МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА ANDROID ЗАСТОСУНКУ ДЛЯ ЗБОРУ ТА АНАЛІЗУ ІНФОРМАЦІЇ З ВІДКРИТИХ ДЖЕРЕЛ»**

Виконав: студент 2 курсу, групи 8.1212-іпз-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
А.Є. Халач
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, к.ф.-м.н. Кривохата А.Г.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Халачу Андрію Євгеновичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка Android застосунку для збору та аналізу інформації з відкритих джерел

керівник роботи Кривохата Анастасія Григорівна, к.ф.-м.н
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » травня 2023 року № 642-с

2. Строк подання студентом роботи 01.12.2023 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка Android застосунку для збору та аналізу інформації з відкритих джерел.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 03.05.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	24.05.2023	
2.	Збір вихідних даних.	12.06.2023	
3.	Обробка методичних та теоретичних	03.07.2023	
	джерел.		
4.	Розробка першого та другого розділу.	04.09.2023	
5.	Розробка третього розділу.	06.11.2023	
6.	Оформлення та нормоконтроль	27.11.2023	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	15.12.2023	

Студент _____ А.Є. Халач
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.Г. Кривохата
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка Android застосунку для збору та аналізу інформації з відкритих джерел»: 57 с., 28 рис., 14 джерел.

ГРАФІК, ANDROID-РОЗРОБКА, ANDROID STUDIO, API, GRAPHVIEW, JAVA.

Об’єкт дослідження – процес розробки Android застосунку.

Мета роботи: розробка Android застосунку для збору та аналізу інформації з відкритих джерел.

Метод дослідження – методи проєктування та розробки програмного забезпечення.

У кваліфікаційній роботі розглянуто інструменти розробки Android застосунків, розглянуто альтернативні API та додатки. Досліджено взаємодію з OpenWeatherAPI та бібліотекою GraphView. Було виконано налаштування робочого середовища та проєктування додатку.

Кваліфікаційна робота складається з трьох розділів. У першому розділі наведено опис ОС Android та видів мобільних додатків. Розглянуто основні технології метеорології, додатки та API які використовують їх. Другий розділ присвячено проєктуванню застосунку. Основну концепцію та логіку побудови додатку проілюстровано відповідними діаграмами. В третьому розділі наведено фрагменти коду основних модулів та результати роботи застосунку.

SUMMARY

Master’s qualifying paper «Development of the Android Application for Collecting and Analyzing Information From Open Sources»: 57 pages, 28 figures, 14 references.

GRAPH, ANDROID DEVELOPMENT, ANDROID STUDIO, API, GRAPHVIEW, JAVA.

Object of study – the process of creating an Android application using the API.

The aim of the study is to develop an Android application for collecting and analyzing information from open sources.

Research method are software design and development methods.

The qualification work examines the tools for developing Android applications, considers alternative APIs and applications. The interaction with OpenWeatherAPI and the GraphView library was investigated. The work includes setting up a working environment and designing an application.

The qualification work consists of three sections. The first chapter describes the Android OS and types of mobile applications. The main meteorology technologies, applications and APIs that use them are considered. The second section is devoted to the design of the application. The basic concept and logic of building an application are illustrated with diagrams. The third section contains code snippets of the main modules and the results of the application.

ЗМІСТ

ВСТУП

Сучасний розвиток інформаційних технологій і використання мобільних пристроїв стали невід'ємною частиною нашого повсякденного життя. У цьому контексті розробка мобільних додатків, наприклад для моніторингу погоди, набуває все більшого значення. Забезпечення швидкого та зручного доступу до актуальної та достовірної інформації про погоду стає ключовим завданням для багатьох користувачів. У кваліфікаційній роботі розроблено Android-додаток прогнозу погоди за допомогою відкритих джерел, який надає користувачам зручний та інтуїтивно зрозумілий інтерфейс для отримання актуальної інформації про погоду в будь-якому регіоні. Для забезпечення цієї функціональності додаток використовує сервіс OpenWeather API як джерело даних про погоду та бібліотеку GraphView. OpenWeather API визнаний своєю високою точністю та широким покриттям, що дозволяє надавати достовірний прогноз для різних регіонів світу. Інтеграція з цим сервісом дозволить додатку отримувати актуальні дані про погоду та забезпечити користувачів достовірними та точними прогнозами.

Для ефективної взаємодії з OpenWeather API розроблений застосунок використовує HTTP-запити через архітектуру REST. Такі запити можуть включати методи GET для отримання інформації про погоду за певними параметрами, наприклад, за координатами місцезнаходження, назвою міста чи іншими фільтрами. Це забезпечить швидкий та надійний обмін даними між додатком та сервером погодного сервісу, що є ключовим аспектом для забезпечення коректної роботи додатка.

Для досягнення мети було поставлено такі завдання:

– проаналізувати програмне забезпечення для розробки;
– спроєктувати додаток;
– розробити інтерфейс додатку;
– інтегрувати зовнішні API для отримання актуальних даних про

погоду;

– реалізувати функціонал відображення поточної погоди;

– проаналізувати сучасне програмне забезпечення для розробки.

Структурно кваліфікаційна робота складається з трьох розділів, переліку посилань та двох додатків.

1 ТЕОРЕТИЧНА ЧАСТИНА

1.1 Актуальність теми

Актуальність розробки програми прогнозу погоди для Android зумовлена невід’ємною роллю інформації про погодні умови в повсякденному житті людей. Сучасні технології дають змогу створювати зручні та багатофункціональні мобільні додатки, здатні надавати точну й актуальну інформацію про погоду в режимі реального часу.

Погодні зміни мають значний вплив на різні аспекти повсякденного життя людей, включаючи подорожі, заняття спортом, планування заходів і звичайні переміщення. Враховуючи цю реальність, створення надійного та зручного застосунку для прогнозу погоди стає важливим завданням, яке сприяє підвищенню комфорту та ефективності повсякденної діяльності користувачів. Крім того, використання різних API надає унікальні можливості для поліпшення функціональності застосунку, забезпечуючи точні дані про погоду та інтеграцію з географічною інформацією. Ці API являють собою потужні інструменти, здатні значно розширити можливості додатка і забезпечити користувачами більш повний та інформативний досвід використання.

Таким чином, розробка Android додатка прогнозу погоди з використанням API є актуальним і перспективним завданням, що поєднує в собі високий потенціал для задоволення потреб користувачів в актуальній і точній інформації про погоду.

1.2 Огляд ОС Android

Операційна система Android – це відкрита платформа, а значить, що вона не прив’язана до одного провайдера. Це допомагає Android завоювати ринок,

оскільки будь-який виробник і провайдер може створювати і продавати пристрої, що підтримують цю операційну систему, також це стимулює конкуренцію між виробниками, що сприяє виникненню новаторських ідей і технологій у світі мобільних пристроїв. Вихідний код Android доступний кожному для вивчення або модифікування. Це дає змогу дати широкий простір для створення зручних інтерфейсів і додатків для Android-пристроїв [1]. Можливості Android включають:

- можливість здійснювати та приймати телефонні дзвінки;
- можливість встановлювати різноманітні додатки з Google Play або інших джерел;
- функціонал для зйомки фотографій та запису відео за допомогою камери пристрою;
- можливість виходу в Інтернет для перегляду вебсторінок, використання онлайн-сервісів та інтерактивних додатків;
- підтримка безпровідних технологій, таких як 4G, Wi-Fi, GPS, NFC, Bluetooth тощо;
- можливість використовувати захисні засоби, такі як сканер відбитку пальця, для верифікації користувача.

ОС Android може використовуватися на пристроях з різними розмірами екрана та іншими технічними характеристиками, тому вона постачається з набором інструментів, які допомагають додатку адаптуватися під конкретний пристрій. Політика Google більш жорстка. Наприклад, якщо застосунок вимагає наявність фронтальної камери, то цей застосунок в Google Play побачать телефони тільки з фронтальною камерою [1].

Операційна система Android користується великою спільнотою розробників, що сприяє обміну досвідом та розвитку кращих практик. Доступність великої кількості інструментів та ресурсів робить процес розробки більш продуктивним.

Важливо також відзначити, що Android дозволяє розробникам

використовувати різноманітні інтегровані сервіси та API для покращення функціональності своїх додатків. Наприклад, для розробки WeatherApp було використано API OpenWeather, що надає зручний доступ до точних даних про погоду та інтеграції з географічною інформацією.

Загалом, високий рівень адаптованості, відкритий характер та можливості розробки для різних пристроїв роблять операційну систему Android привабливим вибором для розробки мобільних додатків.

1.3 Види Android додатків

Взагалі існує три види мобільних додатків: нативні, веб та гібридні.

Додатки, розроблені нативно для конкретної операційної системи, відомі як нативні додатки для своєї платформи. Наприклад, додатки, розроблені нативно для Android, не можуть бути запущені на iOS.

Основними перевагами нативних додатків є їх здатність максимально використовувати всі можливості програмного забезпечення та функцій операційної системи. Ці додатки можуть прямо взаємодіяти з різними апаратними засобами пристрою, такими як камера, мікрофон та GPS. Вони також відрізняються високою швидкістю та надійністю, ефективно використовуючи ресурси смартфону в порівнянні з іншими типами мобільних додатків [2]. Однак нативні додатки мають свій недолік, пов’язаний з їхньою специфікою. Якщо потрібно створити додаток для різних платформ, це вимагатиме великих зусиль для розробки та підтримки кожної платформи окремо. Це через те, що мови програмування для різних операційних систем відрізняються, і для кожної платформи доведеться використовувати відмінні підходи та інструменти.

Це значно підвищує вартість і тривалість розробки. Крім того, для кожної версії додатка потрібно буде забезпечити подальше функціонування після випуску та виконувати оновлення для кожної з них окремо [2].

Вебдодаток, з самої суті, не може вважатися повноцінним додатком. Це, по

суті, вебсайт, оформлений у вигляді нативного додатка, але йому відсутній повний функціонал останнього. Зазвичай вебдодатки розробляються мовами, такими як HTML5, а також іншими широко використовуваними інструментами, включаючи CSS, JavaScript та Ruby. Вони відкриваються через браузер і не вимагають окремого встановлення, що відрізняє їх від нативних аналогів.

Однією з основних переваг вебдодатків є відсутність необхідності у встановленні та оновленнях, оскільки сторінка автоматично оновлюється при її відкритті. Вони також не займають простір у пам’яті пристрою. Проте важливо враховувати особливість цих додатків: вони повністю залежать від використаного браузера, і через різницю у браузерах може виникнути проблема з роботою деяких функцій в певних випадках [2].

Гібридні додатки поєднують особливості нативних і вебдодатків. Схожість з нативними додатками полягає в тому, що їх можна встановити з магазину, наприклад, Play Market. Основною мовою розробки та функціональністю вони подібні до вебдодатків, використовуючи HTML5. Навіть при обробці через вбудований у додаток браузер, гібридні додатки можуть взаємодіяти з апаратними засобами пристрою, такими як камера чи мікрофон.

