призначені для роботи з великими обсягами нереляційних даних, таких як документи, графи, ключ-значення. Одним

із прикладів NoSQL баз даних є Firebase Realtime Database, який забезпечує швидке та масштабоване зберігання та синхронізацію даних в реальному часі.

Було обрано засіб реалізації NoSql бази даних Firebase. Приклади зображені на рисунках 2.8-2.9.

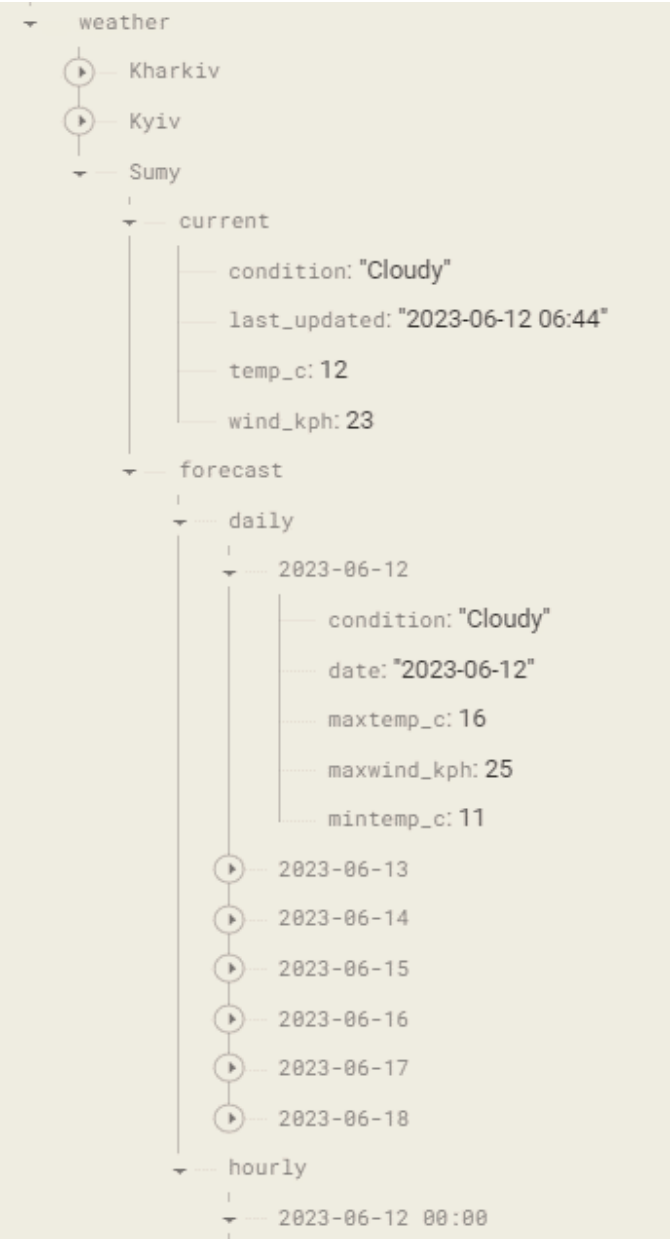

Рисунок 2.8 – Ілюстрація документу СУБД Firebase для збереження даних про показники погоди

3 РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ

3.1 Архітектура мобільного додатку

3.1.1 Архітектура мобільного додатку користувача

На початковому етапі розробки мобільного додатку користувача для відстеження погодних показників побудована його архітектура на основі патерну Model-View-ViewModel (MVVM), яка зображена на рисунку 3.1.

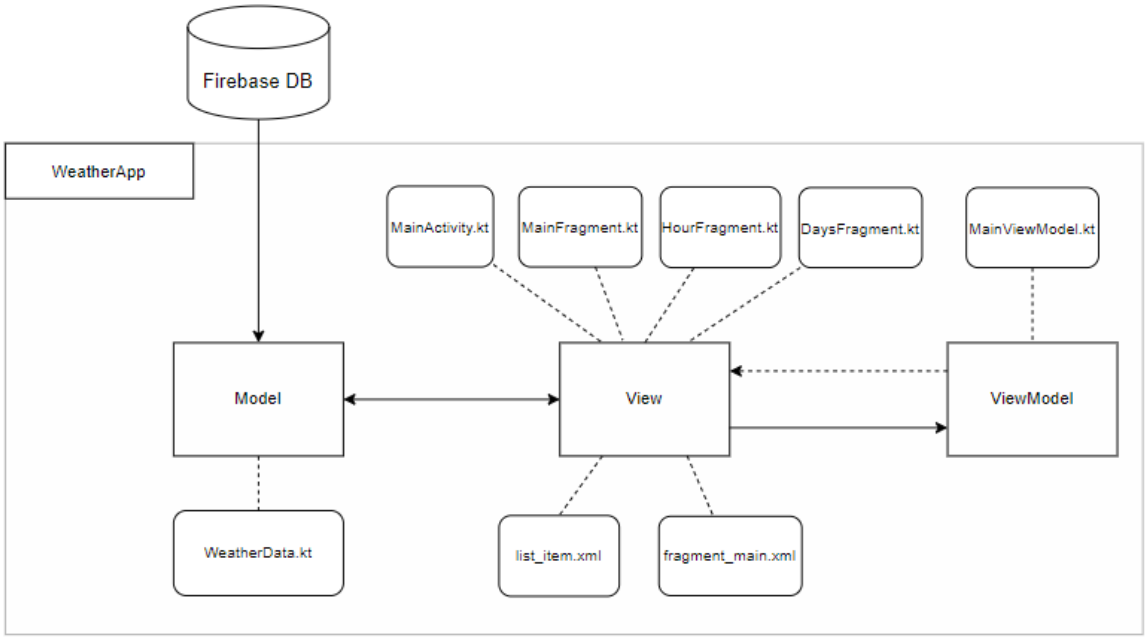

Рисунок 3.1 – Архітектура мобільного додатку користувача

Архітектура додатку користувача включає наступні компоненти:

– Model (WeatherData.kt);
– View (MainActivity.kt, MainFragment.kt, HourFragment.kt, DaysFragment.kt, файли макетів: list_item.xml та fragment_main.xml);
– ViewModel (MainViewModel.kt).

3.1.2 Архітектура мобільного додатку адміністратора

На етапі розробки мобільного додатку адміністратора для оновлення погодних даних побудована його архітектура, яка заснована на парадигмі Model-View-Controller (MVC). Архітектура додатку адміністратора зображена на рисунку 3.2

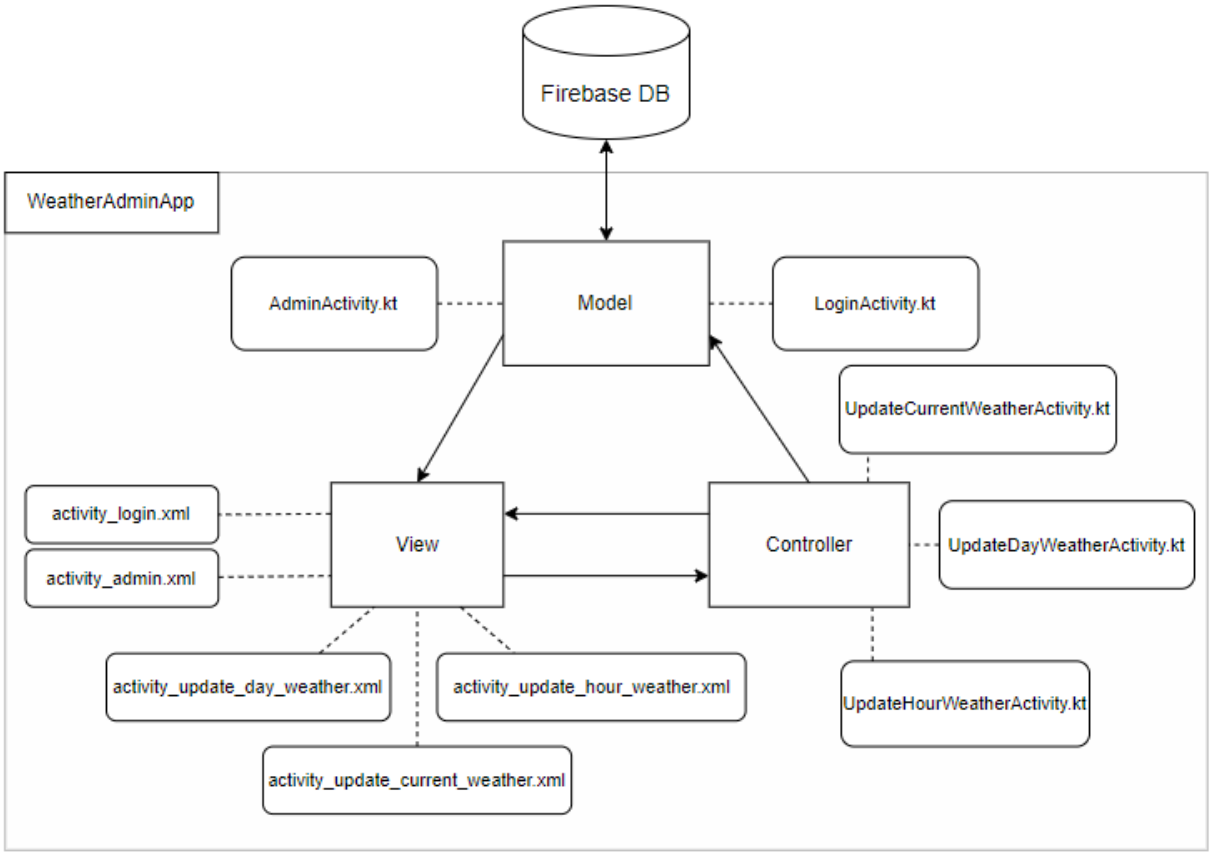

Рисунок 3.2 – Архітектура мобільного додатку адміністратора

Архітектура додатку адміністратора включає наступні компоненти:

- Model (LoginActivity.kt, AdminActivity.kt);
- View (Файли макетів інтерфейсу)
- Controller (Файли логіки оновлення інформації).

3.2 Програмна реалізація

3.2.1 Створення дизайну мобільного додатку користувача

У процесі розробки мобільного додатку користувача передбачено створення макетів для взаємодії з користувачем. За допомогою мови розмітки сторінок XML та середовища розробки Android Studio, було створено макети головного екрану та одного елементу списку з прогнозом відповідно до вимог, визначених в додатку А.

Створені макети представлені на рисунках 3.3-3.4.

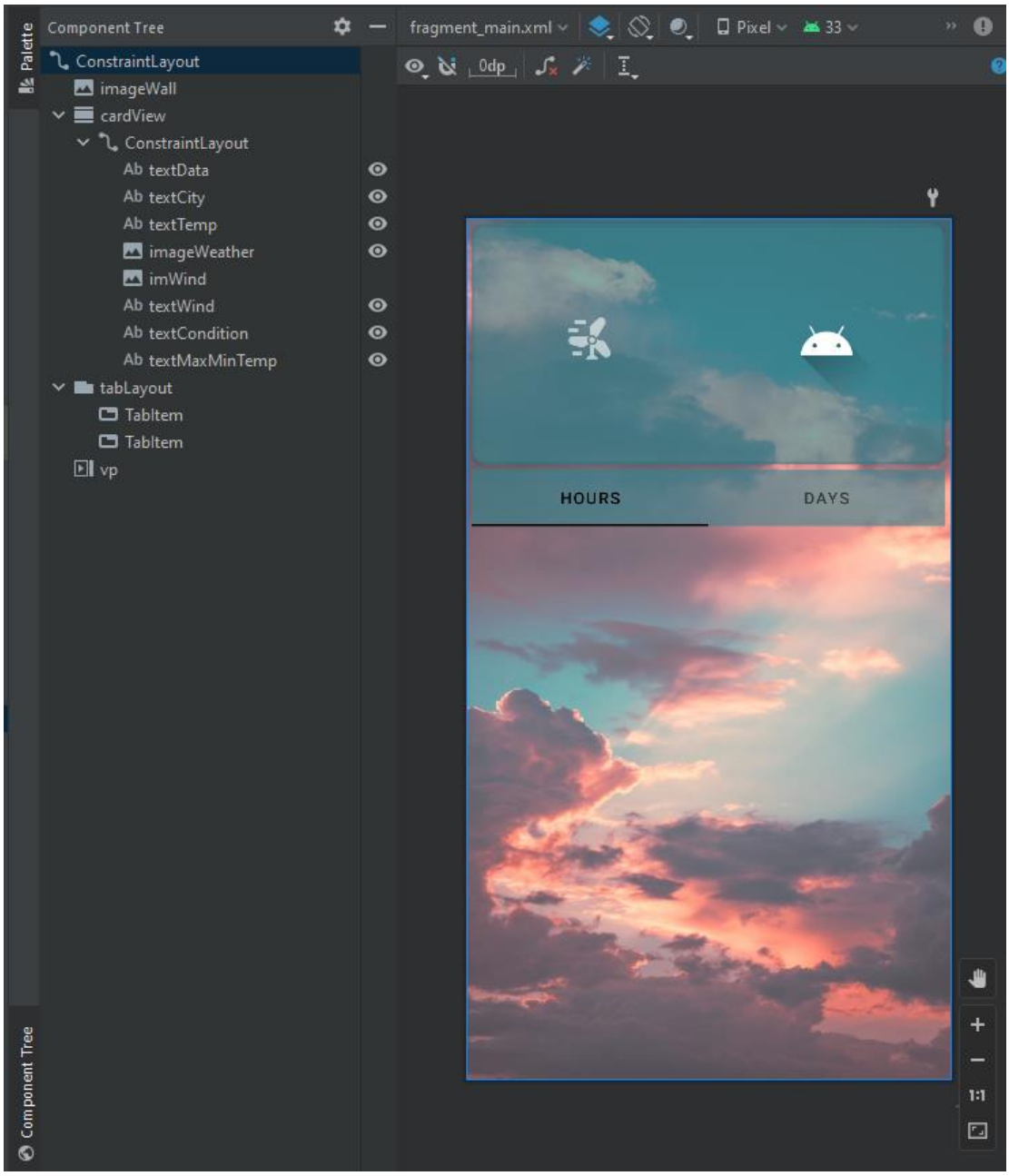

Рисунок 3.3 – Макет головного екрану мобільного додатка користувача

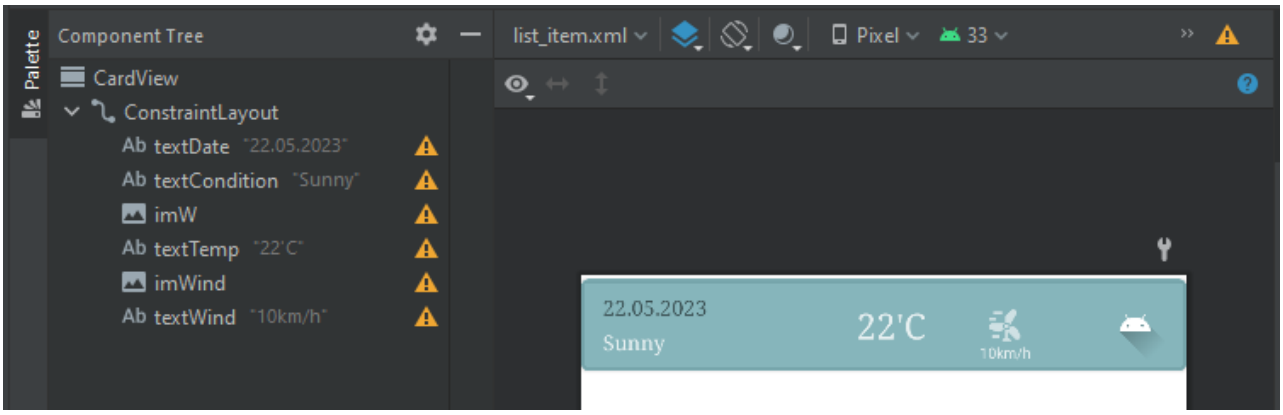

Рисунок 3.4 – Макет елемента списку прогнозу мобільного додатка користувача

3.2.2 Створення дизайну мобільного додатку адміністратора

У процесі розробки мобільного додатку адміністратора передбачено створення макетів для взаємодії з користувачем. За допомогою мови розмітки сторінок XML та середовища розробки Android Studio, було створено макети екранів:

- Авторизації;
- Вибору активності для оновлення даних;
- Оновлення поточних погодних даних;
- Оновлення прогнозу по дням
- Оновлення прогнозу по годинам

Створені макети представлені на рисунках 3.5-3.9.