Однією із вагомих переваг гібридних додатків є їхній кросплатформений характер: додаток можна написати один раз мовою HTML5 і використовувати його на різних операційних системах. Це робить їх більш економічними в розробці порівняно з нативними додатками. Крім того, гібридні додатки мають менший розмір, завдяки чому вони швидше встановлюються, що особливо зручно для користувачів у місцях з повільним Інтернетом [2].

Однак недоліком гібридних додатків є менша швидкодія та потужність порівняно з нативними аналогами, також гібридні додатки можуть зазнавати обмежень у доступі до специфічних API та функцій пристрою, які нативні додатки можуть використовувати повністю [2].

Для розробки додатку з прогнозу погоди буде використовуватися нативний спосіб, оскільки задачею є розробка додатку для платформи Android.

1.4 Технології та методи збору даних про погоду

У цьому розділі буде проведено огляд технологій і методів, що використовуються для збору даних про погоду, що є важливим аспектом для розроблення програми прогнозу погоди. Розглянемо основні технології та методи, які забезпечують надійність і актуальність інформації про погоду.

Автоматичні метеостанції. Використовують сенсори та датчики для вимірювання різних параметрів, таких як температура, вологість, тиск та інші. Ці станції автоматично передають дані в центральні системи.

Геостаціонарні супутники. Супутники, що знаходяться на геостаціонарних орбітах, надають безперервне спостереження за погодними умовами на великих територіях. Вони передають дані про вологість, хмарність та інші параметри.

Полярні орбітальні супутники. Супутники, що рухаються по полярних орбітах, надають детальніші дані про погоду, особливо щодо температур і атмосферного складу.

Радіозондування. Зонди, оснащені сенсорами, піднімаються в атмосферу і передають дані про температуру, тиск і вологість на різних висотах. Ці дані використовуються для складання вертикальних профілів атмосфери.

Мережа ближніх станцій. Метеостанції, розташовані на земній поверхні, збирають дані про погоду в реальному часі. Їхня мережа забезпечує точні та миттєві вимірювання.

Інтегровані датчики. Багато сучасних мобільних пристроїв оснащені датчиками, що вимірюють температуру, вологість і навіть тиск. Ці дані можуть бути використані для надання місцевої інформації про погоду.

Огляд технологій і методів збору даних про погоду дає змогу зрозуміти, яким чином виходять інформаційні основи для прогнозів, що є важливим аспектом для розроблення застосунку, метою якого є надання точних і актуальних даних про погоду [3].

1.5 Огляд API отримання погоди

OpenWeather є однією з найбільш популярних платформ для отримання інформації про погоду через API (див. рис. 1.1). Ця система надає доступ до широкого спектру погодних даних, включаючи температуру, вологість, тиск, швидкість вітру, прогноз погоди та інші параметри. OpenWeatherMap пропонує як безкоштовні, так і платні плани, залежно від обсягу запитів та додаткових функцій [4].

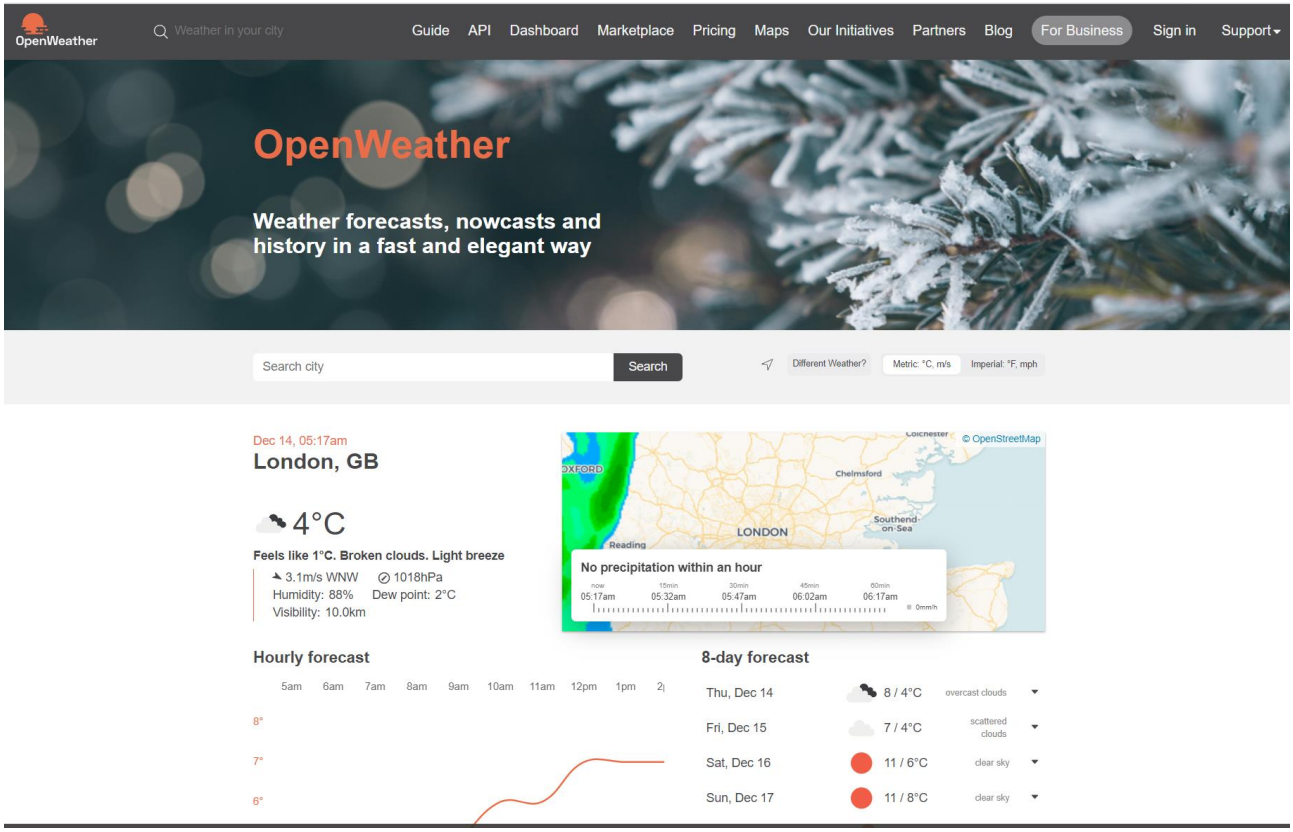

Рисунок 1.1 – Сайт OpenWeather

AccuWeather визначається як відома система для отримання точних погодових даних через API (див. рис. 1.2). Ця платформа надає можливість звертатися до поточних погодних умов, прогнозів на кілька днів, радарних зображень та інших погодних параметрів. Крім того, AccuWeather пропонує безкоштовний план, а також платні варіанти з розширеними функціями та більшим обсягом запитів [5].

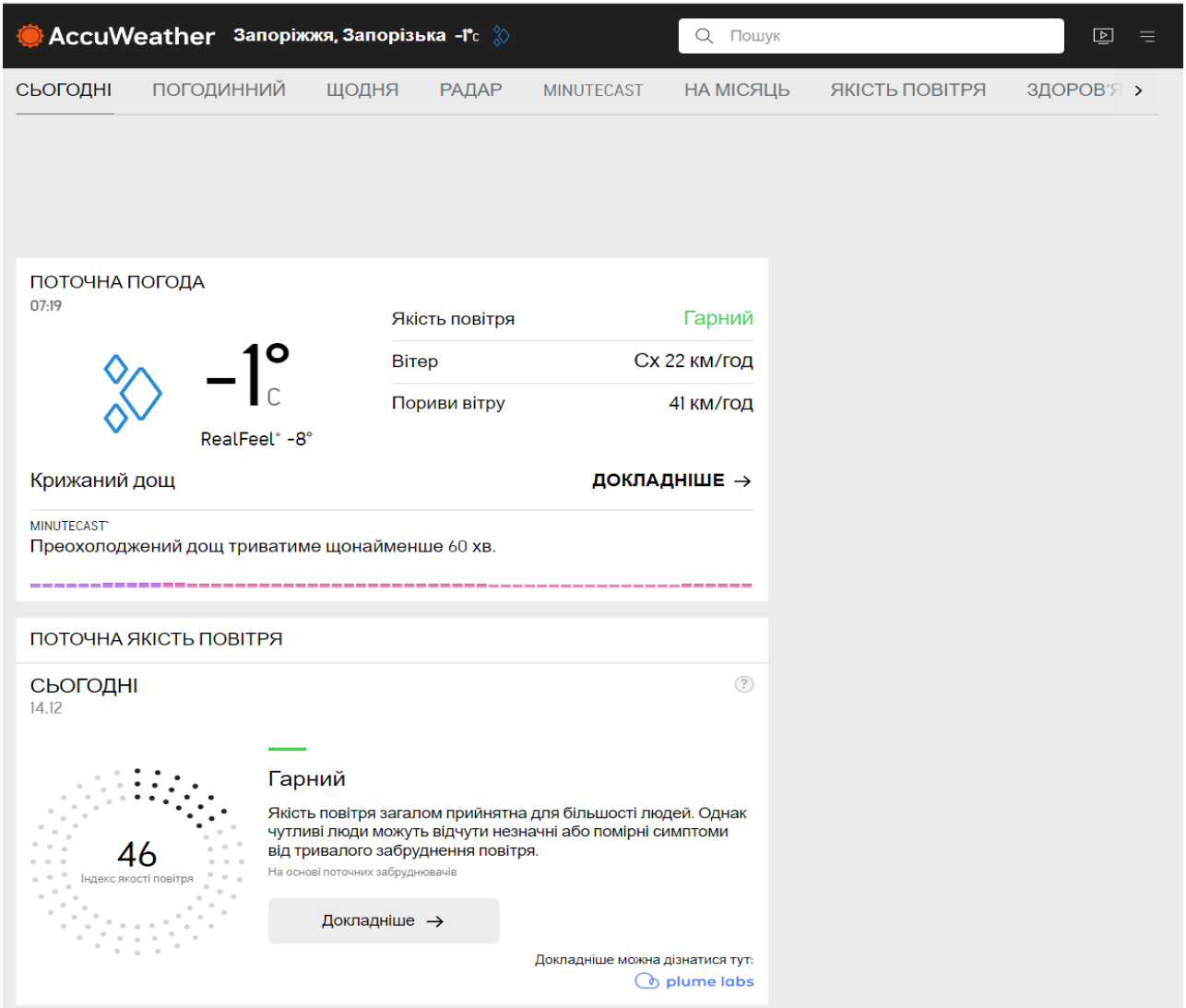

Рисунок 1.2 – Сайт AccuWeather

Weatherstack представляє собою поширене API для отримання інформації про погоду, що забезпечує простий та ефективний спосіб отримання точних даних про погоду для розробки додатків та вебсайтів (див. рис. 1.3).

Weatherstack надає детальну інформацію про погоду, охоплюючи такі параметри, як температура, вологість, швидкість вітру, атмосферний тиск та інші важливі показники.

API пропонує безкоштовний тариф, що дозволяє розробникам випробовувати його функціонал без витрат. В додатку доступні платні плани, які надають розширені можливості та обсяг запитів, відповідно до потреб користувачів [6].

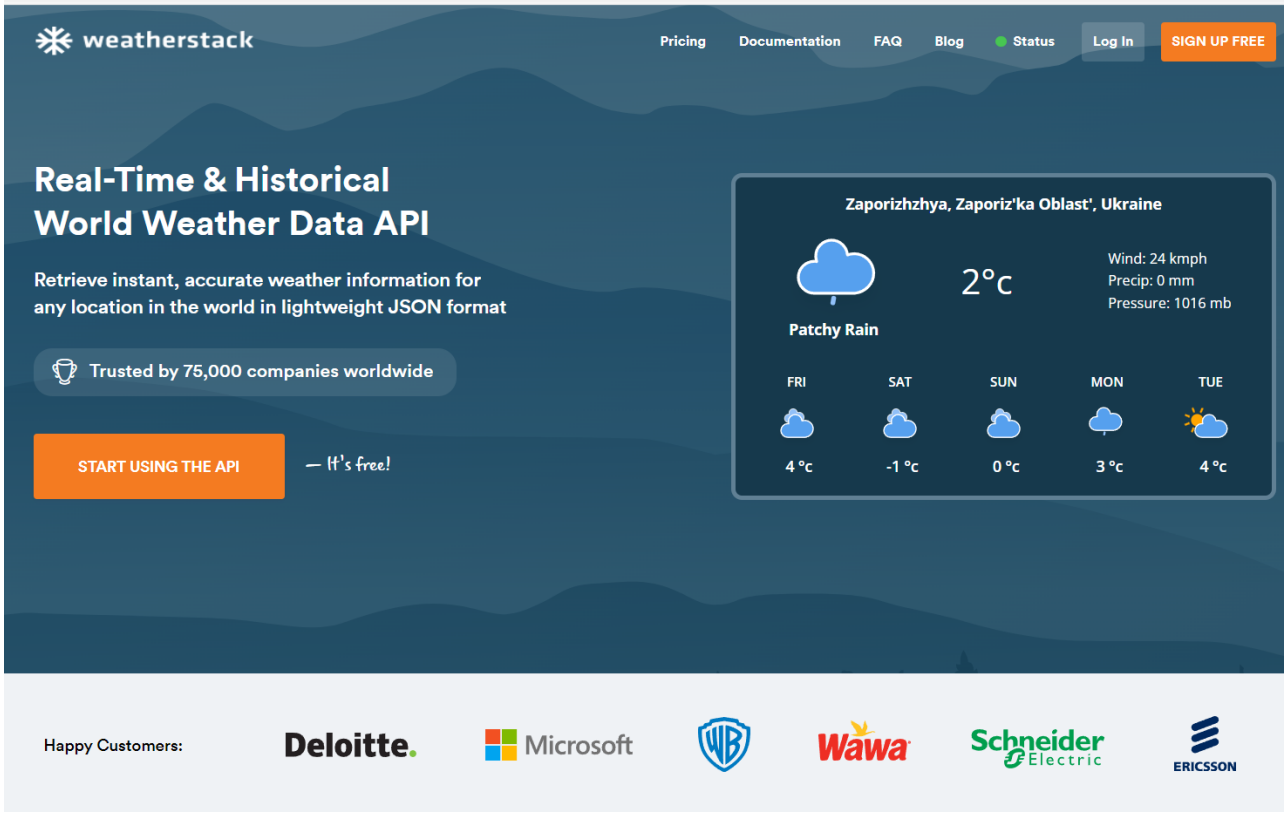

Рисунок 1.3 – Сайт WeatherStack

1.6 Огляд схожих програм

У цьому пункті розглянемо дві схожі по функціоналу але різні за підходом додатки AccuWeather та CARROT Weather.

AccuWeather надає карту з погодними умовами, сповіщення про негоду та інші корисні функції. Сервіс добре інтегрується з різними платформами та пристроями, що робить його популярним серед користувачів (див. рис. 1.4).

Глобальне покриття робить його корисним для широкого кола користувачів, а деталізація інформації, представленої у вигляді інтерактивних карт і прогнозів на кілька днів, додає зручності. Завдяки інтерфейсу, побудованому за принципами Material Design, користувачі легко орієнтуються. З іншого боку, можливий недолік у вигляді реклами у безкоштовній версії може впливати на загальний користувацький досвід. Деякі розширені функції можуть бути обмеженими в безкоштовній версії, що може вразити користувачів.

Важливим є також високе споживання ресурсів, що може впливати на тривалість роботи батареї [5].

CARROT Weather – це нестандартний додаток, який використовує гумор та персоналізацію як основну ідею (див. рис. 1.5). Однак він може не відповідати всім смакам і вимагає досить багато ресурсів, також іноді є проблеми з відображенням. Слід відмітити що немало користувачів у відгуках відмічають проблеми зі стабільністю та швидкістю додатку.

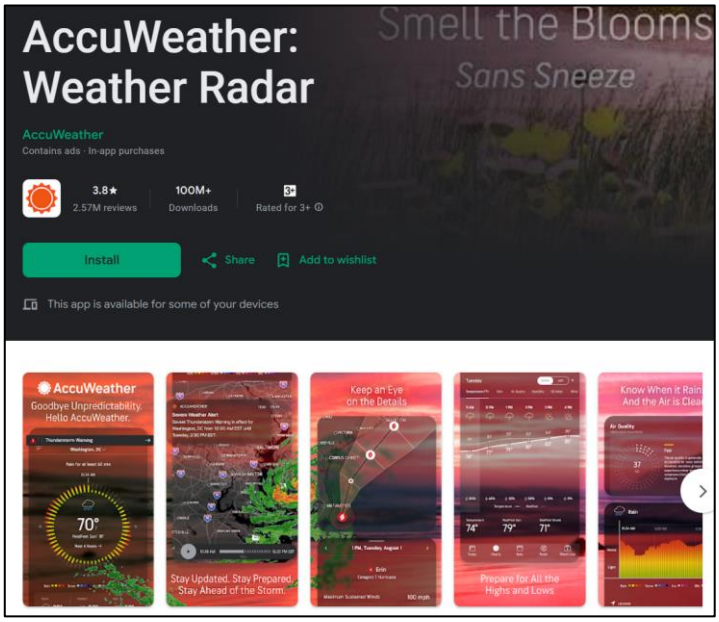

Рисунок 1.4 – AccuWeather

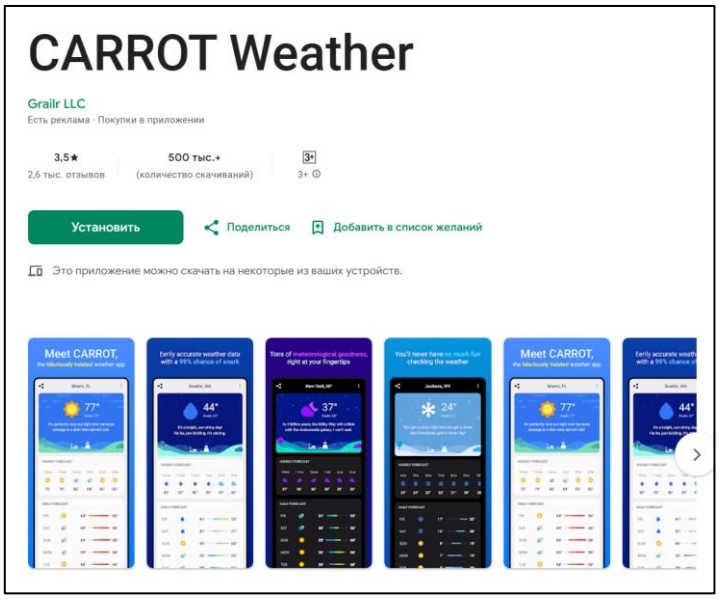

Рисунок 1.5 – Додаток CarrotWeather

Таким чином було висвітлено актуальність проблематики створення мобільних додатків для прогнозу погоди. Розглянуто основні характеристики та особливості операційної системи Android, види додатків у мобільній розробці. Окремо оглянули різні API та додатки для прогнозу погоди.

2 ПРОЄКТУВАННЯ ДОДАТКУ

2.1 Формулювання вимог до додатка

У цьому розділі ми визначимо основні вимоги, які керуватимуть процесом розроблення додатка прогнозу погоди для платформи Android. Ці вимоги визначають параметри та характеристики, які додаток повинен мати для задоволення потреб користувачів.

Формулювання вимог відіграє ключову роль у створенні функціонального, зручного у використанні та ефективного застосунку.

Функціональні вимоги. Відображення поточних умов: додаток має надавати інформацію про поточну температуру, вологість, швидкість вітру та інші ключові параметри погоди, будувати графік з можливістю вибору проміжку. Для зручності використання, інтерфейс застосунку має бути українською мовою.

Нефункціональні вимоги. Додаток має забезпечувати швидкий доступ до даних і чуйність інтерфейсу навіть при повільному інтернет-з’єднанні. Висока стабільність і надійність роботи, мінімізація помилок і збоїв. Додаток має коректно функціонувати на різних версіях операційної системи Android. Оптимальне використання батареї та ресурсів пристрою. Інтуїтивно зрозумілий інтерфейс, легкість у навігації, зрозумілі іконки та елементи керування. адаптивний дизайн, адаптивність інтерфейсу до різних розмірів екранів мобільних пристроїв.

Формулювання вимог є важливим етапом процесу розроблення, воно дає змогу визначити кінцеві цілі та очікування від додатка прогнозу погоди, забезпечуючи ясний напрямок для всього проєкту.

2.2 Середовище розробки

За допомогою середовища Android Studio, використовуючи мову JAVA, API OpenWeather та Google Maps, буде спроєктований та розроблений мобільний додаток. Цей додаток надає можливість переглядати поточну інформацію про погоду, додавати міста для швидкого доступу.

Android Studio є інтегрованим середовищем програмування та розробки для операційної системи Android, спроєктованим для заміщення плагіну ADT для Eclipse. Це середовище розроблене на основі вихідного коду продукту IntelliJ IDEA Community Edition, що розробляється компанією JetBrains. Android Studio розвивається в рамках відкритої моделі розробки та поширюється під ліцензією Apache 2.0 (див. рис. 2.1).