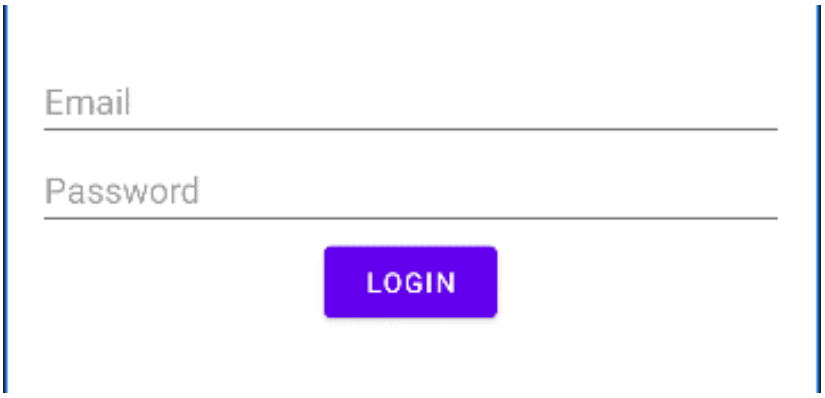

Рисунок 3.5 – Макет екрану авторизації

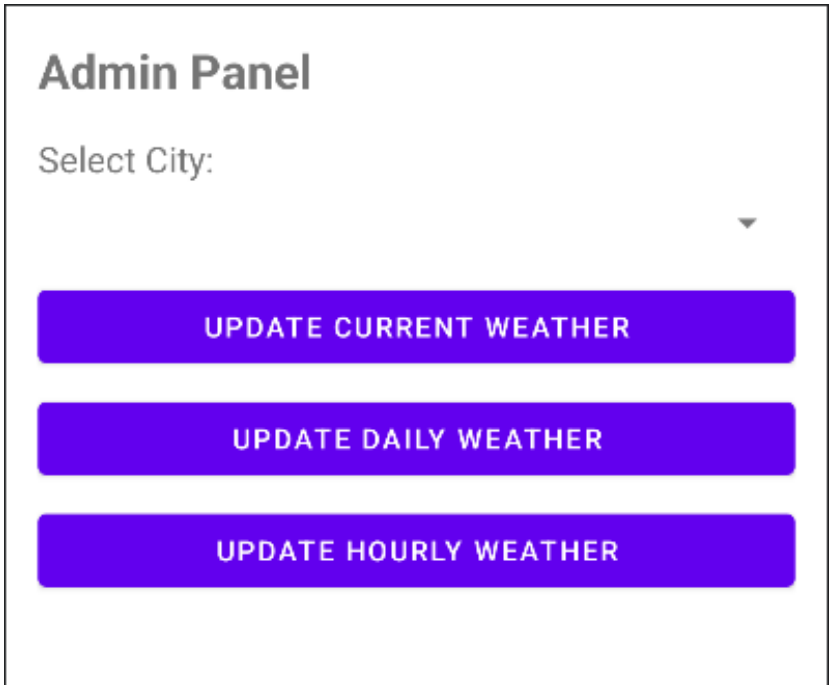

Рисунок 3.6 – Макет екрану вибору активності

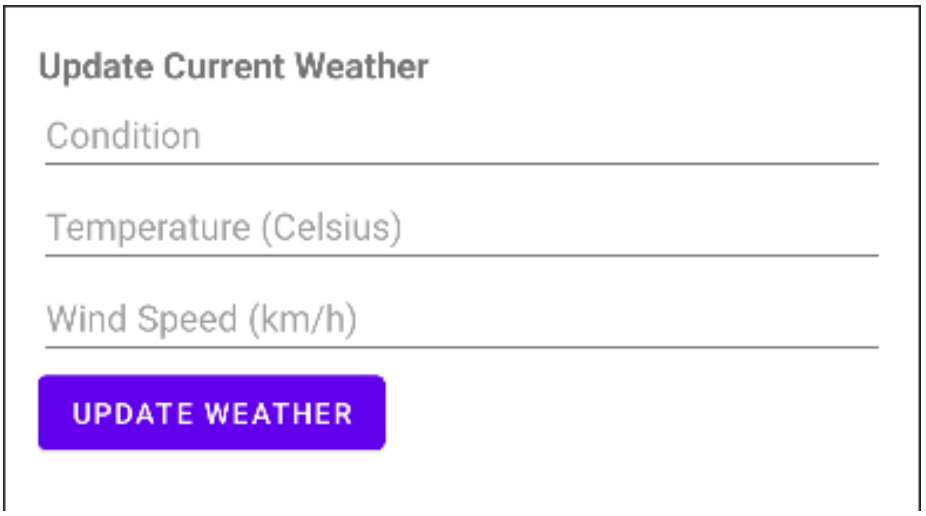

Рисунок 3.7 – Макет екрану оновлення поточних погодних даних

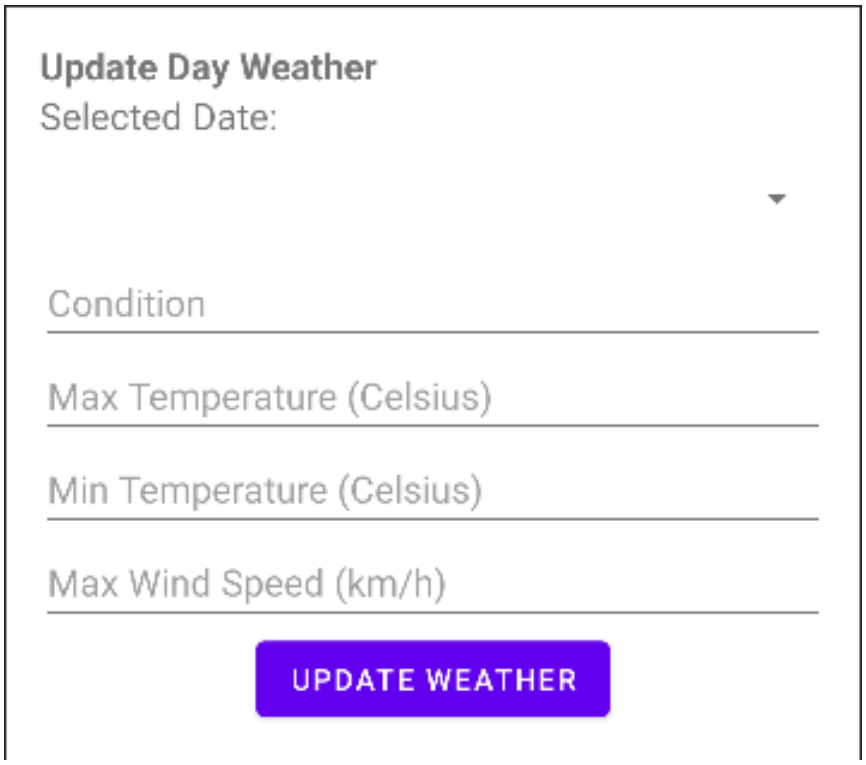

Рисунок 3.8 – Макет екрану оновлення прогнозу по дням

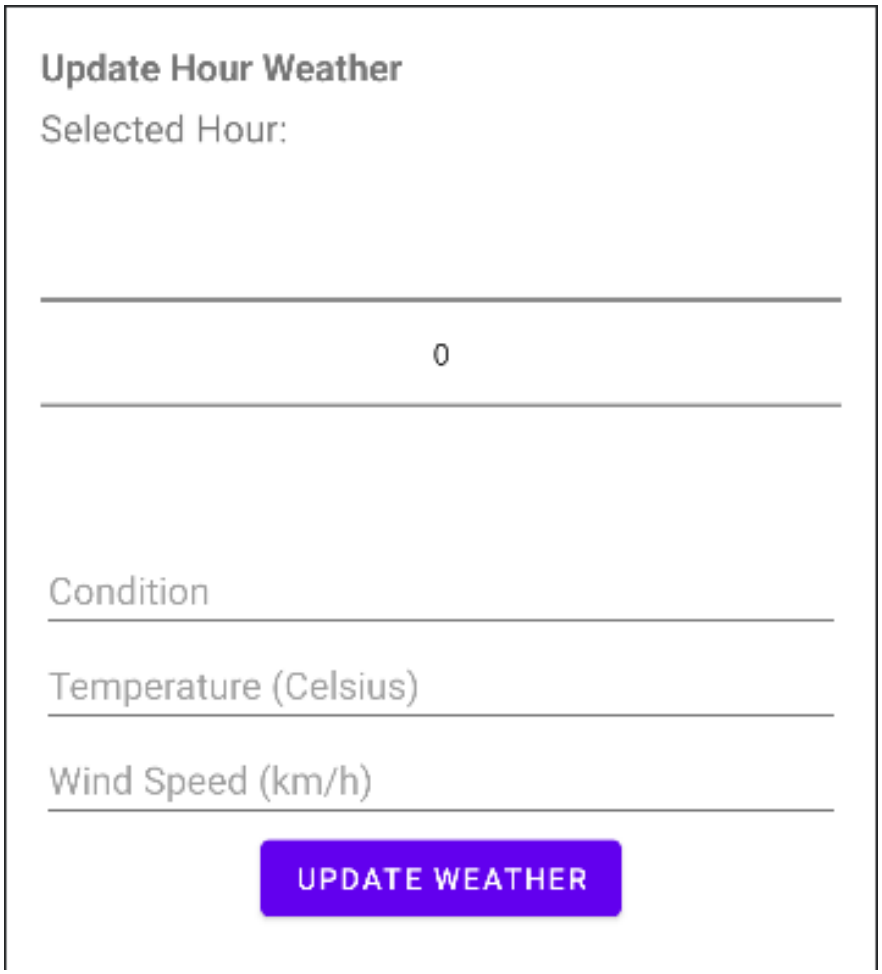

Рисунок 3.9 – Макет екрану оновлення прогнозу по годинам

3.2.3 Налаштування бази даних

Налаштування бази даних є важливим етапом розробки мобільного додатку. Було використано Firebase Realtime Database. Етапи підключення та налаштування бази даних наведено на рисунках 3.10 – 3.11.

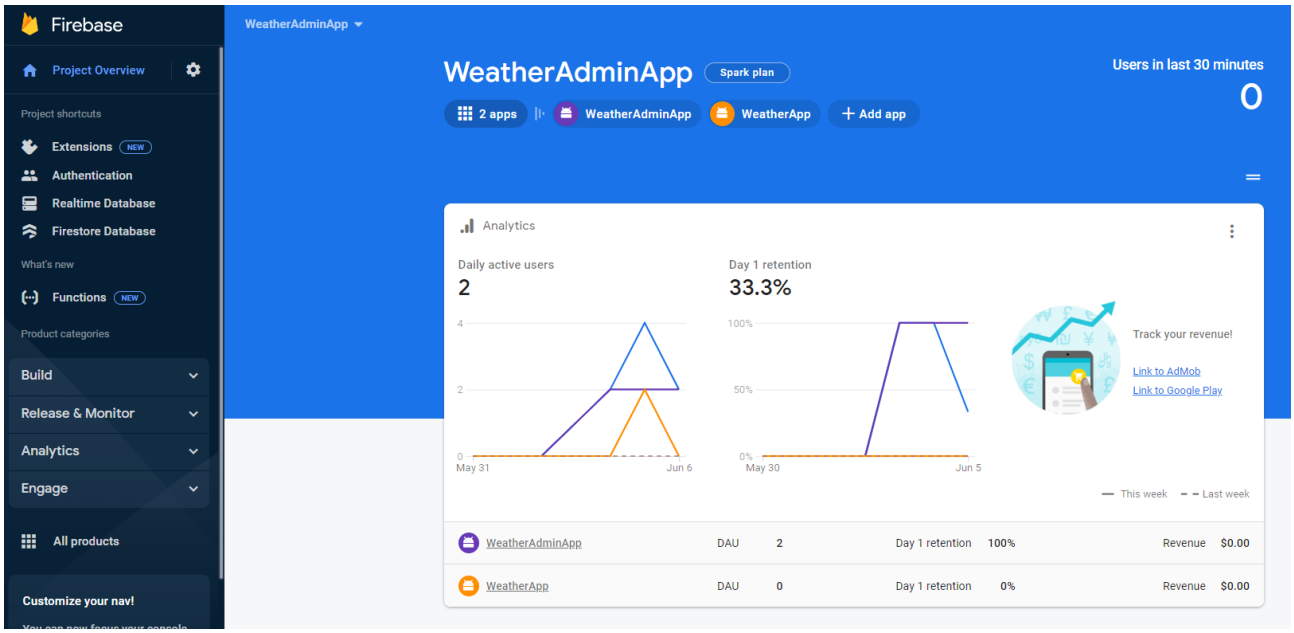

Рисунок 3.10 – Додання та налаштування проектів в Console Firebase

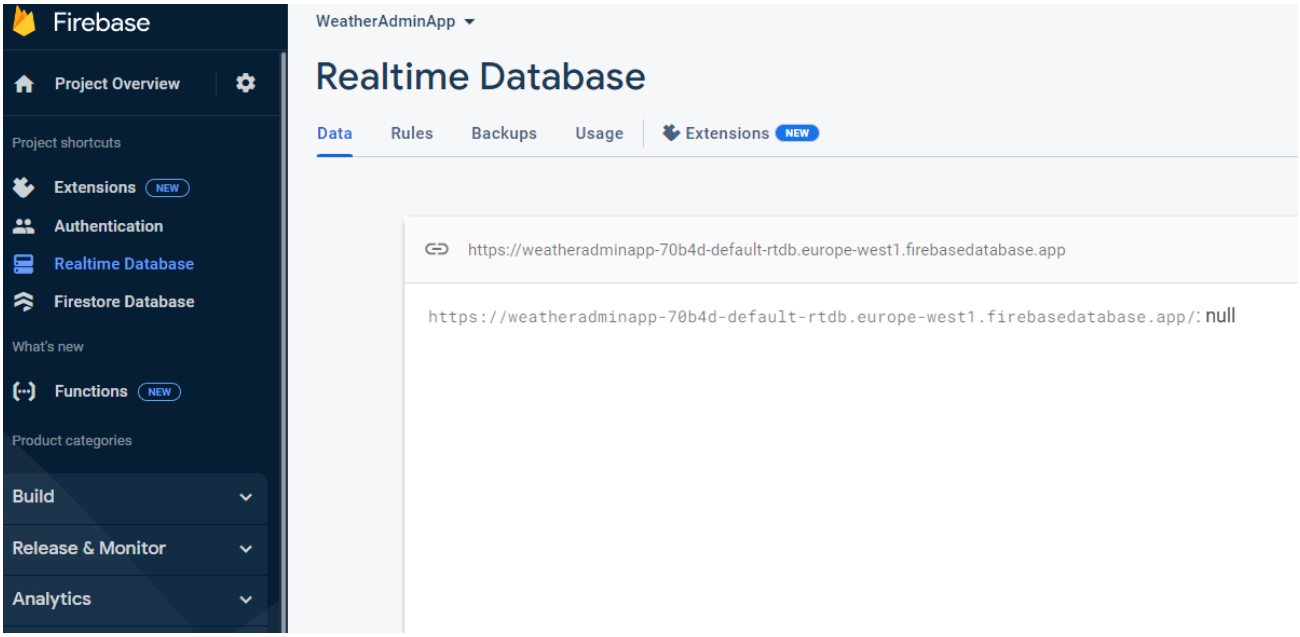

Рисунок 3.11 – Підключення Realtime Database

3.2.4 Програмна реалізація додатку для адміністраторів

Першим етапом реалізації є створення активності для авторизації адміністраторів:

– Розробка макета інтерфейсу для авторизації;
– Ініціалізація Firebase Auth та обробка натискання кнопки входу;
– Перевірка облікових даних користувача та надання відповідного повідомлення.