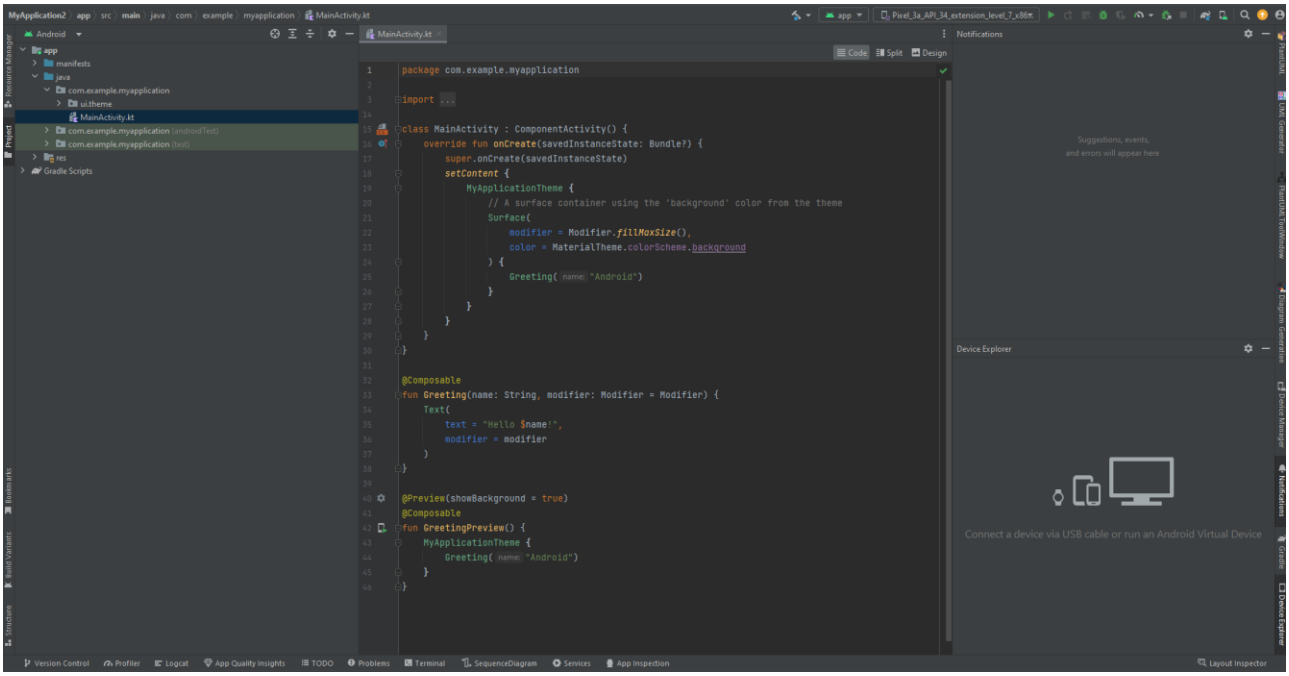

Рисунок 2.1 – Робоче місце у Android Studio

Це середовище розробки адаптоване для виконання типових завдань, пов’язаних із створенням програм для операційної системи Android. У його складі містяться інструменти для спрощення тестування програм на сумісність із різними версіями платформи. Також воно включає інструменти для

проєктування застосунків, які працюють на пристроях з екранами різної роздільності, таких як планшети, смартфони, ноутбуки, годинники, окуляри тощо.

Окрім можливостей, характерних для IntelliJ IDEA, Android Studio має додаткові функції, такі як уніфікована підсистема складання, тестування та розгортання застосунків. Ця підсистема базується на інструментах збірки Gradle і підтримує використання засобів безперервної інтеграції. Android Studio представляє собою розширене середовище розробки, яке включає ряд нових функцій та оновлень.

Живі макети (layout). Редактор з концепцією WYSIWYG надає можливість живого кодування та відображення програми в реальному часі.

Консоль розробника. Включає підказки по оптимізації, допомогу з перекладом, стеження за напрямком, агітації та акції, а також метрики Google Analytics.

Резерви бета релізів та покрокові релізи. Дозволяє зручно випробовувати та впроваджувати нові функції перед остаточним випуском.

Базування на Gradle. Система збірки Gradle є ключовою частиною середовища розробки Android Studio та грає важливу роль у вдалих розробках проєктів під операційну систему Android. Gradle є потужним інструментом для автоматизації завдань збирання, тестування та розгортання додатків.

Android-орієнтований рефакторинг та швидкі виправлення. Забезпечує зручні інструменти для оптимізації та виправлення коду.

Lint утиліти. В середовищі розробки Android Studio є важливим інструментом для покращення якості коду та забезпечення оптимальної продуктивності додатка. Вони допомагають виявляти й усувати різноманітні проблеми та недоліки під час розробки, забезпечуючи високий стандарт коду та оптимізовану роботу програми.

Використання можливостей ProGuard та підписів до програм. Допомагає забезпечити безпеку та ефективність програм.

Шаблони для створення поширених Android дизайнів та компонентів.

Надає зручні інструменти для розробки стандартних та естетичних інтерфейсів, що спрощує та прискорює процес створення інтерфейсу користувача. Ці шаблони дозволяють створювати стандартні та естетичні елементи інтерфейсу, забезпечуючи єдність та професійний вигляд додатків.

Багатий редактор макетів. Дозволяє користувачам перетягувати та розміщувати компоненти інтерфейсу методом drag-and-drop, а також одночасно переглядати макети на різних конфігураціях екранів [7].

2.3 Архітектура MVC

Програми для Android базуються на архітектурі «Модель-Вид-Контролер» (MVC). Відповідно до цієї архітектури, кожен об’єкт додатка повинен бути об’єктом моделі, виду або контролера. Це досить зручна архітектура, тому ми будемо використовувати її для праці з API (див. рис. 2.2).

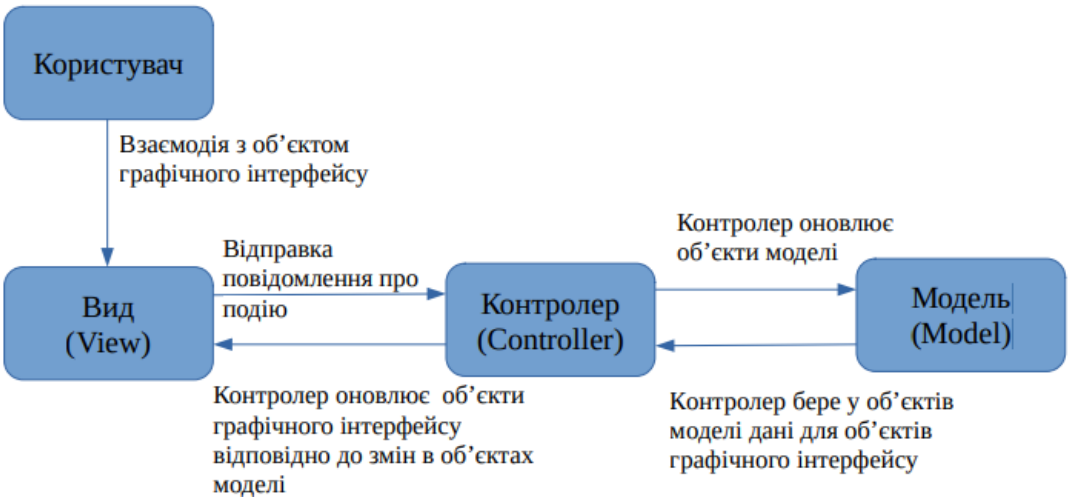

Рисунок 2.2 – Зображення архітектури MVC

Об’єкт моделі містить дані програми та бізнес-логіку. Класи моделі проєктуються для моделювання сутностей, з якими працює додаток. Об’єкти моделі утворюють рівень моделі та призначені для зберігання та управління даними.

Об’єкти виду відображаються на екрані та реагують на дії користувача. У Android використовуються класи уявлень, які можна налаштовувати, а також можна створювати власні класи уявлень. Об’єкти виду утворюють рівень подання.

Об’єкти контролера пов’язують об’єкти виду та моделі, містять логіку застосування і реагують на події, ініційовані об’єктами виду. В Android контролер зазвичай наслідує клас Activity, Fragment або Service.

Об’єкти моделі та виду не взаємодіють безпосередньо, але взаємодіють через посередника-контролера, який отримує повідомлення від одних об’єктів і передає інструкції іншим.

Розділення коду на класи та рівні спрощує розуміння програми та полегшує її супровід. Концепції MVC також сприяють повторному використанню класів та полегшують можливі модифікації.

Основними перевагами є принцип розділення обов’язків, який чітко визначає області відповідальності класів, ще одним вагомим плюсом є легкість повторного використання коду за рахунок відділення логіки від представлення [8].

MVC є однією з найпопулярніших архітектурних парадигм програмування, але існують інші підходи, які також використовуються для організації коду та розділення відповідальностей в програмних проєктах. Порівняння MVC з іншими архітектурними шаблонами наведемо нижче.

MVVM (Model-View-ViewModel). Обидві моделі пропонують розділення обов’язків та структуру для розробки додатків. MVVM же має додатковий компонент, ViewModel, який відповідає за зберігання та управління даними, які використовуються для відображення. ViewModel відокремлює представлення від бізнес-логіки та взаємодії з даними. MVVM може бути корисною, коли важлива автоматична синхронізація даних та їх відображення.

MVP (Model-View-Presenter). У MVP присутній презентер, який відокремлює взаємодію від відображення та моделі. Презентер робить взаємодію більш прозорою, оскільки він контролює потік даних в той час як у MVC може

виникнути проблема залежності виду від моделі та контролера.

Flux. Flux – це архітектурний шаблон, який визначає односторонній потік даних. Flux особливо підходить для великих та складних додатків, де прогнозованість та управління станом грають критичну роль. У той час як MVC може бути зручним для менших проєктів з меншою складністю.

2.4 Налаштування та використання API

Програмний інтерфейс застосунків (API) – це набір визначень підпрограм, протоколів та засобів для розробки програмного забезпечення. Простіше кажучи, API – це чітко визначені методи для взаємодії різних компонентів, які розробник може використовувати для швидкої розробки програм. API може бути призначений для вебсистем, операційних систем, баз даних, апаратного забезпечення та програмних бібліотек [9].

Для взаємодії з сервісом, в роботі використовуватимуться HTTP-запити за допомогою архітектури REST.

HTTP (Hyper Text Transfer Protocol) є протоколом передачі даних у комп'ютерних мережах, входячи до категорії протоколів 7-го прикладного рівня моделі OSI. Його основне завдання полягає в передачі вебсторінок, які зазвичай представляють собою текстові файли з розміткою HTML. Однак HTTP успішно використовується і для передачі різних інших файлів, які можуть бути пов'язані або не пов'язані з вебсторінками.

Принциповим припущенням HTTP є те, що клієнтська програма, зазвичай веббраузер, здатна відображати гіпертекстові вебсторінки та файли інших типів у формі, зручній для користувача. Для забезпечення правильного відображення, HTTP дозволяє клієнту дізнатися мову та кодування символів вебсторінки або запросити версію сторінки в певній мові або кодуванні за допомогою позначень, визначених у стандарті MIME.

HTTP взаємодіє за схемою «запит-відповідь», використовуючи глобальні

URI для ідентифікації ресурсів. Протокол не зберігає свого стану між запитами та відповідями, і компоненти можуть самостійно здійснювати збереження інформації про стан.

Кожен запит чи відповідь складається з трьох частин: стартового рядка, заголовків та тіла повідомлення, яке містить дані запиту, запитаний ресурс або опис проблеми, якщо запит не виконано. Стартові рядки розрізняються для запитів та відповідей. Запити можуть бути типу GET (для отримання вмісту ресурсу), POST (для передачі даних користувача), PUT (для завантаження ресурсу на сервер), DELETE (для видалення ресурсу). REST (Representational State Transfer) – це підхід до архітектури мережевих протоколів, що забезпечує доступ до інформаційних ресурсів. Був описаний та популяризований Роєм Філдінгом у 2000 році, заснований на принципах функціонування Всесвітньої павутини та можливостях HTTP. REST використовує стандартні формати передачі даних, такі як HTML, XML та JSON. Цей підхід не передбачає збереження інформації про стан між запитами та відповідями, що сприяє масштабованості системи та її еволюції з новими вимогами [11].

Існує кілька сервісів, які надають інформацію про погоду, такі як Yahoo! Weather, Weather Underground, Forecast.Io та OpenWeather. Сервіс OpenWeather було обрано через його точність, покриття та обмеження, які повністю задовольнять потреби у розробці програми без потреби в компромісах.

Для розробки додатку був обраний OpenWeather API, на об’єктивний погляд маючий найбільший функціонал та великі ліміти при використанні безкоштовно. Нижче представлено дослідження основних можливостей і функцій OpenWeather API.

OpenWeather API це потужний інструмент для отримання актуальних метеорологічних даних. Він надає різноманітні функції, такі як поточна погода, прогноз на кілька днів, геолокація тощо. API містить велику базу даних з інформацією про погоду по всьому світу.

Для розробки мобільного застосунку, важливо врахувати основні характеристики OpenWeather API. API надає можливість отримати прогноз

погоди для будь-якого міста і часу. Прогноз оновлюється щогодини і містить дані про температуру, вологість, швидкість вітру та інші параметри [12].

У мобільному застосунку можна використовувати ці дані для відображення поточної погоди та прогнозу на наступні дні. Також API дає змогу отримати інформацію про світанок і захід сонця, а також про погодні умови на найближчі кілька годин. Це корисно для користувачів, які хочуть заздалегідь спланувати свої справи або подорожі.

Мобільний застосунок може також використовувати функцію визначення геолокації користувача, щоб автоматично відображати прогноз погоди для його поточного місця розташування.

Однією з ключових особливостей є простота використання API. OpenWeatherAPI надає простий і зрозумілий інтерфейс, який дає змогу легко отримувати дані про погоду. Для початку роботи необхідно зареєструватися на сайті OpenWeather і отримати API-ключ (див. рис. 2.3), який буде використовуватися для аутентифікації запитів.

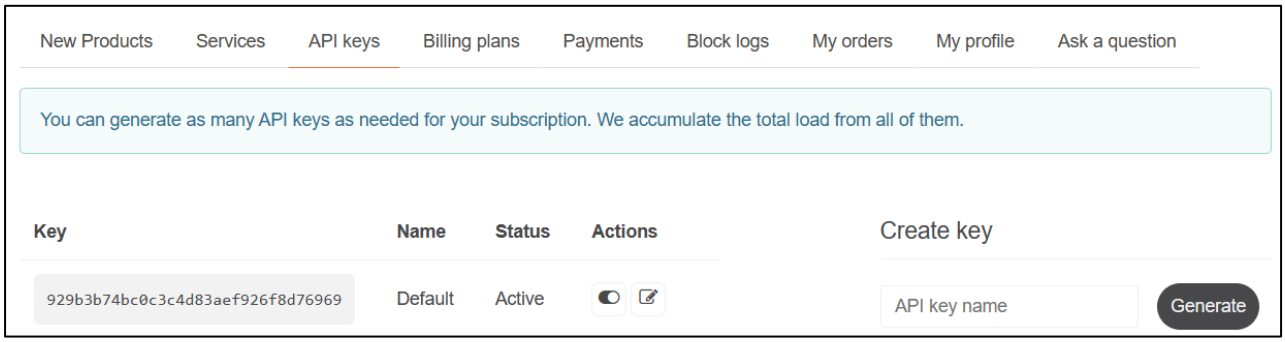

Рисунок 2.3 – Сторінка з ключами API

OpenWeatherAPI також пропонує гнучкий підхід до налаштування запитів. Ви можете вказати конкретне місце розташування (місто або координати) для отримання даних про погоду та обирати одиницю вимірювання температури, мову виведення інформації [12].

Друга основна можливість – це робота з графіками та діаграмами. За допомогою OpenWeatherAPI можна візуалізувати отримані дані про погоду,

створюючи графіки температури, тиску або інших параметрів. Це дає змогу більш наочно уявити зміни погоди в часі. Крім того, OpenWeatherAPI надає можливість роботи з різними мовами програмування. API підтримує безліч мов, таких як Python, JavaScript, PHP та інші. Це дає можливість вибрати найбільш зручну для вас мову програмування при створенні додатків, що використовують погодні дані.

Використання прикладного програмного інтерфейсу (API) через HTTP-запити та архітектуру REST буде сприяти ефективній комунікації між додатком та сервером погодного сервісу. Це забезпечить швидкий та надійний обмін даними, що є важливим елементом для забезпечення плавної роботи додатка.

Загалом, обрана платформа OpenWeather та використання їхнього API через HTTP-запити відповідають високим стандартам надійності, точності та ефективності додатка. Ці можливості стали ключовими для розробки погодного додатка, забезпечуючи надійне та швидке отримання актуальної інформації про погоду для користувачів.

2.5 Концептуальне та логічне проєктування додатку

UML (Unified Modeling Language) – це уніфікована мова моделювання, яка використовується у світі об’єктно-орієнтованого програмування та є невід’ємною частиною уніфікованого процесу розробки програмного забезпечення. Вона представляє собою відкритий стандарт, використовуючи графічні позначення для створення абстрактних моделей систем, відомих як UML-моделі [13].

UML був створений для визначення, візуалізації, проєктування та документування програмних систем. Хоча UML не є мовою програмування, але засоби виконання UML-моделей можуть генерувати виконавчий код за необхідності.

Мову UML можна успішно використовувати на всіх етапах життєвого

циклу аналізу бізнес-систем та розробки прикладних програм. Засоби UML підтримують різні види діаграм, що робить UML універсальним інструментом для опису як програмних, так і бізнес-систем [13].

Використання діаграм у мові UML дозволяє візуалізувати систему і спрощує перетворення її структури в програмний код. UML успішно впроваджується в численних програмних проєктах, де засоби автоматичної генерації коду прискорюють процес розробки, перетворюючи UML-моделі в об’єктно-орієнтований вихідний код. За допомогою UML було побудовано діаграми для поліпшення подальшої розробки [13].

Діаграма використання (Use Case Diagram) у мові моделювання UML є інструментом, який відображає відносини між акторами та прецедентами в системі і становить важливу частину моделі прецедентів, призначену для концептуального опису системи (див. рис. 2.4).

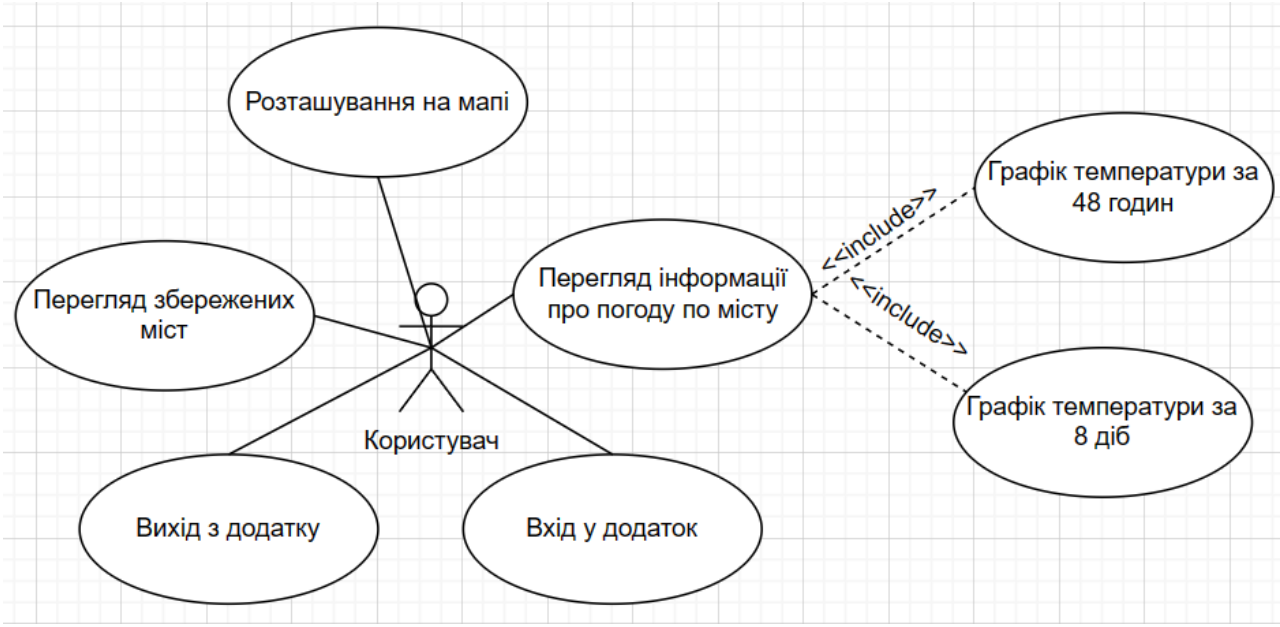

Рисунок 2.4 – Діаграма використання

Прецедент визначає можливість або функціональність системи, що може бути модельованою, і яка дозволяє користувачам отримати конкретний, вимірний результат, який їм потрібен. Кожен прецедент визначає окремий сервіс системи і описує типовий спосіб взаємодії користувача з системою.

Використання варіантів часто застосовується для конкретизації зовнішніх вимог до системи. В даному випадку основними прецедентами для користувача є перегляд інформації про погоду та перегляд збережених міст. Вони будуть доступні після авторизації користувача у системі [14].

Діаграма компонентів є важливим елементом мови моделювання UML, яка дозволяє систематизувати та візуалізувати структурні компоненти програмної системи та їх взаємодію. У контексті розробки мобільного застосунку для стартап-команд, така діаграма допомагає представити вихідний, бінарний та виконуваний код як ключові компоненти системи.

Наведено діаграму компонентів мобільного застосунку з прогнозу погоди (див. рис. 2.5).