Додання даних для авторизації адміністраторів відбувається з використанням Console Firebase. Приклад сторінки додання даних для авторизації наведено на рисунку 3.12.

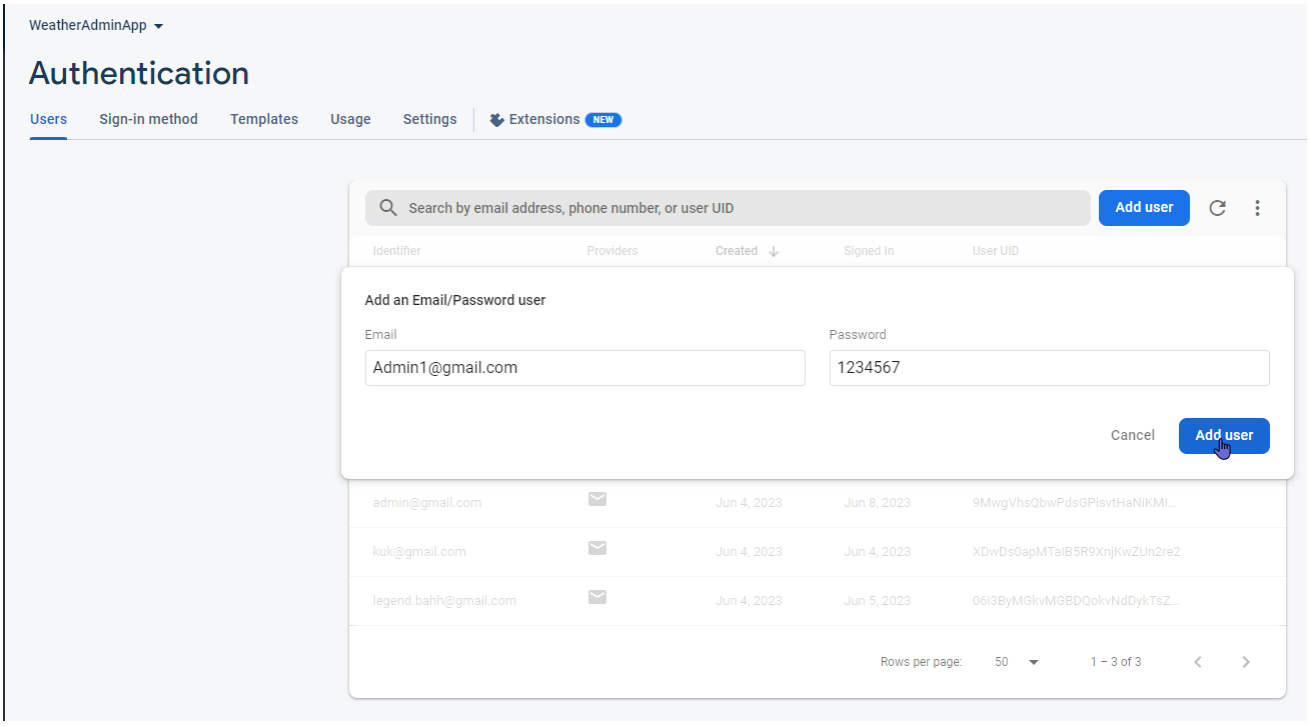

Рисунок 3.12 – Приклад додання даних для авторизації адміністратора

Наступним етапом є створення активності для адміністрування:

– Розробка інтерфейсу для головної активності адміністратора;
– Ініціалізація Firebase Realtime Database та налаштування посилань на дані;
– Налаштування випадаючого списку з містами;
– Обробка натискання кнопок переходу до наступної активності

Далі оновлення даних в базі розгалужується на три активності з записом даних до бази:

– Оновлення поточної погоди;
– Оновлення денної погоди;
– Оновлення годинної погоди.

При розробці макету екрана оновлення денної погодної інформації, було використано випадаючий список, з елементів 7 днів. Приклад реалізації логіки списку наведено на рисунку 3.13.

```
private fun generateDaysList(): List<String> {
    val days = ArrayList<String>()
    val calendar : Calendar = Calendar.getInstance()
    val dateFormat = SimpleDateFormat( pattern: "yyyy-MM-dd", Locale.getDefault())

    // Додаємо сьогоднішню дату до списку
    val currentDate : String = dateFormat.format(calendar.time)
    days.add(currentDate)

    // Генеруємо наступні 6 днів, починаючи з наступного дня
    for (i : Int in 1 ≤ .. ≤ 6) {
        calendar.add(Calendar.DAY_OF_MONTH, 1)
        val date : String = dateFormat.format(calendar.time)
        days.add(date)
    }

    return days
}
```

Рисунок 3.13 – Функція отримання списку з датами 7 днів

При розробці макету оновлення годинної інформації, для вибору часу було використано Number Piker. Його налаштування зображені на рисунку 3.14.

```
// Налаштування NumberPicker для вибору години дня
binding.hourPicker.minValue = 0
binding.hourPicker.maxValue = 23
binding.hourPicker.setFormatter { value : Int ->
    val hourFormat = DecimalFormat( pattern: "00")
    val hour : String! = hourFormat.format(value)
    "$hour:00" ^setFormatter
}
```

Рисунок 3.14 – Налаштування NumberPicker

3.2.5 Програмна реалізація клієнтського додатку

На першому етапі розробки було створено два фрагменти: HoursFragment та DaysFragment. Фрагмент - це компонент, який представляє окремий екран або частину інтерфейсу. У цих фрагментах використано RecyclerView для відображення списків годин та днів, а також створено відповідні адаптери: HoursAdapter та DaysAdapter.

Другий етап розробки пов'язаний з моделюванням даних. Створено дата класи WeatherData, CurrentWeather, ForecastWeather, DailyWeather, HourlyWeather для представлення інформації про погоду. Ці моделі дозволяють структурувати дані з Firebase та передавати їх між різними компонентами додатка. Приклад перелічених дата класів наведено на рисунку 3.15.

```kotlin
data class WeatherData(
    val weather: Map<String, CityWeather>? = null,
    val current: CurrentWeather? = null,
    val forecast: ForecastWeather? = null
)

data class CityWeather(
    val current: CurrentWeather?,
    val forecast: ForecastWeather?
)

data class CurrentWeather(
    val condition: String? = null,
    val last_updated: String? = null,
    val temp_c: Double? = null,
    val wind_kph: Double? = null
)


data class ForecastWeather(
    val daily: Map<String, DailyWeather>? = null,
    val hourly: Map<String, HourlyWeather>? = null
)


data class DailyWeather(
    val condition: String? = null,
    val date: String? = null,
    val maxtemp_c: Double? = null,
    val maxwind_kph: Double? = null,
    val mintemp_c: Double? = null
)

data class HourlyWeather(
    val condition: String? = null,
    val temp_c: Double? = null,
    val time: String? = null,
    val wind_kph: Double? = null
)
```

Рисунок 3.15 – Дата класи для представлення інформації про погоду

На третьому етапі створено ViewModel для керування даними погоди. Клас MainViewModel розширює базовий клас ViewModel і містить MutableLiveData, який надає можливість спостерігати за змінами даних.

Останній етап розробки пов'язаний зі збиранням та відображенням даних погоди. Було використано бібліотеку Coil для завантаження та відображення іконок погоди з локального репозиторію. Також використано DiffUtil для ефективного оновлення списків даних у адаптерах.

3.3 Використання мобільного додатку

3.2.5 Використання додатку зі сторони адміністратора

Після запуску додатку, адміністратор потрапляє на екран авторизації. Приклад запуску додатка адміністратора наведений на рисунку 3.16.

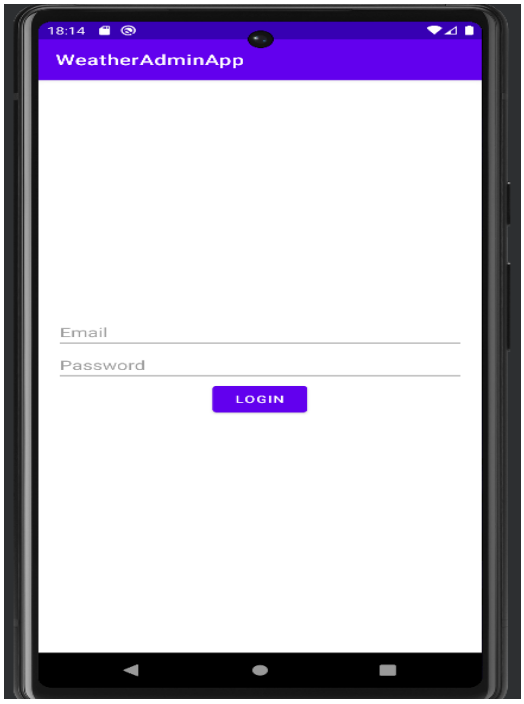

Рисунок 3.16 – Екран авторизації

Якщо, адміністратор вводить невірні дані для авторизації, додаток виводить спливаюче повідомлення, про невдалу авторизацію. Приклад введення некоректних даних наведений на рисунку 3.17.

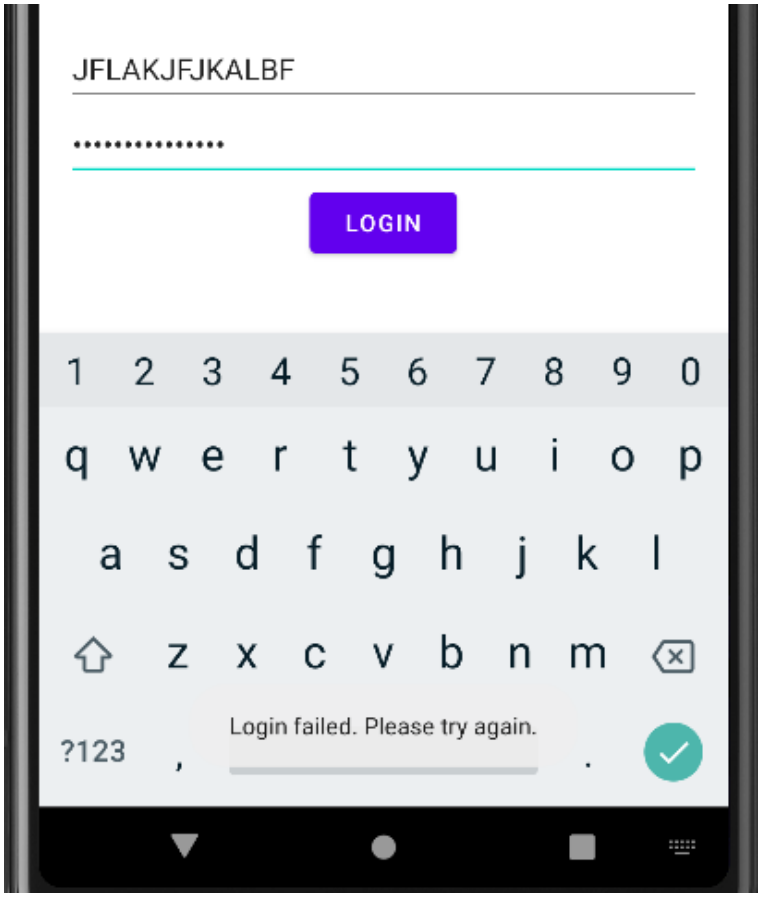

Рисунок 3.17 – Спливаюче повідомлення про помилку в ході авторизації

Після вдалої авторизації, адміністратор потрапляє в меню вибору активності (рис.3.18).

Тут він може вибрати оновлювання даних для вибраного міста:

– Update Current Weather (Оновлення поточної погоди)
– Update Day Weather (Оновлення денної погоди)
– Update Hour Weather (Оновлення годинної погоди)

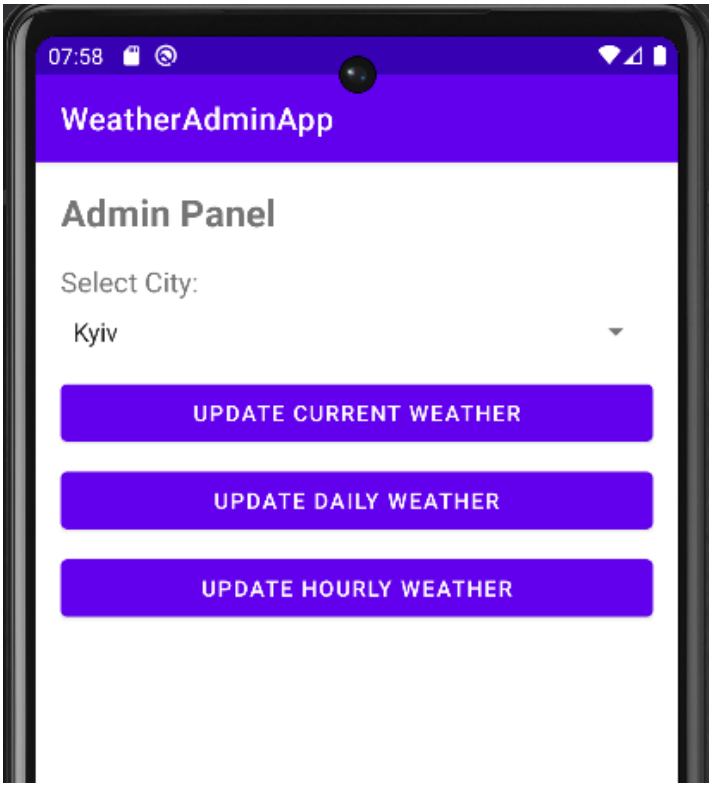

Рисунок 3.18 – Екран вибору даних для редагування

Натискаючи на першу кнопку адміністратор потрапляє на екран зміни поточної погоди з формою для запису погодних даних(рис. 3.19), в формі йому потрібно записати стан погоди, температуру повітря та швидкість вітру, в разі запису не коректних даних, з’явиться повідомлення про невірно введені дані(рис.3.20).

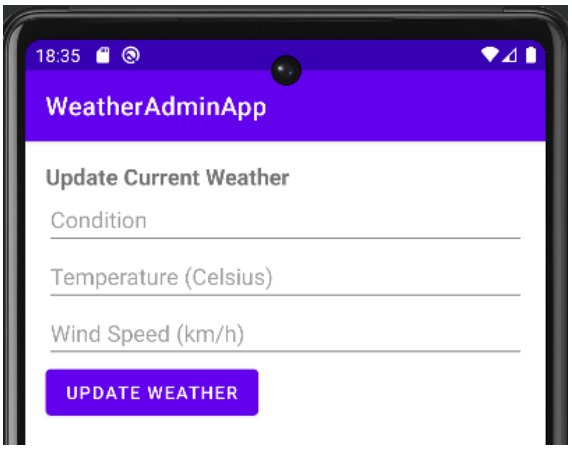

Рисунок 3.19 – Екран з формою для запису поточної погоди

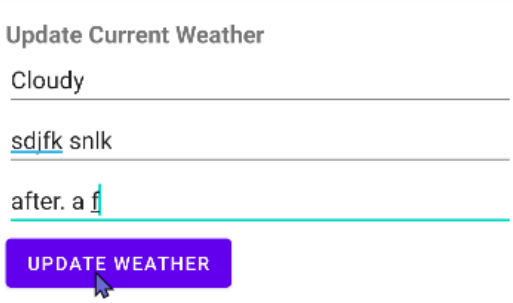

Рисунок 3.20 – Приклад введення некоректних даних

Після заповнення погодних показників і натискання кнопки Update Weather, користувач потрапляє в попереднє меню та отримує оповіщення, про успішне оновлення інформації (рис.3.21)

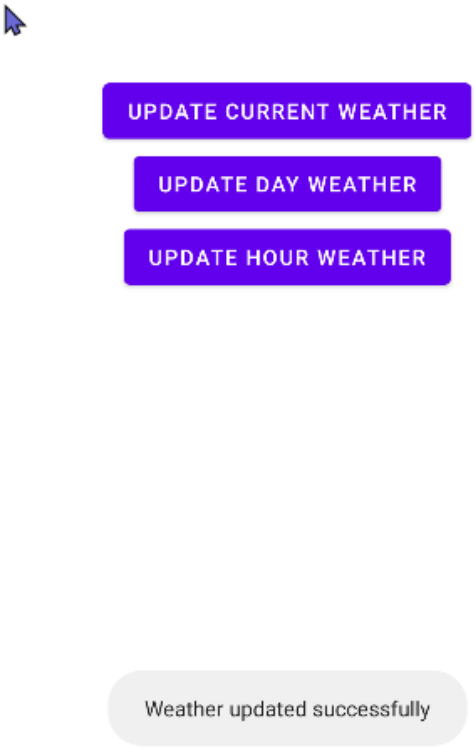

Рисунок 3.21 – Повернення в меню з вибором активності

На рисунках 3.22-3.23, зображено екран оновлення погоди по дням.

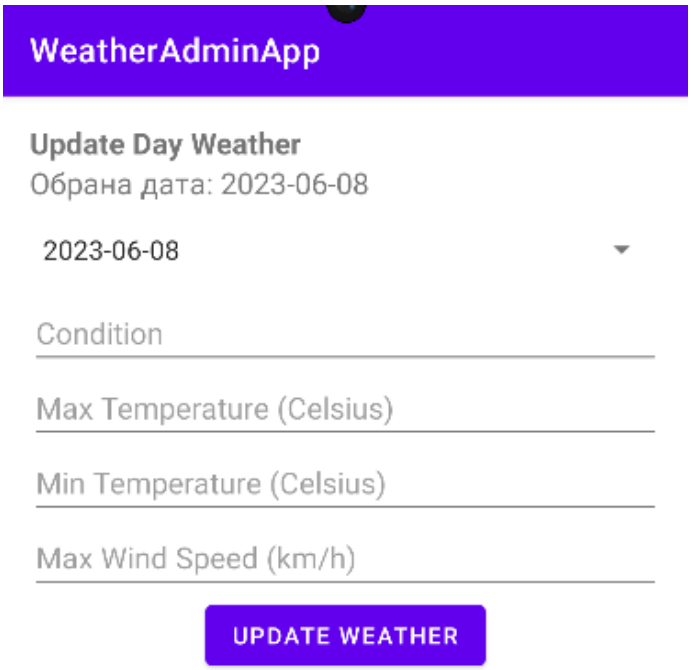

Рисунок 3.22 Екран оновлення погоди по дням

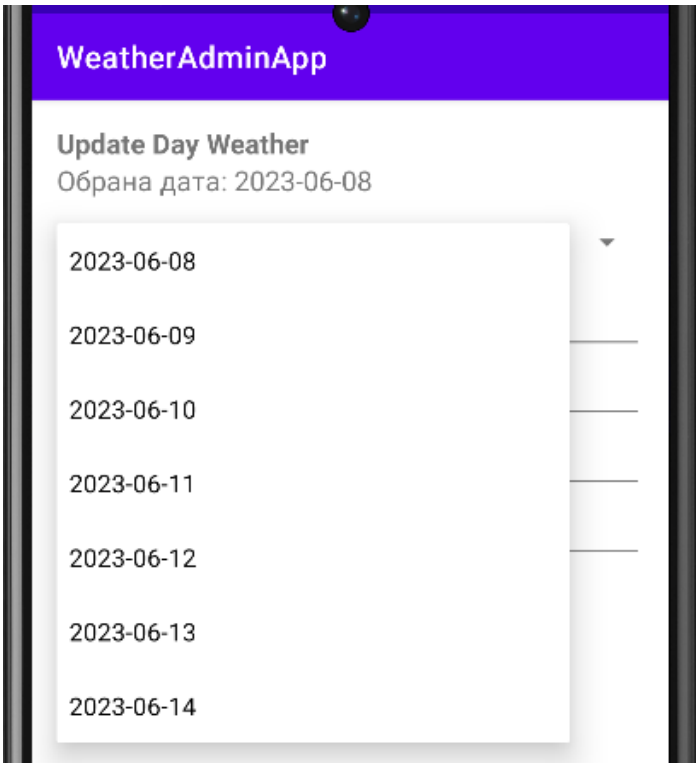

Рисунок 3.23 – Список для вибору дня

При виборі оновлення по годинам, адміністратор потрапляє на екран з формою для заповнення з жестовим вибором часу (рис.3.24).

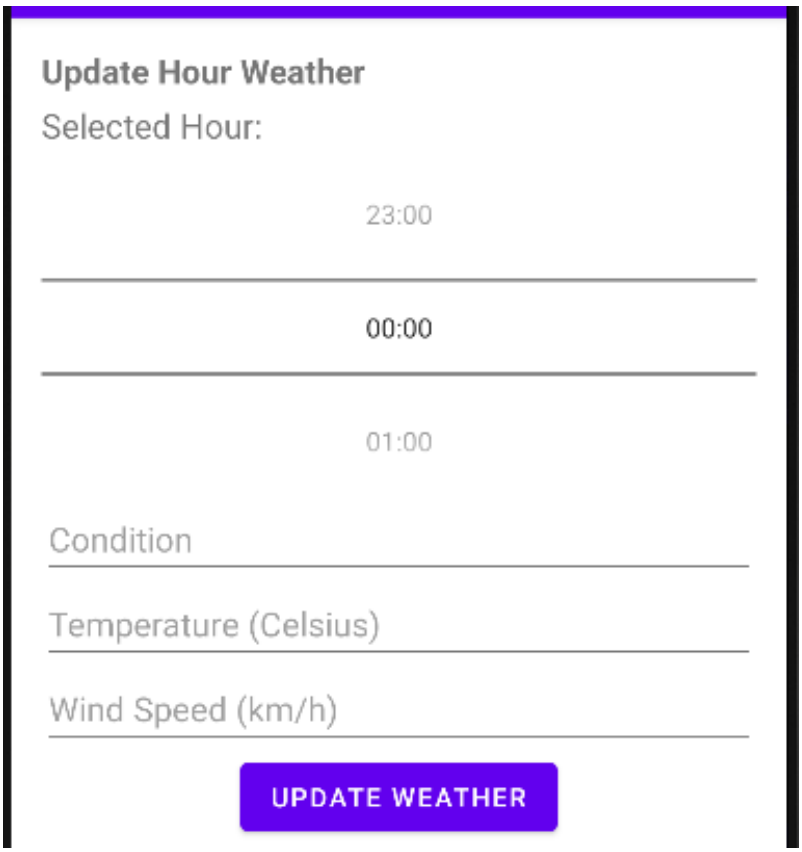

Рисунок 3.24 – Екран оновлення погодинної інформації

3.2.5 Використання клієнтського додатку

Після запуску додатку, користувач потрапляє на головний екран. Приклад головного екрану зображено на рисунку 3.25

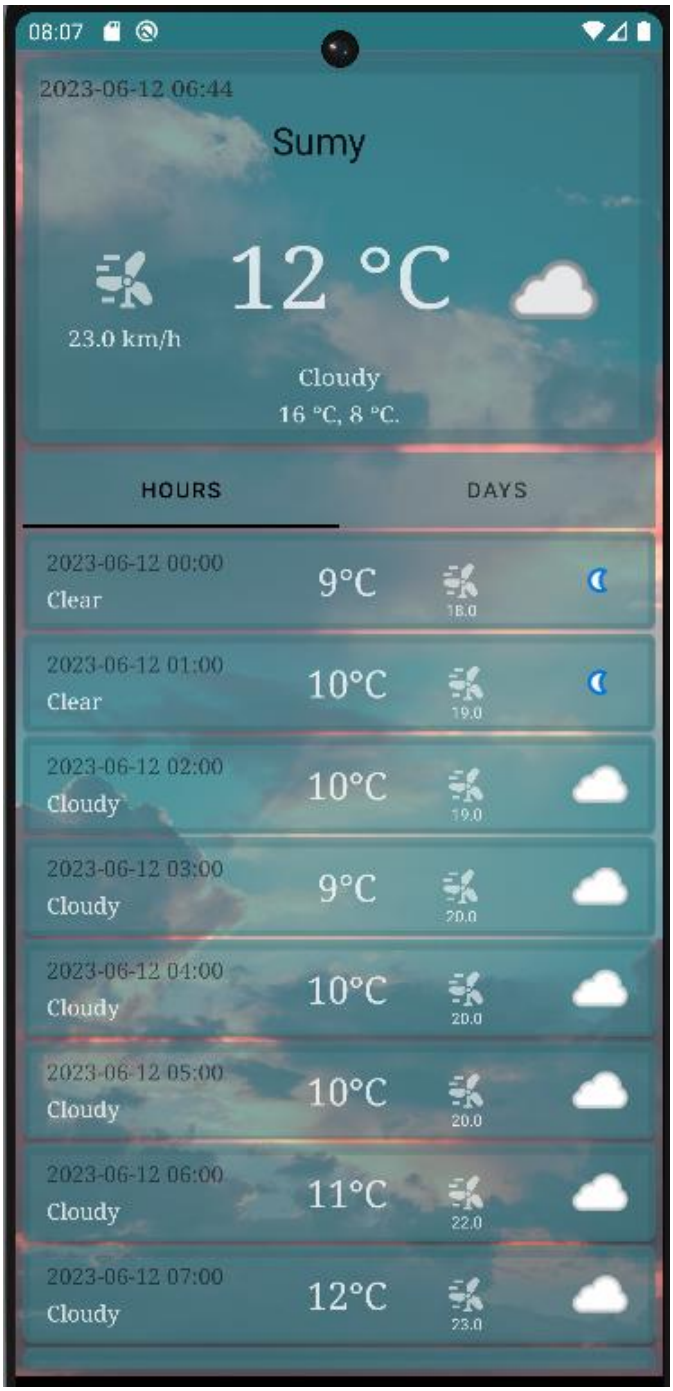

Рисунок 3.25 – Головний екран клієнтського додатку

Головний екран додатку містить поточну погодну інформацію, два списки з погодинним та денним прогнозом. Приклад прогнозу на тиждень зображений на рисунку 3.26.

Рисунок 3.26 – Приклад тижневого прогнозу

На рисунку 3.27 зображений список міст для вибору.

Рисунок 3.27 – Вибір міста для прогнозу

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи було створено мобільний додаток для відстеження погодних показників, а також окремий додаток для адміністрування погодних даних.

Під час дослідження предметної області була визначена актуальність розробки і проведений аналіз існуючих аналогів. На основі цього була сформульована специфікація функціональних вимог до додатку.

Було поставлено мету роботи і вибрано необхідні інструменти і технології для реалізації мобільного додатку. Головною метою дослідження було забезпечити користувачам достовірну погодну інформацію, щоб вони могли зручно планувати свої дії і адаптуватися до погодних умов.

Були проведені моделювання бізнес-процесів мобільного додатку, які були представлені у вигляді діаграм бізнес-процесів на контекстному та першому рівні декомпозиції, варіантів використання, діаграм послідовності та компонентів.

Реалізація мобільного додатка була здійснена з використанням мови програмування Kotlin в середовищі розробки Android Studio. На початковому етапі була описана архітектура додатка. Збереження та оновлення погодної інформації здійснюється за допомогою підключення додатка до нереляційної бази даних Firebase Realtime Database. З метою відокремлення доступу до даних та створення зручного інтерфейсу для редагування погодних даних адміністратором було створено окремий додаток з модулем авторизації для адміністраторів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Майбутнє розробки мобільних додатків [Електронний ресурс] – Режим доступу до ресурсу: URL: https://web-promo.ua/ua/blog/budushhee-razrabotki-mobilnyh-prilozhenij-9-tendenczij-o-kotoryh-nuzhno-znat-razrabotchiku/#statistika-rinku-mobilnih-dodatkiv (дата звернення: 20.02.23)

2. Тренди мобільної розробки на 2023 рік [Електронний ресурс] – Режим доступу до ресурсу: URL: https://wezom.com.ua/ua/blog/trendi-mobilnoyi-rozrobki-na-2023-rik (дата звернення: 20.02.23)

3. ВООЗ скасувала надзвичайний стан через COVID-19 [Електронний ресурс] – Режим доступу до ресурсу: URL: https://coronavirus.rbc.ua/rus/news/zaginuli-ponad-20-mln-lyudey-vooz-skasuvav-1683294016.html (дата звернення 09.05.23)

4. MVC(Model-View-Controller) Architecture Pattern in Android with Example: [Електронний ресурс] – Режим доступу до ресурсу: URL: https://www.geeksforgeeks.org/mvc-model-view-controller-architecture-pattern-in-android-with-example/ (Дата звернення: 25.02.23)

5. MVP Android Kotlin Architecture [Електронний ресурс] – Режим доступу до ресурсу: URL: https://www.manektech.com/blog/mvp-android-kotlin-architecture (Дата звернення: 02.03.23)

6. MVVM (Model View ViewModel) Architecture Pattern in Android [Електронний ресурс] – Режим доступу до ресурсу: URL: https://www.codingninjas.com/codestudio/library/android-mvvm-model-view-viewmodel-architecture (Дата звернення: 10.03.23)

7. Kotlin Clean Architecture [Електронний ресурс] – Режим доступу до ресурсу: URL: https://proandroiddev.com/kotlin-clean-architecture-1ad42fcd97fa (Дата звернення: 12.03.23)

8. Веб ресурс мобільного додатка «Weather Underground» [Електронний ресурс] – Режим доступу до ресурсу: URL: https://www.wunderground.com (Дата звернення: 18.03.23)

9. Мобільний додаток «Daily Weather» [Електронний ресурс] – Режим доступу до ресурсу: URL:https://play.google.com/store/apps/details?id=com.rlk.weathers&hl=en_US(Дата звернення: 18.03.23)

10. Мобільний додаток «WeatherBug» [Електронний ресурс] – Режим доступу до ресурсу: URL: https://play.google.com/store/apps/details?id=com.aws.android&hl=en_US (Дата звернення: 20.03.23)

11. Тренди в дизайні мобільних додатків [Електронний ресурс] – Режим доступу до ресурсу: URL: https://media.lifta.space/ua/media/stock/trendy-v-dizajne-mobilnyh-prilozhenij (Дата звернення: 13.03.23)

12. Як створюється інтерфейс мобільних додатків [Електронний ресурс] – Режим доступу до ресурсу: URL: https://wezom.com.ua/ua/blog/dizajn-interfejsov-mobilnyh-prilozhenij (Дата звернення: 15.03.23)

13. Effectiveness of Kotlin vs. Java in android app development tasks [Електронний ресурс] – Режим доступу до ресурсу: URL: https://www.sciencedirect.com/science/article/abs/pii/S0950584920301439#preview-section-references (Дата звернення: 23.03.23)

14. Firebase Realtime Database. Documentation [Електронний ресурс] – Режим доступу до ресурсу: URL: https://firebase.google.com/docs/database (Дата звернення: 02.04.23)

15. Neil Smyth. Android Studio 4.2 Development Essentials - Kotlin Edition [Електронний ресурс] – Режим доступу до ресурсу: URL: https://books.google.com.ua/books?hl=ru&lr&id=2PgvEAAAQBAJ&oi=fnd&pg=PT24&dq=android+studio&ots=uUvKN3I9a2&sig=ht5eaZ_Lp-iayndwbA0uHyGgLC4&redir_esc=y#v=onepage&q&f=false (Дата звернення: 04.02.23)

16. Відомості про Adobe Photoshop [Електронний ресурс] – Режим доступу до ресурсу: URL: https://www.britannica.com/technology/Adobe-Photoshop (Дата звернення: 06.02.23)

17. Top 10 Advantages of Firebase [Електронний ресурс] – Режим доступу до ресурсу: URL: https://blog.back4app.com/advantages-of-firebase/ (Дата звернення: 07.02.23)

18. Відомості про WBS діаграму[Електронний ресурс] – Режим доступу до ресурсу: URL: https://asana.com/resources/work-breakdown-structure (Дата звернення: 05.02.23)

19. Відомості про OBS діаграму [Електронний ресурс] – Режим доступу до ресурсу: URL: https://smallbusiness.chron.com/implications-organizational-culture-project-structure-73039.html (Дата звернення: 08.02.23)

20. Відомості про діаграму Ґанта [Електронний ресурс] – Режим доступу до ресурсу: URL: https://www.atlassian.com/agile/project-management/gantt-chart (Дата звернення: 09.02.23)

ДОДАТОК А

ТЕХНІЧНЕ ЗАВДАННЯ

на розробку інформаційної системи
«Мобільний додаток відстеження погодних показників»

ПОГОДЖЕНО:

Доцент кафедри комп’ютерних наук

_____ Неня В.Г.

Студент групи ІТ-91

_____ Кущ Б.С.

Суми 2023

1. ПРИЗНАЧЕННЯ Й МЕТА СТВОРЕННЯ МОБІЛЬНОГО ДОДАТКУ

1.1 Призначення розробки

Мобільний додаток призначений для отримання користувачами можливості перегляду прогнозу та відстеження поточних погодних умов (температури, швидкості вітру, кількості опадів).