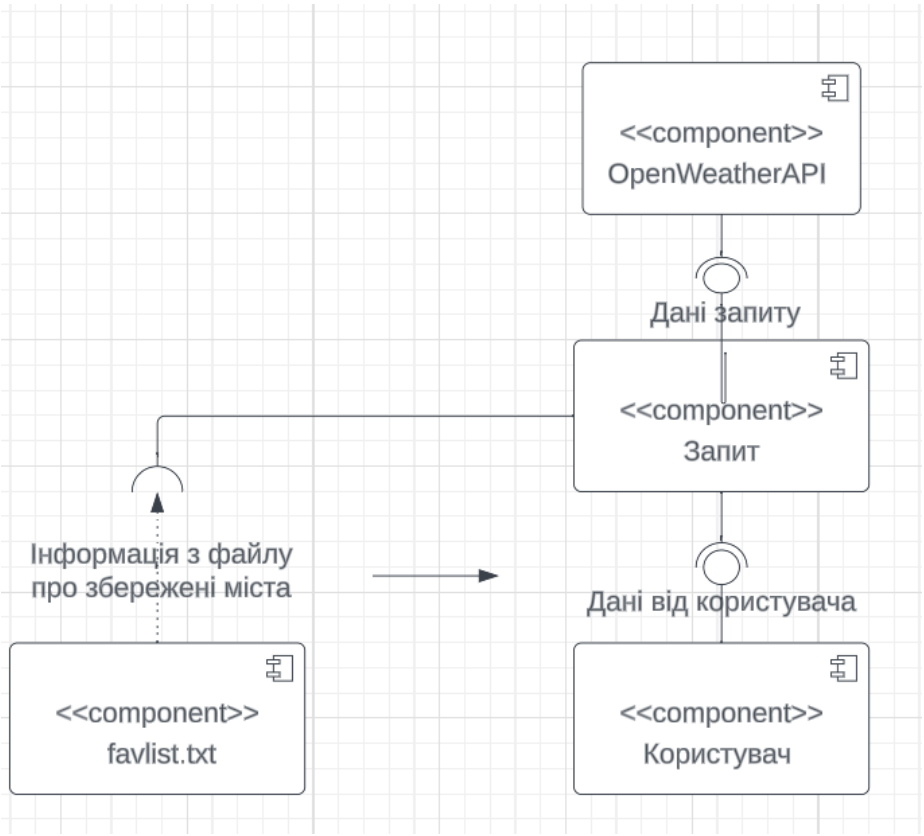

Рисунок 2.5 – Діаграма компонентів

Ця діаграма є ефективним інструментом для графічного відображення фізичного розподілу компонентів системи, що дозволяє визначити архітектурну структуру та зв’язки між ними. Її основна мета полягає в переході від логічного

представлення системи до її фізичного об’єкту, надаючи візуальне відображення взаємодії та залежностей між компонентами. Діаграма графічно ілюструє, як компоненти системи розподілені фізично та взаємодіють між собою, що сприяє кращому розумінню її структури та організації [14].

Діаграма компонентів використовується для ілюстрації загальних фізичних зв’язків між компонентами системи, що розробляється, і встановлює їхні взаємозв’язки як класифікатори, використовуючи компоненти як базові елементи.

Компоненти можуть існувати на етапі компіляції або виконання коду розроблюваної програми. Головними елементами діаграми вважаються компоненти, інтерфейси та залежності між ними. Діаграма компонентів може відобразити загальні залежності та взаємозв’язки, використовуючи компоненти як елементи фізичного представлення розробленої моделі.

Кожен компонент може втілювати набір інтерфейсів, організуючи елементи моделі в межах фізичного пакету. Діаграма також надає можливість додати коментарі та артефакти для покращення сприймання діаграми та надання додаткової інформації про реалізацію системи.

Стрілки на діаграмі, позначені пунктирною лінією та з’єднують модулі, відображають взаємозалежність між компонентами, аналогічну тій, яка використовується під час компіляції початкового коду.

Створимо діаграму класів для майбутнього додатку (див. рис. 2.6), де класи будуть представлені відповідно до архітектурного шаблону MVC (Model-View-Controller).

Діаграма класів, також відома як Static Structure diagram, є структурною діаграмою у мові моделювання UML. Вона призначена для відображення загальної структури ієрархії класів у системі, їх взаємодії, атрибутів (полів), методів, інтерфейсів і зв’язків між ними [14]. Цей тип діаграми широко використовується не лише для документування та візуалізації, але також для побудови системи за допомогою прямого або зворотного проєктування.

Метою створення діаграми класів є графічне зображення статичної

структури декларативних елементів системи, таких як класи, типи і інші.

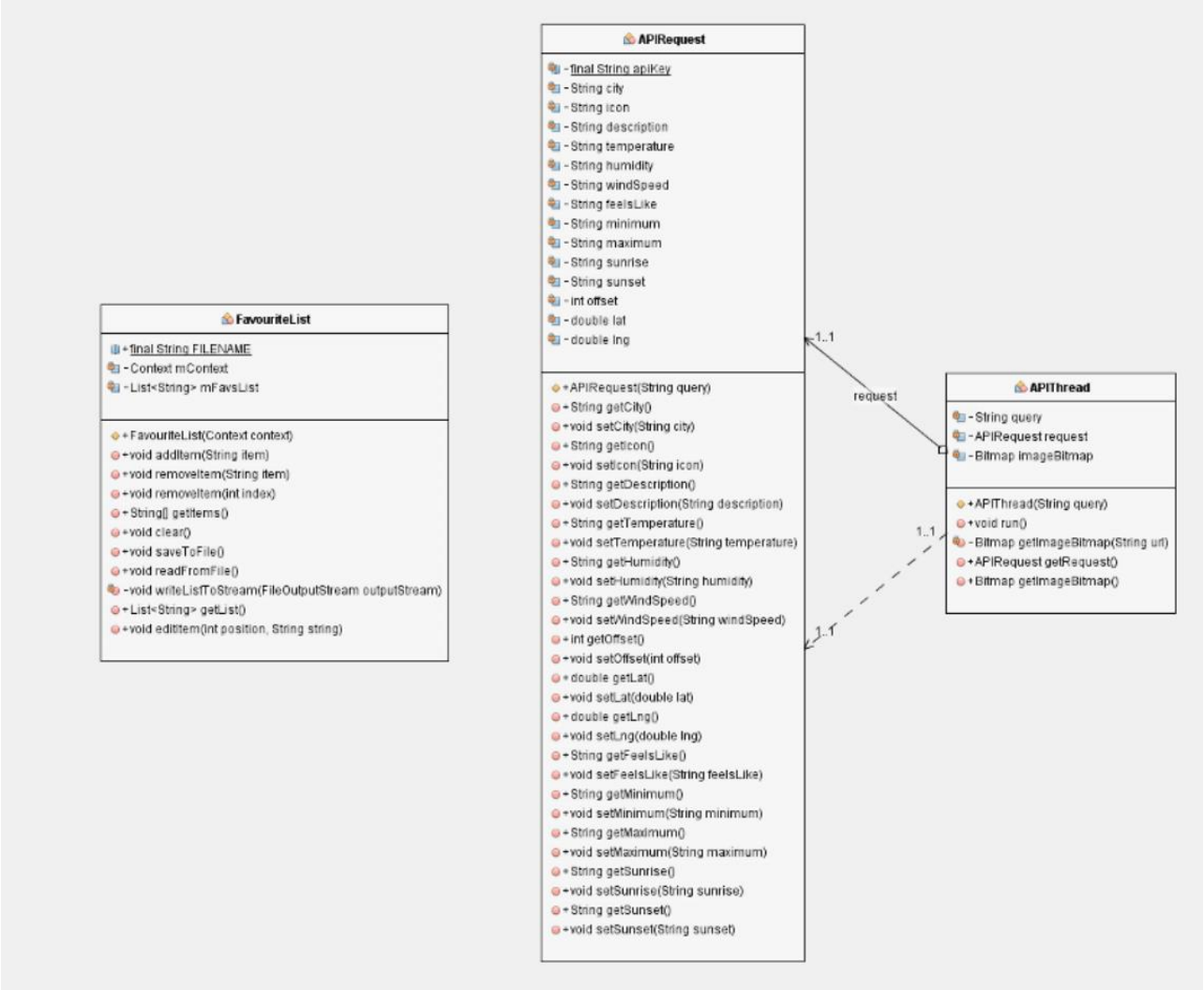

Рисунок 2.6 – Діаграма класів

У результаті аналізу вимог до додатка можна визначити дві ключові першочергові вимоги: інтеграція з погодним сервісом та використання прикладного програмного інтерфейсу (API) для ефективної взаємодії з обраною погодною платформою.

Обрання сервісу OpenWeather для інтеграції обґрунтоване його точністю, широким покриттям та відповідністю вимогам додатка. Ця інтеграція дозволить забезпечити користувачів актуальною та достовірною інформацією про погоду, що є ключовим фактором для забезпечення якості обслуговування.

У даному розділі були визначені основні вимоги, які визначатимуть

напрямок та параметри розробки додатка для прогнозу погоди на платформі Android. Формулювання цих вимог є важливим етапом, оскільки воно визначає ключові аспекти створення функціонального, зручного у використанні та ефективного застосунку.

3 РОЗРОБКА ДОДАТКУ

3.1 Підготовка до розробки

Проєкт розроблявся у середовищі розробки Android Studio. Початковим етапом є вибір шаблону (див. рис. 3.1), який визначає структуру проєкту відповідно до його функціональності.