1.2 Мета створення мобільного додатку

Метою створення мобільного додатку є надання користувачам достовірної інформації про погоду, щоб вони могли зручно планувати свої дії та адаптуватися до погодних умов.

1.3 Цільова аудиторія

До цільової аудиторії можна віднести наступні групи користувачів:

- Фізично активні люди, які планують свої заняття відповідно до погодних умов.
- Люди, які працюють на вулиці та залежать від погодних умов, наприклад, будівельники, садівники та рибалки.
- Звичайні користувачі, які хочуть мати доступ до актуальних погодних даних.

2 ВИМОГИ ДО МОБІЛЬНОГО ДОДАТКУ

2.1 Вимоги до мобільного додатку в цілому

2.1.1 Вимоги до структури й функціонування мобільного додатку

Кінцевий продукт даного проекту має бути представлений мобільним додатком, який коректно функціонує з мобільними пристроями на базі ОС Android.

2.1.2 Вимоги до персоналу

Адміністратори мобільного додатку не повинні мати особливі технічні навички для роботи з додатком. Але вини повинні регулярно оновлювати погоду відносно кожного представленого міста.

2.1.3 Вимоги збереження інформації

Уся погодна інформація, надана у додатку, повинна зберігатися в базі даних Firebase Realtime Database.

2.1.4 Вимоги до розмежування доступу

Розроблюваний мобільний додаток має бути доступним для використання на мобільних пристроях через Інтернет. Права доступу до інформації в додатку розподіляються на дві групи користувачів: адміністратор та користувач.

Користувач має доступ до перегляду погодної інформації.

Адміністратор має доступ до редагування інформації відповідно з додатку адміністратора після авторизації.

2.2 Структура мобільного додатку

2.2.1 Загальна інформація про структуру мобільного додатку

До структури мобільного додатку входять усі його компоненти, які є загальнодоступними для користувачів. На головному екрані додатку, користувач може побачити основні погодні показники відповідно до обраної локації. Головний екран також містить:

- Кнопка вибору міста (відповідає за відображення погодних показників в вибраному місті)
- Вкладка з інформаційними блоками погодинного прогнозу (показники погоди на день відповідно з 0:00 по 23:00, на кожну годину)
- Вкладка з інформаційними блоками прогнозу на наступний тиждень

2.2.2 Дизайн та структура мобільного додатку

Дизайн мобільного додатку повинен мати сучасний та мінімалістичний стиль, зручні для перегляду види та розміри шрифтів, а також логічно розташовані інформаційні блоки, графічні елементи та інші об'єкти.

Дизайну майбутнього мобільного додатку зображено на рисунку А.1.

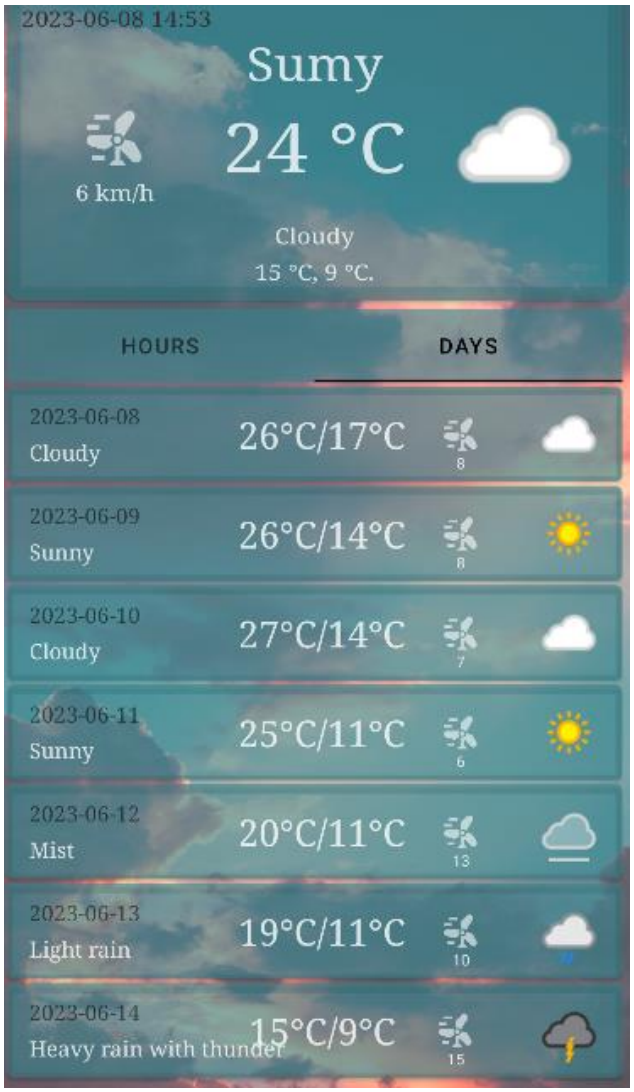

Рисунок А.1 – Дизайну мобільно додатку

2.3 Вимоги до функціонування додатку

Розроблений мобільний додаток повинен задовольняти такі функціональні вимоги:

- Перегляд поточних погодних даних (наприклад: температура, опади та швидкість вітру) за вибраним містом;
- Оновлення погодної інформації;
- Формування погодинного/тижневого прогнозу погоди.

2.4 Вимоги до видів забезпечення

2.3.1 Вимоги до інформаційного забезпечення

Реалізація мобільного додатку відбувається із використанням:

- Android Studio
- Kotlin
- Adobe Photoshop[16]
- Realtime Database[17]

2.4.2 Вимоги до лінгвістичного забезпечення

Мобільний додаток буде містити один лінгвістичний пакет для локалізації інтерфейсу – англійська мова.

2.4.3 Вимоги до програмного забезпечення

Програмне забезпечення користувач повинне мати: Android 8.0 або вище.

3 СКЛАД І ЗМІСТ РОБІТ ЗІ СТВОРЕННЯ МОБІЛЬНОГО ДОДАТКУ

Детальний опис етапів створення мобільного додатку наведено в таблиці А.1.

Таблиця А.1. – Етапи створення мобільного додатку

№	Склад і зміст робіт	Строк розробки (у робочих днях)
1	Постановка задачі проекту	9 днів
2	Проектування структури додатка	12 днів
3	Розробка дизайну додатка	7 днів
4	Розробка бази даних	30 днів
5	Розробка функціональності	10 днів
6	Оформлення документації	5 днів
	Загальна тривалість робіт	73 дні

4 ВИМОГИ ДО СКЛАДУ Й ЗМІСТУ РОБІТ ІЗ ВВЕДЕННЯ МОБІЛЬНОГО ДОДАТКУ В ЕКСПЛУАТАЦІЮ

Для введення мобільного додатку в експлуатацію необхідно:

- Врахувати специфіку платформи, на якій буде запущений додаток.
- Забезпечити підтримку додатку протягом тривалого часу та виконувати регулярні оновлення для удосконалення продукту та вирішення виявлених помилок.
- Забезпечити комунікацію з користувачами та швидке реагування на їх запити та проблеми.

ДОДАТОК Б

ПЛАНУВАННЯ РОБІТ

Деталізація мети проекту методом SMART. Продуктом дипломного проекту є мобільний додаток для відстеження погодних показників.

Мета проекту полягає у створенні додатку, який допоможе користувачам отримувати актуальну та точну погодну інформацію.

За допомогою мобільного додатку, користувачі зможуть переглядати погодні умови згідно з вибраним містом. Це дозволить їм забезпечити комфорт та безпеку під час різних активностей, таких як подорожі, прогулянки та спортивні заходи. Крім того, додаток допоможе планувати події заздалегідь, визначати оптимальний одяг та взуття, а також приймати розумні рішення стосовно проведення зовнішньої діяльності.

Результат деталізації методом SMART розміщено у таблиці Б.1.

Таблиця Б.1 – Деталізація мети проекту методом SMART

Specific (конкретна)	Створити додаток для відстеження погодних показників, для отримання актуальної погодної інформації.
Measurable (вимірювана)	Результатом роботи проекту є випуск додатку та 1000 завантажень за перший місяць.
Achievable (досяжна)	Реалізація проекту здійснюється за допомогою середовища розробки Anroid Studio, з використанням Weather API для отримання погодних показників.

Продовження таблиці Б.1 – Деталізація мети проекту методом SMART

Relevant (реалістична)	В наявності є всі необхідні технічні та програмні засоби. Розробники мають достатню кваліфікацію для виконання поставлених завдань.
Time-framed (обмежена в часі)	Є обмеження в часі для досягнення цілі. Робота повинна бути завершена відповідно до узгоджених термінів замовником проекту. Проект має бути виконаний згідно з календарним планом.

Для планування структури робіт використовується WBS діаграма[18], яка графічно відображає згруповані елементи проекту в формі пакетів робіт, які мають ієрархічний зв'язок з продуктом проекту. Побудуємо WBS структуру, детально описавши роботи, які потрібно виконати на кожному етапі створення проекту. Також проведемо декомпозицію робіт для даного проекту, яка відображена на рисунку Б.1.

Після побудови WBS ми розробимо структуру організації для впровадження готового проекту (OBS)[19]. Організаційна структура проекту стосується внутрішньої організаційної структури проекту і не враховує відносин між проектними групами або учасниками та батьківськими організаціями. Діаграма OBS представлена на рисунку Б.2. Список виконавців, які беруть участь в проекті, наведено в таблиці Б.2.

Таблиця Б.2 – Виконавці проекту

Роль	**Ім'я**	**Проектна роль**
Розробник	Кущ Б.С.	Виконує розробку основного функціоналу проекту.

Продовження таблиці Б.2 – Виконавці проекту

Проектувальник	Кущ Б.С.	Визначає функціонал системи та інструменти, для її реалізації. Розробляє макет інтерфейсу користувача
Тестувальник	Кущ Б.С.	Відповідає за тестування функціоналу та дизайну мобільного додатку.
Керівник проекту	Неня В.Г.	Формує постановку задачі проекту.
Менеджер проекту	Кущ Б.С.	Відповідає за виконання термінів, розподіл ресурсів та завдань між учасниками. Виконує збір та аналіз даних.

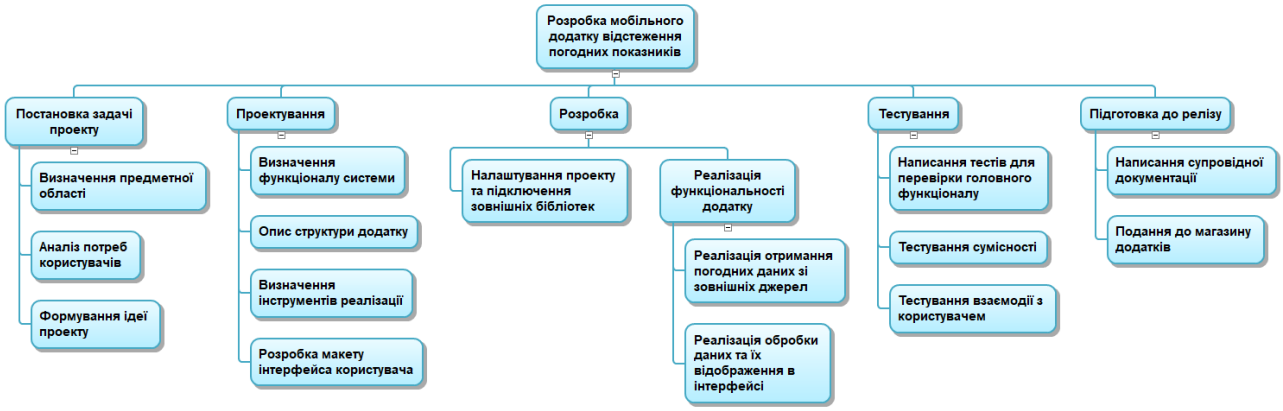

Рисунок Б.1 – WBS. Структура робіт проекту

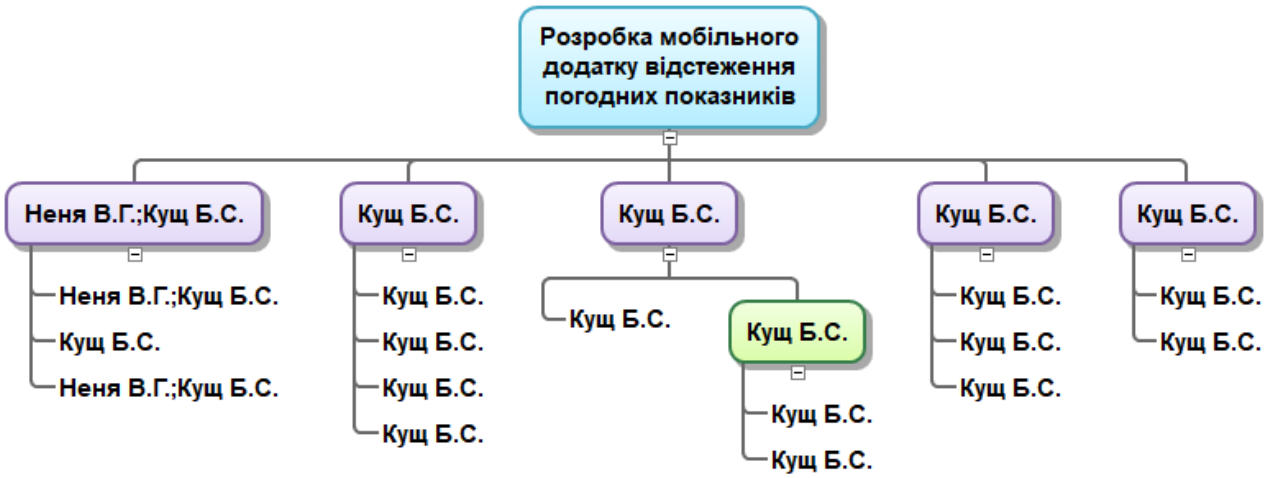

Рисунок Б.2 – Організаційна структура проекту (OBS)

Створено календарний план для виконання дипломного проекту. Діаграма Ґанта є найпоширенішим форматом графіка в будь-якій галузі[20]. Вона надає можливість менеджерам проекту та всій команді розробників візуалізувати графіки часу та зв'язок між окремими завданнями та етапами проекту. Тривалість робіт вказана у днях, але фактична тривалість виконання робіт становить приблизно 2-3 години на день. Щоб мати реалістичне уявлення про тривалість виконання робіт з урахуванням обмеження використання ресурсів, вихідних та святкових днів, побудовано календарний графік. Діаграма Ґанта та список робіт зображені на рисунку Б.3.