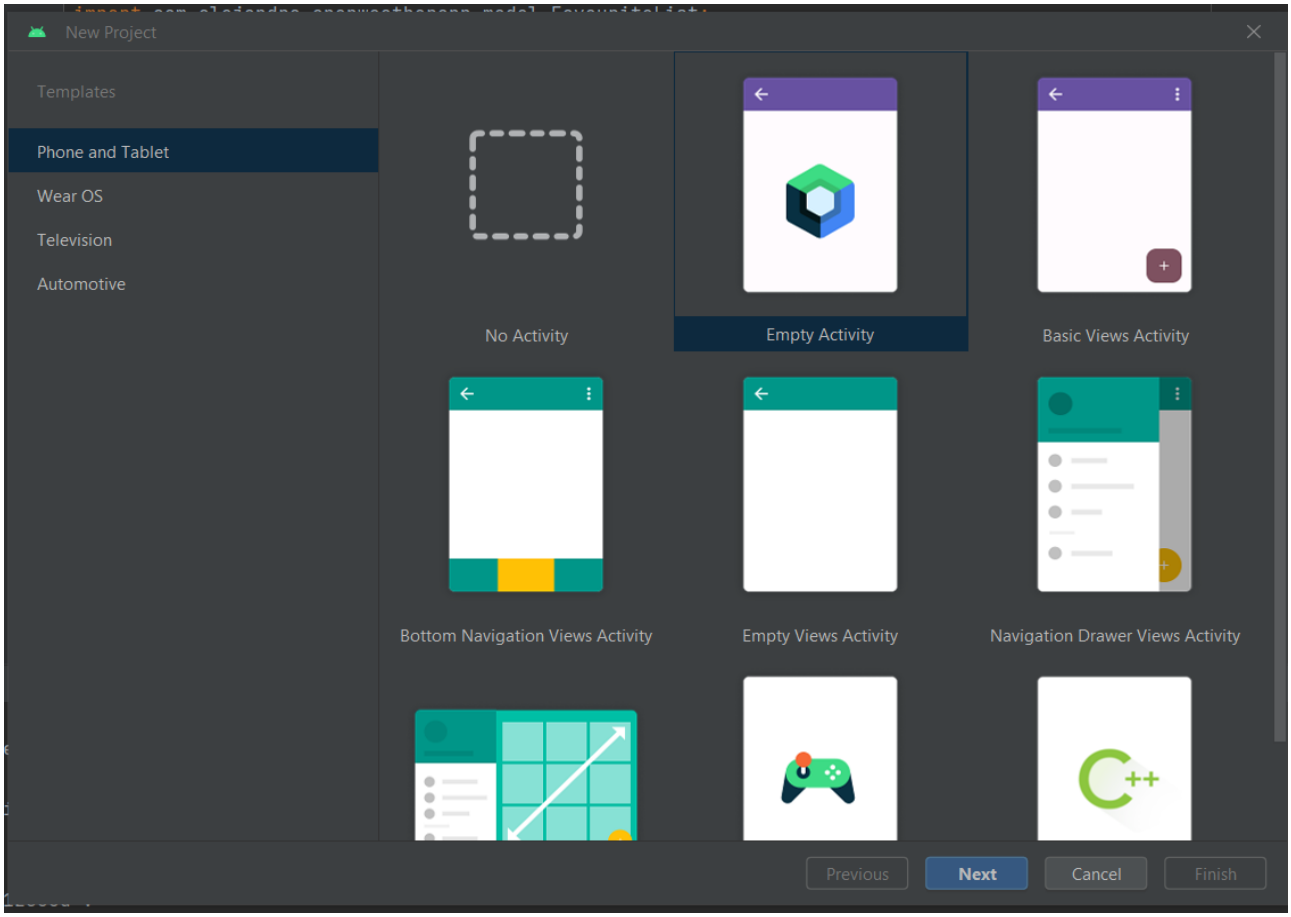

Рисунок 3.1 – Шаблони додатків у Android Studio

На наступному етапі виконується налаштування конфігурації проєкту (див. рис. 3.2). Вибираються назва мобільного додатку, пакету, розташування проєкту на комп’ютері, мова програмування та мінімальна версія Android, яку буде підтримувати додаток. Після цих налаштувань розпочинається процес будування проєкту.

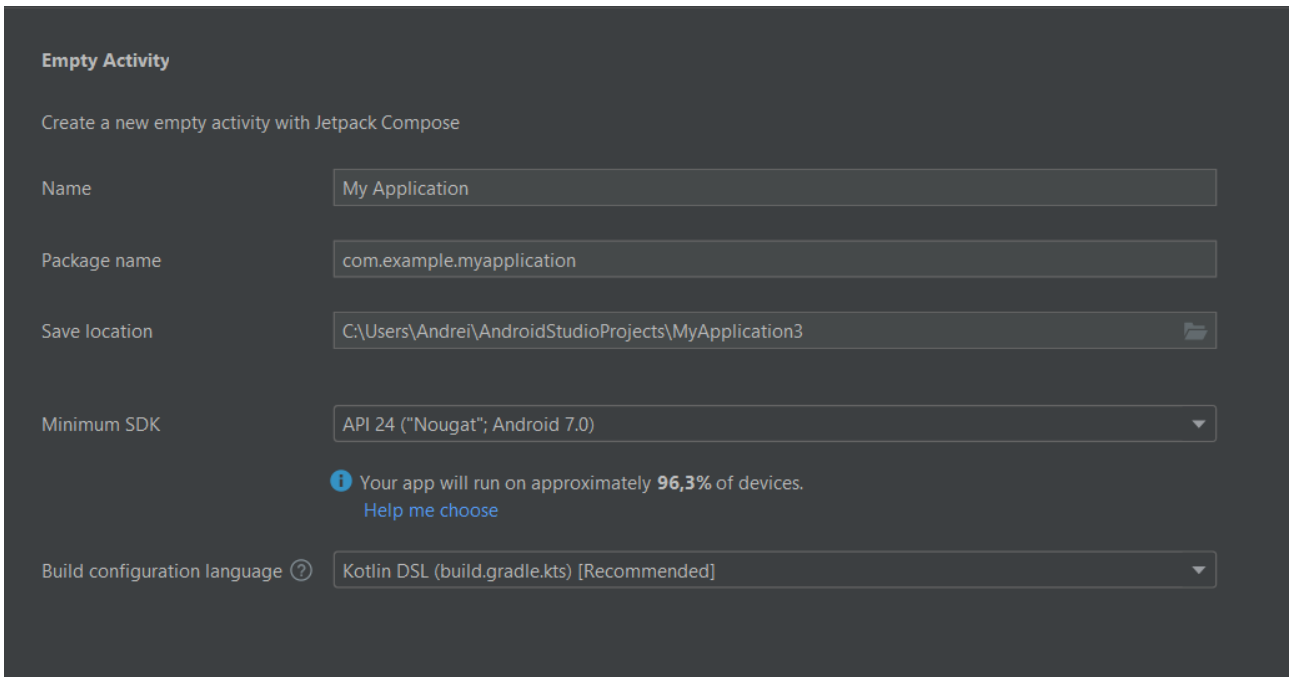

Рисунок 3.2 – Вікно створення проєкту

Після створення додатку, далі буде додавання залежностей у файл build.gradle, без них ми не зможемо використовувати бібліотеки у проєкті. Для додавання бібліотек в файл build.gradle у проєкті Android Studio, ви повинні вказати залежність у розділі dependencies (див. рис. 3.3).

```
dependencies {
    implementation 'androidx.appcompat:appcompat:1.4.2'
    implementation 'com.google.android.material:material:1.6.1'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
    implementation 'androidx.lifecycle:lifecycle-livedata-ktx:2.3.1'
    implementation 'androidx.lifecycle:lifecycle-viewmodel-ktx:2.3.1'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
    implementation 'androidx.coordinatorlayout:coordinatorlayout:1.1.0'
    implementation 'com.jjoe64:graphview:4.2.2'
}
```

Рисунок 3.3 – Налаштовані залежності

Структура будь-якого мобільного додатку містить дві ключові частини. Backend-частина, яка відповідає за функціональність та залишається невидимою

для користувачів, і Frontend-частина, яка відображає інтерфейс і представляється користувачеві під час взаємодії з додатком.

Ініційно розробляється інтерфейс, створюються всі необхідні елементи, з якими користувач буде взаємодіяти. Це є першим кроком у створенні додатку перед подальшим розгортанням його функціоналу.

3.2 Налаштування API

Для отримання ключу API нам потрібно зареєструватись на сайті OpenWeather та мати аккаунт в Google. Розберемо OpenWeatherAPI, нам потрібно створити новий ключ та обрати який тип запиту ви хочете робити. Наприклад, для отримання поточних погодних даних це може бути weather, а для прогнозу – forecast.

Специфіку кожного типу запиту можна знайти у документації OpenWeather API. Зазвичай вони включають URL, до якого додається ваш ключ API та параметри запиту (див. рис. 3.4).

```
        try{
                URL url = new URL("https://api.openweathermap.org/data/2.5/weather?q=" +
query + "&units=metric&appid=" + this.apiKey);
                connection = (HttpURLConnection) url.openConnection();
                connection.setRequestMethod("POST");
                connection.setRequestProperty("Content-Type",
                      "application/x-www-form-urlencoded");
                connection.setRequestProperty("Content-Length",
                      Integer.toString(urlParameters.getBytes().length));
                connection.setRequestProperty("Content-Language", "en-US");
                connection.setUseCaches(false);
                connection.setDoOutput(true);
                //Send request
                DataOutputStream wr = new DataOutputStream (
                      connection.getOutputStream());
                wr.writeBytes(urlParameters);
                wr.close();
```

Рисунок 3.4 – Запит до OpenWeatherAPI

Отриманий ключ API записуємо у конфігураційний файл додатка або в налаштування, де використовується OpenWeather API, переглянемо основні запити до цього API (див. рис. 3.5):

– отримання погодних даних за назвою міста;
– прогноз на кілька днів за назвою міста;
– поточні погодні умови за координатами;
– прогноз на кілька днів за координатами.

```
https://api.openweathermap.org/data/2.5/weather?q={city}&appid={your_api_key}
https://api.openweathermap.org/data/2.5/forecast?q={city}&appid={your_api_key}
https://api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}&appid={your_api_key}
https://api.openweathermap.org/data/2.5/forecast?lat={lat}&lon={lon}&appid={your_api_key}
```

Рисунок 3.5 – Приклади основних запитів

На рисунку 3.5 {city} вказує на назву міста,{your_api_key} на ключ який отримали на сайті сервісу, {lat} і {lon} на широту та довготу.

За допомогою цього API ми можемо отримати наступну інформацію:

– температура повітря;
– опис стану погоди (хмарно, ясно, дощ, тощо);
– вологість повітря;
– швидкість вітру;
– тиск;
– географічні координати місця;
– час сходу та заходу сонця;
– зображення погоди (іконка), яку можна використовувати для візуального відображення стану погоди.

Приклад коду який використовує бібліотеку JSON для обробки відповіді в форматі JSON від сервера OpenWeather API (див. рис. 3.6).

```
InputStream is = connection.getInputStream();
        BufferedReader rd = new BufferedReader(new InputStreamReader(is));
        StringBuilder response = new StringBuilder();
        String line;
        while ((line = rd.readLine()) != null) {
           response.append(line);
           response.append('\r');
        }
        rd.close();
        res = response.toString();
        JSONObject json = new JSONObject(res);
        this.city = json.getString("name");
        JSONObject weather = (JSONObject)json.getJSONArray("weather").get(0);
        this.description = weather.getString("description");
        this.icon = weather.getString("icon");
}
```

Рисунок 3.6 – Обробка відповіді

3.3 Реалізація інтерфейсу

Після визначення дизайну мобільного додатку розпочинається розробка його інтерфейсу. Визначаються ключові елементи інтерфейсу, такі як кнопки, поля введення, та інші, а також обираються візуальні компоненти, наприклад, картинка заднього фону та іконка кнопки пошуку.

Після створення проєкту за обраним шаблоном важливим етапом є завантаження елементів, які будуть використовуватися в інтерфейсі мобільного додатку. Ці елементи зазвичай знаходяться у папці drawable, розташованій у каталозі ресурсів (res). Основний файл для розробки фронтенду – це activity_main.xml, який представляє собою шаблон Empty Activity.

activity_main.xml містить текстову частину з кодом на мові XML, який можна редагувати для додавання, вилучення або зміни елементів інтерфейсу. Зручність цього файлу полягає в тому, що всі внесені зміни відображаються у режимі дизайну в реальному часі, що дозволяє програмісту візуально спостерігати, як код впливає на вигляд інтерфейсу.