Розробка мобільного додатку відстеження погодних показників"	**73 days**	**Mon 20.02.23**	**Tue 30.05.23**	
Постановка задачі проекту	**9 days**	**Mon 20.02.23**	**Thu 02.03.23**	
Визначення предметної області	2 days	Mon 20.02.23	Tue 21.02.23	
Аналіз потреб користувачів	3 days	Wed 22.02.23	Fri 24.02.23	3
Формування ідеї проекту	4 days	Mon 27.02.23	Thu 02.03.23	4
Проектування	**15 days**	**Fri 03.03.23**	**Thu 23.03.23**	**2**
Визначення функціоналу системи	3 days	Fri 03.03.23	Tue 07.03.23	
Опис структури додатку	10 days	Wed 08.03.23	Tue 21.03.23	7
Визначення інструментів реалізації	2 days	Wed 22.03.23	Thu 23.03.23	8
Розробка	**39 days**	**Fri 24.03.23**	**Tue 16.05.23**	**6**
Реалізація додатку адміністратора	**19 days**	**Fri 24.03.23**	**Tue 18.04.23**	
Розробка макетів інтерфейса	6 days	Fri 24.03.23	Fri 31.03.23	9
Реалізація модулю авторизації	4 days	Sat 01.04.23	Wed 05.04.23	12
Реалізація дадання даних до бази	9 days	Thu 06.04.23	Tue 18.04.23	13
Реалізація додатку користувача	**20 days**	**Wed 19.04.23**	**Tue 16.05.23**	**11**
Розробка макетів інтерфейса	5 days	Wed 19.04.23	Tue 25.04.23	14
Реалізація отримання погодних даних з бази даних	5 days	Wed 26.04.23	Tue 02.05.23	16
Реалізація обробки даних та їх відображення в інтерфейсі	10 days	Wed 03.05.23	Tue 16.05.23	17
Тестування	**4 days**	**Wed 17.05.23**	**Mon 22.05.23**	**10**
Тестування сумісності	2 days	Wed 17.05.23	Thu 18.05.23	
Тестування взаємодії з користувачем	2 days	Fri 19.05.23	Mon 22.05.23	20
Підготовка до релізу	**6 days**	**Tue 23.05.23**	**Tue 30.05.23**	**19**
Написання супровідної документації	6 days	Tue 23.05.23	Tue 30.05.23	

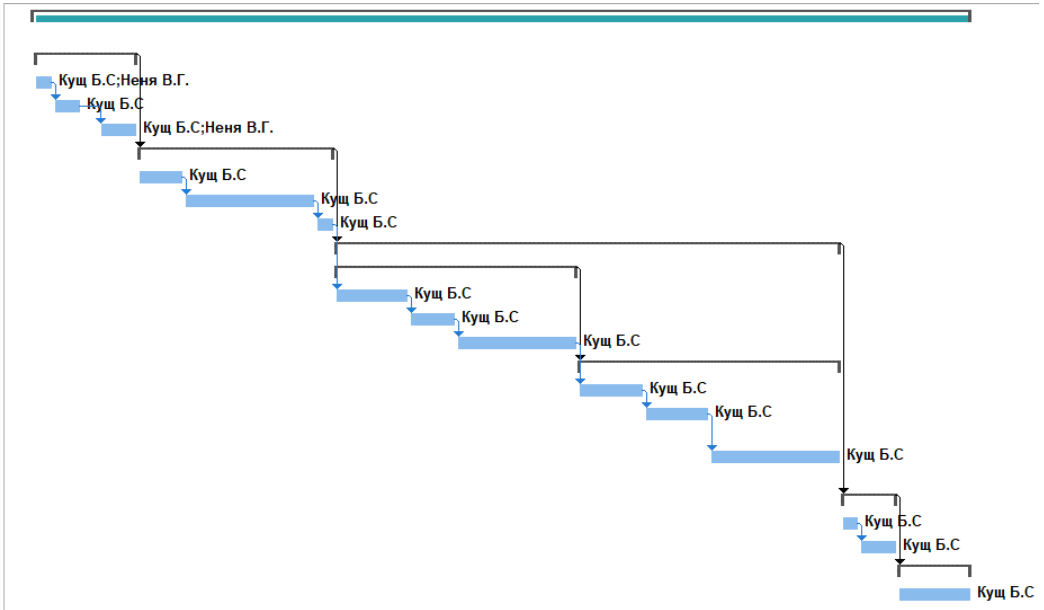

Рисунок Б.3 – Список робіт та Діаграма Ґанта

Проведено аналіз ризиків, який включає якісну і кількісну оцінки. Під час якісної оцінки визначено ризики, які потребують негайного реагування, зазначено їх важливість і був обраний відповідний метод реагування. Кількісна оцінка ризиків здійснилася для більш детального виявлення ризиків та оцінки їх впливу на проект. Кількісну і якісну оцінки можна використовувати окремо або в поєднанні, залежно від доступного часу і бюджету, а також потреби в кількісному або якісному аналізі ризиків. В таблиці Б.5 наведено класифікацію ризиків за показниками ймовірності виникнення та величиною втрат.

Після цього було розроблено план реагування на ризики, який включатиме розробку методів і технологій для зменшення негативного впливу ризиків на проект. Визначено ефективність запланованих заходів та оцінено, чи можуть наслідки впливу ризику на проект бути позитивними або негативними. Ризики оцінювались за показниками, перерахованими в таблиці Б.3. На основі цих оцінок побудована матриця ймовірності виникнення ризиків та їх впливу, що зображена в таблиці Б.4.

Таблиця Б.3 – Шкала оцінювання ймовірності виникнення та впливу ризику на виконання проекту

Оцінка	Ймовірність виникнення	Вплив ризику
1	Низька	Низький
2	Середня	Середній
3	Висока	Високий

Таблиця Б.4 – Матриця ймовірності виникнення ризиків та впливу ризику

Ймовірність виникнення			
3		RS_14	
2		RS_2, RS_3, RS_7, RS_9, RS_11	RS_6, RS_13
1	RS_8	RS_1, RS_5, RS_15,	RS_4, RS_10, RS_12
	1	2	3

Вплив ризику

- зелений колір – прийнятні ризики;
- жовтий колір – виправданні ризики;
- червоний колір – недопустимі ризики.

На підставі отриманого значення індексу ризики класифікують: за рівнем ризику, що знаходиться в табл. Б.5.

Таблиця Б.5 – Шкала оцінювання за рівнем ризику

№	Назва	Межі	Ризики, що входять (номери)
1	Прийнятні	$1 \leq R \leq 2$	1,5,8,15
2	Виправдані	$3 \leq R \leq 4$	2,3,4,7,9,10,11,12
3	Недопустимі	$6 \leq R \leq 9$	6,13,14

Таблиця Б.6 – Оцінка ймовірностей виникнення, величини втрат та індексу ризику

ID	Статус ризику	Опис ризику	Ймовірність ризику	Вплив ризику	Ранг ризику	План А	Тип стратегії реагування	План Б
RS_1	Відкритий	Непорозуміння між розробниками та замовником	Низька	Середній	2	Налагодження відносин між розробниками та замовником	Попередження	Створення здорової атмосфери в колективі
RS_2	Відкритий	Поява продуктів-конкурентів	Середня	Середній	4	Постійне вдосконалення додатку, впровадження нових функції	Прийняття	
RS_3	Відкритий	Несумісність з різними пристроями	Середня	Середній	4	Адаптація продукту під інші пристрої	Прийняття	Проведення тестування на різних пристроях
RS_4	Відкритий	Низька кваліфікація розробників	Низька	Високий	3	Підвищити кваліфікацію персоналу	Пом’якшення	Надати посилання на ресурси для отримання знань

Продовження таблиці Б.6

ID	Статус ризику	Опис ризику	Ймовірність ризику	Вплив ризику	Ранг ризику	План А	Тип стратегії реагування	План Б
RS_5	Відкритий	Недостатня продуманість архітектури	Низька	Середній	2	Ретельне планування архітектури додатку перед початком розробки	Попередження	
RS_6	Відкритий	Помилки в програмному коді	Середня	Високий	6	Використання тестування/ревізії коду для виявлення та виправлення помилок	Попередження	
RS_7	Відкритий	Внесення змін в ТЗ замовником	Середня	Середній	4	Перед початком проекту узгодити з замовником процедури зміни ТЗ.	Зменшення	
RS_8	Відкритий	Додання зайвого функціоналу	Низька	Низький	1	Визначення чітких критеріїв, щодо функціональності додатку	Зменшення	Узгодження потрібного функціоналу з замовником

Продовження таблиці Б.6

ID	Статус ризику	Опис ризику	Ймовірність ризику	Вплив ризику	Ранг ризику	План А	Тип стратегії реагування	План Б
RS_9	Відкритий	Невиконання моніторингу проекту	Середня	Середній	4	Налагодити проміжний контроль результатів реалізації проекту.	Попередження	Здійснювати моніторинг проекту співробітниками.
RS_10	Відкритий	Втрата даних при розробці	Низька	Високий	3	Резервне копіювання даних	Ухилення	Зберігання даних на хмарних платформах
RS_11	Відкритий	Недостатній розвиток функціональності	Середня	Середній	4	Планування способів розширення функцій на етапі розробки, врахування фідбеку користувачів та ринкові тенденції.	Попередження	Узгодження з замовником можливе розширення функціоналу
RS_12	Відкритий	Відсутність підтримки	Низька	Високий	3	Забезпечити постійну технічну підтримку та вчасні виправлення помилок	Попередження	

Продовження таблиці Б.6

ID	Статус ризику	Опис ризику	Ймовірність ризику	Вплив ризику	Ранг ризику	План А	Тип стратегії реагування	План Б
RS_13	Відкритий	Розбіжності в сприйнятті дизайну	Середня	Високий	6	Провести презентацію прототипів дизайну замовнику, щоб забезпечити згоду перед розробкою.	Зменшення	Враховати вимоги замовника щодо вигляду додатку
RS_14	Відкритий	Відсутність участі замовника	Висока	Середній	6	Використання регулярних зустрічей та презентації для залучення замовника до процесу розробки	Зменшення	Забезпечити структурований зворотний зв'язок з замовником
RS_15	Відкритий	Затримки у графіку	Низька	Середній	2	Розробка реалістичного графіку з урахуванням можливих затримок.	Ухилення	Внесення необхідних змін для виконання проекту в строк

ДОДАТОК В

ЛІСТНИГ КОДУ

Додаток адміністратора

LoginActivity.kt:

```
class LoginActivity : AppCompatActivity() {
    private lateinit var binding: ActivityLoginBinding
    private lateinit var firebaseAuth: FirebaseAuth

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityLoginBinding.inflate(layoutInflater)
        setContentView(binding.root)

        firebaseAuth = FirebaseAuth.getInstance()

        binding.buttonLogin.setOnClickListener {
            val email = binding.editTextEmail.text.toString()
            val password = binding.editTextPassword.text.toString()

            if (email.isNotEmpty() && password.isNotEmpty()) {
                firebaseAuth.signInWithEmailAndPassword(email, password)
                    .addOnCompleteListener(this) { task ->
                        if (task.isSuccessful) {
                            // Вдала авторизація
                            Toast.makeText(
                                this@LoginActivity,
                                "Login successful!",
                                Toast.LENGTH_SHORT
                            ).show()

                            startActivity(Intent(this, AdminActivity::class.java))
                        } else {
                            // Невдалося авторизуватися
                            Toast.makeText(
                                this@LoginActivity,
                                "Login failed. Please try again.",
                                Toast.LENGTH_SHORT
                            ).show()
                        }
                    }
            } else {
                Toast.makeText(
                    this@LoginActivity,
                    "Please enter email and password.",
                    Toast.LENGTH_SHORT
```

```
                ).show()
            }
        }
    }
}
```

AdminActivity.kt:

```
class AdminActivity : AppCompatActivity() {
    private lateinit var binding: ActivityAdminBinding
    private lateinit var database: DatabaseReference

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityAdminBinding.inflate(layoutInflater)
        setContentView(binding.root)

        database = FirebaseDatabase.getInstance("https://weatheradminapp-70b4d-
default-rtdb." +
                "europe-west1.firebasedatabase.app").getReference("weather")
        FirebaseDatabase.getInstance().setPersistenceEnabled(true)

        val cities = arrayOf("Kyiv", "Sumy", "Kharkiv")

        val adapter = ArrayAdapter(this, R.layout.simple_spinner_item, cities)
        adapter.setDropDownViewResource(R.layout.simple_spinner_dropdown_item)
        binding.citySpinner.adapter = adapter

        binding.updateCurrentWeatherButton.setOnClickListener {
            val intent = Intent(this, UpdateCurrentWeatherActivity::class.java)
            intent.putExtra("city", binding.citySpinner.selectedItem.toString())
            startActivity(intent)
        }
        binding.updateDayWeatherButton.setOnClickListener {
            val intent = Intent(this, UpdateDayWeatherActivity::class.java)
            intent.putExtra("city", binding.citySpinner.selectedItem.toString())
            startActivity(intent)
        }
        binding.updateHourWeatherButton.setOnClickListener {
            val intent = Intent(this, UpdateHourWeatherActivity::class.java)
            intent.putExtra("city", binding.citySpinner.selectedItem.toString())
            startActivity(intent)
        }
    }
}
```

UpdateCurrentWeatherActivity.kt:

```
class UpdateCurrentWeatherActivity : AppCompatActivity() {
    private lateinit var binding: ActivityUpdateCurrentWeatherBinding
    private lateinit var database: DatabaseReference
    private lateinit var city: String

    override fun onCreate(savedInstanceState: Bundle?) {
```

```
        super.onCreate(savedInstanceState)
        binding = ActivityUpdateCurrentWeatherBinding.inflate(layoutInflater)
        setContentView(binding.root)

        city = intent.getStringExtra("city") ?: ""
        database = FirebaseDatabase.getInstance(
            "https://weatheradminapp-70b4d-default-rtdb." +
                    "europe-west1.firebasedatabase.app"
        ).getReference("weather").child(city).child("current")

        binding.btnUpdateCurrentWeather.setOnClickListener {
            updateCurrentWeather()
        }
    }

    private fun updateCurrentWeather() {
        val condition = binding.editCondition.text.toString().trim()
        val temperature = binding.editTemperature.text.toString().toDoubleOrNull()
        val windSpeed = binding.editWindSpeed.text.toString().toDoubleOrNull()

        if (condition.isNotEmpty() && temperature != null && windSpeed != null) {
            val currentWeather = hashMapOf<String, Any>()
            currentWeather["condition"] = condition
            currentWeather["last_updated"] = getCurrentDateTime()
            currentWeather["temp_c"] = temperature
            currentWeather["wind_kph"] = windSpeed

            database.setValue(currentWeather)
                .addOnSuccessListener {
                    Toast.makeText(this, "Weather updated successfully",
Toast.LENGTH_SHORT).show()
                    finish()
                }
                .addOnFailureListener {
                    Toast.makeText(this, "Failed to update weather",
Toast.LENGTH_SHORT).show()
                }
        } else {
            Toast.makeText(this, "Please fill in all the fields",
Toast.LENGTH_SHORT).show()
        }
    }

    private fun getCurrentDateTime(): String {
        val dateFormat = SimpleDateFormat("yyyy-MM-dd HH:mm", Locale.getDefault())
        val cal = Calendar.getInstance()
        return dateFormat.format(cal.time)
    }
}
```

UpdateDayWeatherActivity.kt:

```
class UpdateDayWeatherActivity : AppCompatActivity() {
    private lateinit var database: DatabaseReference
    private lateinit var binding: ActivityUpdateDayWeatherBinding
```

```
    private lateinit var selectedDate: String
    private lateinit var city: String

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityUpdateDayWeatherBinding.inflate(layoutInflater)
        setContentView(binding.root)

        city = intent.getStringExtra("city") ?: ""
        database = FirebaseDatabase.getInstance(
            "https://weatheradminapp-70b4d-default-rtdb." +
                    "europe-west1.firebasedatabase.app"
        ).reference

        selectedDate = intent.getStringExtra("date") ?: ""

        val days = generateDaysList()

        val adapter = ArrayAdapter(this, android.R.layout.simple_spinner_item, days)

adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item)
        binding.spinnerDays.adapter = adapter

        val selectedDayIndex = days.indexOf(selectedDate)
        binding.spinnerDays.setSelection(selectedDayIndex)

        binding.spinnerDays.onItemSelectedListener = object :
AdapterView.OnItemSelectedListener {
            override fun onItemSelected(
                parent: AdapterView<*>?,
                view: View?,
                position: Int,
                id: Long
            ) {
                selectedDate = parent?.getItemAtPosition(position).toString()
            }

            override fun onNothingSelected(parent: AdapterView<*>?) {
            }
        }

        // Обробка натискання кнопки "Оновити погоду"
        binding.btnUpdateWeather.setOnClickListener {
            val date = selectedDate
            val condition = binding.etCondition.text.toString()
            val maxTemp = binding.etMaxTemp.text.toString().toDoubleOrNull()
            val minTemp = binding.etMinTemp.text.toString().toDoubleOrNull()
            val maxWind = binding.etMaxWind.text.toString().toDoubleOrNull()

            if (validateInput(date, condition, maxTemp, minTemp, maxWind)) {
                updateDayWeather(date, condition, maxTemp!!, minTemp!!, maxWind!!)
            } else {
                Toast.makeText(this, "Please enter correct data",
Toast.LENGTH_SHORT).show()
            }
        }
    }
```

```
    private fun generateDaysList(): List<String> {
        val days = ArrayList<String>()
        val calendar = Calendar.getInstance()
        val dateFormat = SimpleDateFormat("yyyy-MM-dd", Locale.getDefault())

        // Додаємо сьогоднішню дату до списку
        val currentDate = dateFormat.format(calendar.time)
        days.add(currentDate)

        // Генеруємо наступні 6 днів, починаючи з наступного дня
        for (i in 1..6) {
            calendar.add(Calendar.DAY_OF_MONTH, 1)
            val date = dateFormat.format(calendar.time)
            days.add(date)
        }

        return days
    }

    private fun validateInput(
        date: String,
        condition: String,
        maxTemp: Double?,
        minTemp: Double?,
        maxWind: Double?
    ): Boolean {
        return date.isNotEmpty() && condition.isNotEmpty() && maxTemp != null &&
minTemp != null && maxWind != null
    }

    private fun updateDayWeather(
        date: String,
        condition: String,
        maxTemp: Double,
        minTemp: Double,
        maxWind: Double
    ) {
        val dayWeather = hashMapOf<String, Any>()
        dayWeather["date"] = date
        dayWeather["condition"] = condition
        dayWeather["maxtemp_c"] = maxTemp
        dayWeather["mintemp_c"] = minTemp
        dayWeather["maxwind_kph"] = maxWind

        // Оновлення даних в Firebase

database.child("weather").child(city).child("forecast").child("daily").child(date)
            .setValue(dayWeather)
            .addOnSuccessListener {
                Toast.makeText(this, "Weather updated successfully",
Toast.LENGTH_SHORT).show()
                finish()
            }
            .addOnFailureListener {
                Toast.makeText(this, "Failed to update weather",
Toast.LENGTH_SHORT).show()
```

```
            }
    }
}
```

UpdateHourWeatherActivity.kt:

```
class UpdateHourWeatherActivity : AppCompatActivity() {
    private lateinit var database: DatabaseReference
    private lateinit var binding: ActivityUpdateHourWeatherBinding
    private lateinit var city: String

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityUpdateHourWeatherBinding.inflate(layoutInflater)
        setContentView(binding.root)
        city = intent.getStringExtra("city") ?: ""

        database = FirebaseDatabase.getInstance(
            "https://weatheradminapp-70b4d-default-rtdb." +
                    "europe-west1.firebasedatabase.app"
        ).reference

        val hourlyForecastRef = database.child("weather")
            .child(city).child("forecast").child("hourly")

        // Налаштування NumberPicker для вибору години дня
        binding.hourPicker.minValue = 0
        binding.hourPicker.maxValue = 23
        binding.hourPicker.setFormatter { value ->
            val hourFormat = DecimalFormat("00")
            val hour = hourFormat.format(value)
            "$hour:00"
        }

        // Обробка натискання кнопки "Update"
        binding.btnUpdateHourWeather.setOnClickListener {
            // Отримання введених користувачем даних
            val hour = binding.hourPicker.value
            val condition = binding.conditionEditText.text.toString().trim()
            val temperature =
binding.temperatureEditText.text.toString().toDoubleOrNull()
            val windSpeed =
binding.windSpeedEditText.text.toString().toDoubleOrNull()

            if (condition.isNotEmpty() && temperature != null && windSpeed != null) {
                // Формування часу в форматі "yyyy-MM-dd HH:00"
                val calendar = Calendar.getInstance()
                calendar.set(Calendar.HOUR_OF_DAY, hour)
                val time =
                    SimpleDateFormat("yyyy-MM-dd HH:00",
Locale.getDefault()).format(calendar.time)

                // Створення об'єкта з даними про погоду
                val hourlyForecast = hashMapOf<String, Any>()
                hourlyForecast["time"] = time
```

```
                hourlyForecast["condition"] = condition
                hourlyForecast["temp_c"] = temperature
                hourlyForecast["wind_kph"] = windSpeed

                // Оновлення даних в Firebase Realtime Database
                hourlyForecastRef.child(time).setValue(hourlyForecast)
                    .addOnCompleteListener { task ->
                        if (task.isSuccessful) {
                            Toast.makeText(this, "Weather updated successfully",
Toast.LENGTH_SHORT)
                                .show()
                        } else {
                            Toast.makeText(this, "Failed to update weather",
Toast.LENGTH_SHORT)
                                .show()
                        }
                    }
            } else {
                Toast.makeText(this, "Please enter correct data",
Toast.LENGTH_SHORT).show()
            }
        }
    }
}
```

activity_login.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical">

    <EditText
        android:id="@+id/editTextEmail"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginEnd="16dp"
        android:hint="Email" />

    <EditText
        android:id="@+id/editTextPassword"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginEnd="16dp"
        android:hint="Password"
        android:inputType="textPassword" />


    <Button
        android:id="@+id/buttonLogin"
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:text="Login" />


</LinearLayout>
```

activity_admin.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    tools:context=".AdminActivity">

    <TextView
        android:id="@+id/titleTextView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Admin Panel"
        android:textSize="24sp"
        android:textStyle="bold" />

    <TextView
        android:id="@+id/cityLabel"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/titleTextView"
        android:layout_marginTop="16dp"
        android:text="Select City:"
        android:textSize="18sp" />

    <Spinner
        android:id="@+id/citySpinner"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/cityLabel"
        android:layout_marginTop="8dp" />

    <Button
        android:id="@+id/updateCurrentWeatherButton"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/citySpinner"
        android:layout_marginTop="16dp"
        android:text="Update Current Weather" />

    <Button
        android:id="@+id/updateDayWeatherButton"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/updateCurrentWeatherButton"
        android:layout_marginTop="8dp"
        android:text="Update Daily Weather" />
```

```
    <Button
        android:id="@+id/updateHourWeatherButton"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/updateDayWeatherButton"
        android:layout_marginTop="8dp"
        android:text="Update Hourly Weather" />

</RelativeLayout>
```

activity_update_current_weather.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">

    <TextView
        android:id="@+id/text_title"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Update Current Weather"
        android:textSize="18sp"
        android:textStyle="bold" />

    <EditText
        android:id="@+id/edit_condition"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Condition" />

    <EditText
        android:id="@+id/edit_temperature"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Temperature (Celsius)" />

    <EditText
        android:id="@+id/edit_wind_speed"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Wind Speed (km/h)" />

    <Button
        android:id="@+id/btn_update_current_weather"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Update Weather" />
</LinearLayout>
```

activity_update_day_weather.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp"
    tools:context=".UpdateDayWeatherActivity">

    <TextView
        android:id="@+id/text_title"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Update Day Weather"
        android:textSize="18sp"
        android:textStyle="bold" />

    <TextView
        android:id="@+id/tvSelectedDate"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:textAppearanceMedium"
        android:text="Selected Date: "
        android:layout_marginBottom="16dp" />

    <Spinner
        android:id="@+id/spinnerDays"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="16dp" />

    <EditText
        android:id="@+id/etCondition"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Condition" />

    <EditText
        android:id="@+id/etMaxTemp"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Max Temperature (Celsius)" />

    <EditText
        android:id="@+id/etMinTemp"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Min Temperature (Celsius)" />

    <EditText
        android:id="@+id/etMaxWind"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Max Wind Speed (km/h)" />
```

```
    <Button
        android:id="@+id/btnUpdateWeather"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:text="Update Weather" />

</LinearLayout>
```

activity_update_hour_weather.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp"
    tools:context=".UpdateHourWeatherActivity">

    <TextView
        android:id="@+id/text_title"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="5dp"
        android:text="Update Hour Weather"
        android:textSize="18sp"
        android:textStyle="bold" />

    <TextView
        android:id="@+id/tvSelectedDate"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="5dp"
        android:text="Selected Hour:"
        android:textAppearance="?android:textAppearanceMedium" />

    <!-- NumberPicker для выбора часа -->
    <NumberPicker
        android:id="@+id/hourPicker"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>

    <!-- Поле ввода состояния погоды -->
    <EditText
        android:id="@+id/conditionEditText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Condition" />

    <!-- Поле ввода температуры -->
    <EditText
        android:id="@+id/temperatureEditText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
```

```
        android:hint="Temperature (Celsius)" />

    <!-- Поле ввода скорости ветра -->
    <EditText
        android:id="@+id/windSpeedEditText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Wind Speed (km/h)" />

    <!-- Кнопка для сохранения обновленных данных -->
    <Button
        android:id="@+id/btnUpdateHourWeather"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:text="Update Weather" />

</LinearLayout>
```

Додаток користувача

MainActivity.kt:

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        supportFragmentManager.beginTransaction()
            .replace(R.id.placeHolder, MainFragment.newInstance())
            .commit()
    }
}
```

WeatherData.kt:

```
data class WeatherData(
    val weather: Map<String, CityWeather>? = null,
    val current: CurrentWeather? = null,
    val forecast: ForecastWeather? = null
)

data class CityWeather(
    val current: CurrentWeather?,
    val forecast: ForecastWeather?
)

data class CurrentWeather(
    val condition: String? = null,
    val last_updated: String? = null,
    val temp_c: Double? = null,
    val wind_kph: Double? = null
)
```

```
data class ForecastWeather(
    val daily: Map<String, DailyWeather>? = null,
    val hourly: Map<String, HourlyWeather>? = null
)


data class DailyWeather(
    val condition: String? = null,
    val date: String? = null,
    val maxtemp_c: Double? = null,
    val maxwind_kph: Double? = null,
    val mintemp_c: Double? = null
)

data class HourlyWeather(
    val condition: String? = null,
    val temp_c: Double? = null,
    val time: String? = null,
    val wind_kph: Double? = null
)
```

MainViewModel.kt:

```
class MainViewModel : ViewModel() {
    val liveDataCurrent = MutableLiveData<WeatherData>()
}
```

ViewPageAdapter.kt:

```
class ViewPageAdapter(
    activity: AppCompatActivity,
    private val fragmentList: List<Fragment>
) : FragmentStateAdapter(activity) {

    override fun getItemCount(): Int {
        return fragmentList.size
    }

    override fun createFragment(position: Int): Fragment {
        return fragmentList[position]
    }
}
```

DaysAdapter.kt:

```
class DaysAdapter : ListAdapter<DailyWeather,
DaysAdapter.ViewHolder>(WeatherDiffCallback()) {
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {
        val binding = ListItemBinding.inflate(LayoutInflater.from(parent.context),
parent, false)
        return ViewHolder(binding)
    }
```

```
    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        val dailyWeather = getItem(position)
        holder.bind(dailyWeather)
    }

    inner class ViewHolder(private val binding: ListItemBinding) :
        RecyclerView.ViewHolder(binding.root) {
        fun bind(dailyWeather: DailyWeather) {
            binding.apply {
                val maxMinTemp =

"${dailyWeather.maxtemp_c?.toInt()}°C/${dailyWeather.mintemp_c?.toInt()}°C"
                textCondition.text = dailyWeather.condition
                textDate.text = dailyWeather.date
                textTemp.text = maxMinTemp
                textWind.text = dailyWeather.maxwind_kph?.toString()

                val iconFileName = getIconFileName(dailyWeather.condition)
                val iconFilePath = "file:///android_asset/$iconFileName"
                binding.imW.load(Uri.parse(iconFilePath))
            }
        }
    }

    class WeatherDiffCallback : DiffUtil.ItemCallback<DailyWeather>() {
        override fun areItemsTheSame(oldItem: DailyWeather, newItem: DailyWeather):
Boolean {
            return oldItem.date == newItem.date
        }

        override fun areContentsTheSame(oldItem: DailyWeather, newItem:
DailyWeather): Boolean {
            return oldItem == newItem
        }
    }

    private fun getIconFileName(condition: String?): String {
        return when (condition) {
            "Sunny" -> "day_1.png"
            "Partly cloudy" -> "day_2.png"
            "Cloudy" -> "day_3.png"
            "Mist" -> "day_4.png"
            "Light rain" -> "day_5.png"
            "Heavy snow" -> "day_6.png"
            "Rain with thunder" -> "day_7.png"
            else -> "loading.png"
        }
    }
}
```

HoursAdapter.kt:

```
class HoursAdapter :
    ListAdapter<HourlyWeather,
```

```
HoursAdapter.HourViewHolder>(HourViewHolder.HourDiffCallback()) {

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): HourViewHolder
{
        val inflater = LayoutInflater.from(parent.context)
        val binding = ListItemBinding.inflate(inflater, parent, false)
        return HourViewHolder(binding)
    }

    override fun onBindViewHolder(holder: HourViewHolder, position: Int) {
        val hourlyWeather = getItem(position)
        holder.bind(hourlyWeather)
    }

    class HourViewHolder(private val binding: ListItemBinding) :
        RecyclerView.ViewHolder(binding.root) {

        fun bind(hourlyWeather: HourlyWeather) {
            binding.apply {
                val temp = "${hourlyWeather.temp_c?.toInt()}°C"
                textCondition.text = hourlyWeather.condition
                textDate.text = hourlyWeather.time
                textTemp.text = temp
                textWind.text = hourlyWeather.wind_kph.toString()

                val iconFileName = hourlyWeather.condition?.let {
                    getIconFileName(
                        it,
                        hourlyWeather.time.toString()
                    )
                }
                val iconFilePath = "file:///android_asset/$iconFileName"
                binding.imW.load(Uri.parse(iconFilePath))
            }
        }


        private fun getIconFileName(condition: String, time: String): String {
            val hourFormat = SimpleDateFormat("HH", Locale.getDefault())
            val hour = hourFormat.parse(time)?.hours ?: 0

            return when (condition) {
                "Sunny" -> "day_1.png"
                "Clear" -> "night_1.png"
                "Partly cloudy" -> if (hour in 6..17) "day_2.png" else "night_2.png"
                "Cloudy" -> if (hour in 6..17) "day_3.png" else "night_3.png"
                "Mist" -> if (hour in 6..17) "day_4.png" else "night_4.png"
                "Light rain" -> if (hour in 6..17) "day_5.png" else "night_5.png"
                "Heavy snow" -> if (hour in 6..17) "day_6.png" else "night_6.png"
                "Rain with thunder" -> if (hour in 6..17) "day_7.png" else
"night_7.png"
                else -> "loading.png"
            }
        }

        class HourDiffCallback : DiffUtil.ItemCallback<HourlyWeather>() {
            override fun areItemsTheSame(oldItem: HourlyWeather, newItem:
```

```
HourlyWeather): Boolean {
                return oldItem.time == newItem.time
            }

            override fun areContentsTheSame(
                oldItem: HourlyWeather,
                newItem: HourlyWeather
            ): Boolean {
                return oldItem == newItem
            }
        }
    }
}
```

MainFragment.kt:

```
class MainFragment : Fragment() {
    private lateinit var binding: FragmentMainBinding
    private val model: MainViewModel by activityViewModels()
    private val database = FirebaseDatabase.getInstance()
    private val weatherDataRef = database.getReference("weather")
    private var selectedCity: String? = null
    private val fList = listOf(
        HoursFragment.newInstance(),
        DaysFragment.newInstance()
    )
    private val tList = listOf(
        "Hours",
        "Days"
    )

    private val cityNames = mutableListOf<String>()

    private val weatherDataListener = object : ValueEventListener {
        @RequiresApi(Build.VERSION_CODES.O)
        override fun onDataChange(dataSnapshot: DataSnapshot) {
            selectedCity?.let { city ->
                val cityWeatherDataSnapshot = dataSnapshot.child(city)
                val cityWeatherData =
cityWeatherDataSnapshot.getValue(WeatherData::class.java)

                Log.d("MyLog", "Received weather data: $cityWeatherData")

                if (cityWeatherData != null) {
                    updateCurrentCardView(cityWeatherData) // Передаем
cityWeatherData в функцию
                    val hourlyWeatherList =
                        cityWeatherData.forecast?.hourly?.values?.toList() ?:
emptyList()
                    val hoursFragment = fList[0] as? HoursFragment
                    hoursFragment?.updateWeatherDataHourly(hourlyWeatherList)

                    val dailyWeatherList =
                        cityWeatherData.forecast?.daily?.values?.toList() ?:
emptyList()
```

```
                val daysFragment = fList[1] as? DaysFragment
                daysFragment?.updateWeatherDataDaily(dailyWeatherList)
            } else {
                Log.d("MyLog", "City weather data is null")
            }
        }
    }

    override fun onCancelled(error: DatabaseError) {
    }
}

override fun onCreateView(
    inflater: LayoutInflater, container: ViewGroup?,
    savedInstanceState: Bundle?
): View {
    binding = FragmentMainBinding.inflate(inflater, container, false)
    return binding.root
}

@RequiresApi(Build.VERSION_CODES.O)
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    init()
    updateCurrentCardView(WeatherData())
    model.liveDataCurrent.observe(viewLifecycleOwner) {
        updateCurrentCardView(WeatherData())
    }
}

private fun init() {
    val adapter = ViewPageAdapter(requireActivity() as AppCompatActivity, fList)
    binding.vp.adapter = adapter

    TabLayoutMediator(binding.tabLayout, binding.vp) { tab, pos ->
        tab.text = tList[pos]
    }.attach()

    initSpinner()
    weatherDataRef.addValueEventListener(weatherDataListener)
}

private fun initSpinner() {
    weatherDataRef.addListenerForSingleValueEvent(object : ValueEventListener {
        override fun onDataChange(dataSnapshot: DataSnapshot) {
            cityNames.clear()
            for (citySnapshot in dataSnapshot.children) {
                val cityName = citySnapshot.key
                cityName?.let { cityNames.add(it) }
            }
            updateSpinnerAdapter()
        }

        override fun onCancelled(error: DatabaseError) {
        }
    })
}
```

```
    private fun updateSpinnerAdapter() {
        val adapter =
            ArrayAdapter(requireContext(), android.R.layout.simple_spinner_item,
cityNames)

adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item)
        binding.citySpinner.adapter = adapter

        binding.citySpinner.onItemSelectedListener = object :
AdapterView.OnItemSelectedListener {
            override fun onItemSelected(
                parent: AdapterView<*>,
                view: View?,
                position: Int,
                id: Long
            ) {
                selectedCity = cityNames[position]
                weatherDataRef.addListenerForSingleValueEvent(object :
ValueEventListener {
                    @RequiresApi(Build.VERSION_CODES.O)
                    override fun onDataChange(dataSnapshot: DataSnapshot) {
                        weatherDataListener.onDataChange(dataSnapshot)
                    }

                    override fun onCancelled(error: DatabaseError) {
                    }
                })
            }

            override fun onNothingSelected(parent: AdapterView<*>) {
            }
        }
    }

    @RequiresApi(Build.VERSION_CODES.O)
    private fun updateCurrentCardView(cityWeatherData: WeatherData) {
        val currentWeatherData = cityWeatherData.current
        val forecastWeatherData =
cityWeatherData.forecast?.daily?.values?.firstOrNull()
        val maxMinTemp =
            "${forecastWeatherData?.maxtemp_c?.toInt()} °C,
${forecastWeatherData?.mintemp_c?.toInt()} °C."
        val condition = currentWeatherData?.condition
        val lastUpdate = currentWeatherData?.last_updated
        val isDaytime = isDaytime(lastUpdate)
        val iconFileName = getIconFileName(condition, isDaytime)
        val iconFilePath = "file:///android_asset/$iconFileName"
        val temp = "${currentWeatherData?.temp_c?.toInt()} °C"
        val wind = "${currentWeatherData?.wind_kph} km/h"

        activity?.runOnUiThread {
            binding.textCondition.text = condition
            binding.textData.text = lastUpdate
            binding.textTemp.text = temp
            binding.textWind.text = wind
            binding.textMaxMinTemp.text = maxMinTemp
```

```
            binding.imageWeather.load(iconFilePath)
        }
    }

    @RequiresApi(Build.VERSION_CODES.O)
    private fun isDaytime(lastUpdate: String?): Boolean {
        if (lastUpdate.isNullOrEmpty()) {
            return false
        }

        val lastUpdateDateTime = LocalDateTime.parse(
            lastUpdate, DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm")
        )

        val hour = lastUpdateDateTime.hour

        return hour in 7..21
    }

    private fun getIconFileName(condition: String?, isDaytime: Boolean): String {
        return when (condition) {
            "Sunny" -> "day_1.png"
            "Clear" -> "night_1.png"
            "Partly cloudy" -> if (isDaytime) "day_2.png" else "night_2.png"
            "Cloudy" -> if (isDaytime) "day_3.png" else "night_3.png"
            "Mist" -> if (isDaytime) "day_4.png" else "night_4.png"
            "Light rain" -> if (isDaytime) "day_5.png" else "night_5.png"
            "Heavy snow" -> if (isDaytime) "day_6.png" else "night_6.png"
            "Rain with thunder" -> if (isDaytime) "day_7.png" else "night_7.png"
            else -> "loading.png"
        }
    }

    companion object {
        @JvmStatic
        fun newInstance() = MainFragment()
    }
}
```

HoursFragment:

```
class HoursFragment : Fragment() {
    private lateinit var binding: FragmentHoursBinding
    private val hourlyAdapter = HoursAdapter()

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = FragmentHoursBinding.inflate(inflater, container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
```

```
        super.onViewCreated(view, savedInstanceState)
        initRecyclerView()
    }

    private fun initRecyclerView() {
        binding.rcView.apply {
            adapter = hourlyAdapter
            layoutManager = LinearLayoutManager(requireContext())
        }
    }

    fun updateWeatherDataHourly(hourlyWeatherList: List<HourlyWeather>) {
        Log.d("MyLog", "Updating weather data in HoursFragment: $hourlyWeatherList")
        val sortedList = hourlyWeatherList.sortedBy { it.time }
        hourlyAdapter.submitList(sortedList)
    }

    companion object {
        @JvmStatic
        fun newInstance() = HoursFragment()
    }
}
```

DaysFragment.kt:

```
class DaysFragment : Fragment() {
    private lateinit var binding: FragmentDaysBinding
    private val daysAdapter = DaysAdapter()

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = FragmentDaysBinding.inflate(inflater, container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        initRecyclerView()
    }

    private fun initRecyclerView() {
        binding.rcView.apply {
            adapter = daysAdapter
            layoutManager = LinearLayoutManager(requireContext())
        }
    }

    fun updateWeatherDataDaily(dailyWeatherList: List<DailyWeather>) {
        Log.d("MyLog", "Updating weather data in DaysFragment: $dailyWeatherList")
        val sortedList = dailyWeatherList.sortedBy { it.date }
        daysAdapter.submitList(sortedList)
    }
```

```
    companion object {
        @JvmStatic
        fun newInstance() = DaysFragment()
    }
}
```

fragment_main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".fragments.MainFragment">

    <ImageView
        android:id="@+id/imageWall"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:alpha="0.75"
        android:scaleType="fitXY"
        android:src="@drawable/pxfuel"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        android:contentDescription="TODO" />

    <androidx.cardview.widget.CardView
        android:id="@+id/cardView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="5dp"
        android:layout_marginTop="5dp"
        android:layout_marginEnd="5dp"
        android:backgroundTint="@color/card_view"
        app:cardCornerRadius="10dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent">

        <androidx.constraintlayout.widget.ConstraintLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent">

            <TextView
                android:id="@+id/textData"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginStart="10dp"
                android:layout_marginTop="10dp"
                android:fontFamily="serif"
```

```
        android:textColor="@android:color/secondary_text_light"
        android:textScaleX="1.1"
        android:visibility="visible"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/textTemp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="25dp"
        android:fontFamily="serif"
        android:textColor="@color/text_color"
        android:textScaleX="1.1"
        android:textSize="55sp"
        android:visibility="visible"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/citySpinner" />

    <ImageView
        android:id="@+id/imageWeather"
        android:layout_width="80dp"
        android:layout_height="80dp"
        android:layout_marginTop="20dp"
        android:contentDescription="weather"
        android:scaleX="1"
        android:scaleY="1"
        android:src="@drawable/ic_launcher_foreground"
        android:visibility="visible"
        app:layout_constraintBottom_toBottomOf="@+id/textTemp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toEndOf="@+id/textTemp"
        app:layout_constraintTop_toTopOf="@+id/textTemp" />

    <ImageView
        android:id="@+id/imWind"
        android:layout_width="40dp"
        android:layout_height="40dp"
        android:contentDescription="wind"
        android:src="@drawable/wind_power"
        app:layout_constraintBottom_toBottomOf="@+id/textTemp"
        app:layout_constraintEnd_toStartOf="@+id/textTemp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="@+id/imageWeather"
        tools:ignore="ImageContrastCheck" />

    <TextView
        android:id="@+id/textWind"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:fontFamily="serif"
        android:textColor="@color/text_color"
        android:textScaleX="1.1"
        android:visibility="visible"
        app:layout_constraintBottom_toBottomOf="@+id/imageWeather"
        app:layout_constraintEnd_toEndOf="@+id/imWind"
```

```
            app:layout_constraintStart_toStartOf="@+id/imWind" />

        <TextView
            android:id="@+id/textCondition"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="20dp"
            android:fontFamily="serif"
            android:textColor="@color/text_color"
            android:textScaleX="1.1"
            android:visibility="visible"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toBottomOf="@+id/textTemp" />

        <TextView
            android:id="@+id/textMaxMinTemp"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="5dp"
            android:layout_marginBottom="10dp"
            android:fontFamily="serif"
            android:textColor="@color/text_color"
            android:textScaleX="1.1"
            android:visibility="visible"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toBottomOf="@+id/textCondition" />

        <Spinner
            android:id="@+id/citySpinner"
            android:layout_width="100dp"
            android:layout_height="wrap_content"
            android:layout_marginStart="40dp"
            android:background="#0000"
            android:dropDownHorizontalOffset="148dp"
            android:dropDownVerticalOffset="48dp"
            android:minHeight="48dp"
            android:popupBackground="@color/teal_700"
            android:popupElevation="8dp"
            android:scaleX="1.5"
            android:scaleY="1.5"
            android:spinnerMode="dropdown"
            android:textSize="96sp"
            android:typeface="serif"
            app:layout_constraintEnd_toEndOf="@+id/textTemp"
            app:layout_constraintStart_toStartOf="@+id/textTemp"
            app:layout_constraintTop_toBottomOf="@+id/textData"
            tools:ignore="SpeakableTextPresentCheck" />


    </androidx.constraintlayout.widget.ConstraintLayout>
</androidx.cardview.widget.CardView>

<com.google.android.material.tabs.TabLayout
    android:id="@+id/tabLayout"
```

```
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginTop="5dp"
        android:backgroundTint="@color/card_view"
        app:layout_constraintEnd_toEndOf="@+id/cardView"
        app:layout_constraintStart_toStartOf="@+id/cardView"
        app:layout_constraintTop_toBottomOf="@+id/cardView">

        <com.google.android.material.tabs.TabItem
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hours" />

        <com.google.android.material.tabs.TabItem
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Days" />

    </com.google.android.material.tabs.TabLayout>

    <androidx.viewpager2.widget.ViewPager2
        android:id="@+id/vp"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:layout_marginBottom="5dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="@+id/tabLayout"
        app:layout_constraintStart_toStartOf="@+id/tabLayout"
        app:layout_constraintTop_toBottomOf="@+id/tabLayout" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

list_item:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.cardview.widget.CardView
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="2.5dp"
    android:backgroundTint="@color/card_view"
    app:cardCornerRadius="5dp"
    app:cardElevation="2dp">

    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <TextView
            android:id="@+id/textDate"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginStart="15dp"
            android:layout_marginTop="10dp"
            android:fontFamily="serif"
```

```
    android:text="22.05.2023"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<TextView
    android:id="@+id/textCondition"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="5dp"
    android:layout_marginBottom="10dp"
    android:fontFamily="serif"
    android:text="Sunny"
    android:textColor="@color/text_color"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintStart_toStartOf="@+id/textDate"
    app:layout_constraintTop_toBottomOf="@+id/textDate" />

<ImageView
    android:id="@+id/imW"
    android:layout_width="50dp"
    android:layout_height="50dp"
    android:layout_marginTop="5dp"
    android:layout_marginEnd="10dp"
    android:layout_marginBottom="5dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:srcCompat="@drawable/ic_launcher_foreground" />

<TextView
    android:id="@+id/textTemp"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="10dp"
    android:fontFamily="serif"
    android:gravity="center"
    android:text="22'C"
    android:textColor="@color/text_color"
    android:textSize="24sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<ImageView
    android:id="@+id/imWind"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginEnd="10dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toStartOf="@+id/imW"
    app:layout_constraintStart_toEndOf="@+id/textTemp"
    app:layout_constraintTop_toTopOf="parent"
    app:srcCompat="@drawable/wind_power" />

<TextView
    android:id="@+id/textWind"
```

```
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="10km/h"
            android:textColor="@color/text_color"
            android:textSize="10sp"
            app:layout_constraintEnd_toEndOf="@+id/imWind"
            app:layout_constraintStart_toStartOf="@+id/imWind"
            app:layout_constraintTop_toBottomOf="@+id/imWind" />

    </androidx.constraintlayout.widget.ConstraintLayout>
</androidx.cardview.widget.CardView>
```