Розташування та розміри кожного елемента, такі як кнопки, текстові поля чи зображення, визначаються у коді XML. Наприклад, параметри для іконки міста можуть включати ширину та висоту у вимірах «dp», відстань від нижньої частини карточки тощо. Це робить процес розробки інтерфейсу більш гнучким та дозволяє точно налаштувати розташування та вигляд кожного елемента.

У коді визначаються параметри всіх елементів, такі як їх розташування та розмір. Наприклад, параметри для іконки міста включають ширину та висоту у спеціальних одиницях розміру «dp», відстань від нижньої частини карточки та шлях до відповідної картинки (див. рис. 3.7).

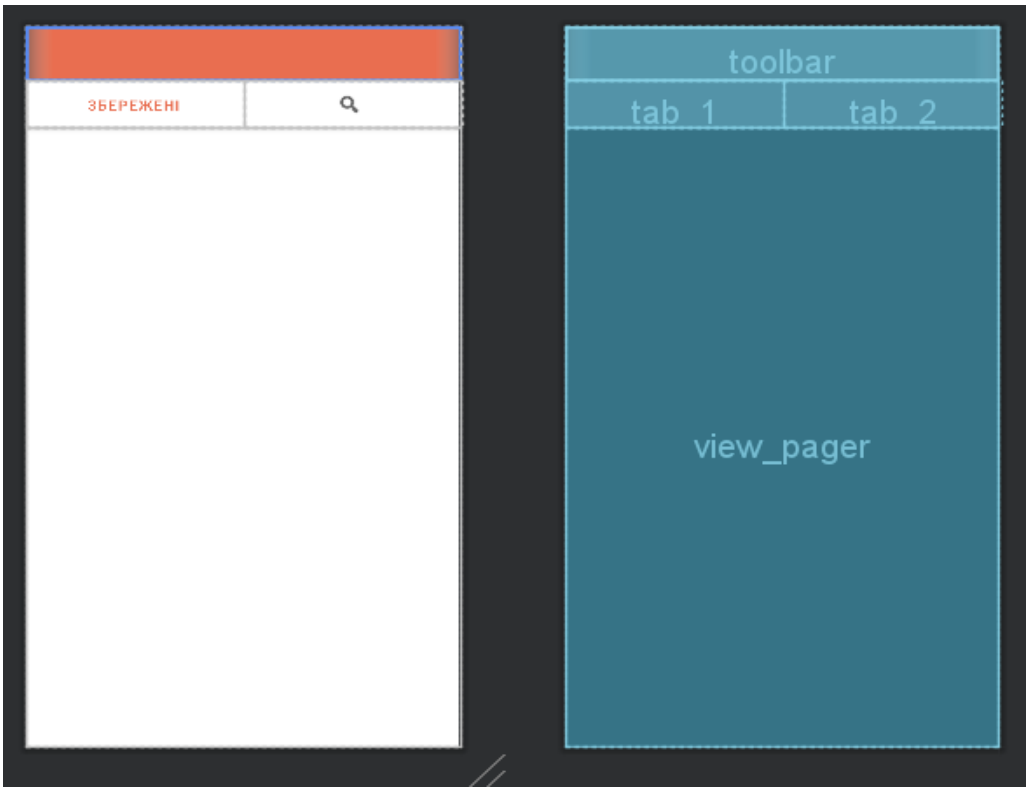

Рисунок 3.7 – Основний екран

Такий підхід до визначення розташування та розмірів елементів інтерфейсу в коді XML дозволяє розробникам точно налаштовувати кожен елемент під різні мобільні пристрої. Використання вимірів у «dp» для параметрів, таких як ширина та висота іконки міста, дозволяє створювати адаптивний інтерфейс, який буде однаково ефективно виглядати на різних екранах.

Розташування та вигляд кожного елемента може бути точно визначено, враховуючи вимоги до дизайну. Однак, крім цього, розробники також можуть скористатися режимом blueprint, який відображає розташування елементів для полегшення процесу розробки (див. рис. 3.8). Цей режим дозволяє швидко переглядати, як розміщені елементи на екрані, що робить візуальну компоновку інтерфейсу більш інтуїтивно зрозумілою та зручною для вирішення потенційних проблем з розташуванням. Таким чином, розробники можуть швидше досягати високої єдності та якості інтерфейсу на різних пристроях.

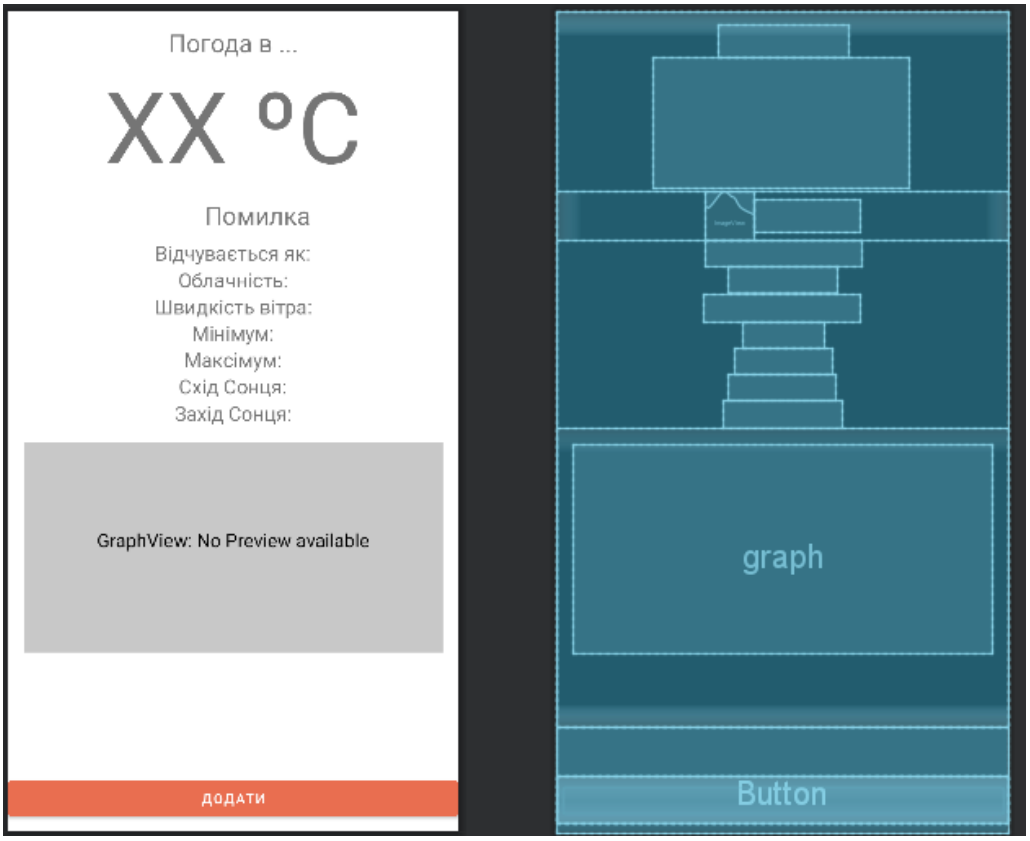

Рисунок 3.8 – Інтерфейс виводу інформації про погоду

Інтерфейс включає іконку статусу погоди, поле для вводу, кнопку перегляду розташування та 8 полів для відображення інформації про погоду. Всі елементи ретельно центруються для забезпечення єдність вигляду на різних пристроях. Розглядаємо інтерфейс у режимі blueprint для легшого сприйняття розташування елементів.

3.4 Логіка додатку

Після визначення алгоритму роботи програми та додавання всіх елементів інтерфейсу час розпочати розробку функціональностей та логіки мобільного додатку. Код написаний на мові Java в середовищі Android Studio, у якості основного шаблону використовується модель MVC.

Розробка розпочалась з підключення необхідних бібліотек, що відбулося через команди імпорту.

Ці бібліотеки використовуються для роботи з URL-запитами та обробки отриманих даних у форматі JSON. Після імпортування бібліотек визначається головний клас MainActivity, в якому будуть розміщені всі команди.

Пройдемося по основним класам: MainActivity: У методі onCreate, який викликається при створенні активності, встановлюється графічний інтерфейс з файлу activity_main.xml за допомогою методу setContentView. Здійснюється також зв’язок з елементами графічного інтерфейсу через їхні id.

Як видно, головний активіті додатку є вкладеним активіті з вкладками, користувач може перемикатися між вкладками, проводячи або натисканням на відповідну вкладку.

Розглянемо фрагменти всередині головного активіті.

FavsFragment. Перший фрагмент головного екрану (MainActivity) містить список улюблених місць для перевірки погоди без необхідності пошуку. Як видно, користувач може додавати або видаляти більше місць. Ці дані зберігаються постійно.

SearchFragment. Другий фрагмент головного екрану (MainActivity) представляє собою форму пошуку, яка виконує HTTP-запит до API на основі вказаного місця. Якщо для запитаного місця немає доступних результатів, користувача буде проінформовано.

WeatherDisplayActivity. Якщо місце успішно знаходиться під час пошуку або якщо натиснута улюблена локація, відображається поточний стан погоди, а також інші цікаві дані.

Кнопка «Додати до улюблених» буде відображена в залежності від того, чи місце вже внесено до обраного чи ні (див. рис. 3.9).

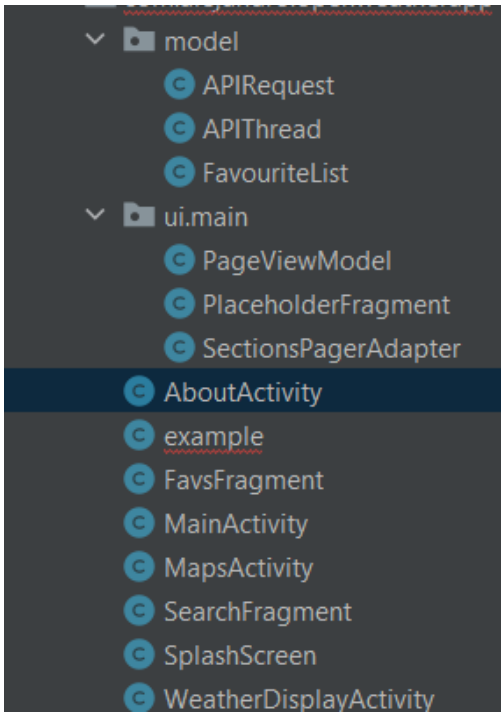

Рисунок 3.9 – Структура додатку

Спосіб збереження даних: Конкретно, дані списку зберігаються у файлі з назвою «favslist.txt» у внутрішньому сховищі пристрою. Метод saveToFile() записує дані зі списку у цей файл, а readFromFile() читає дані з файлу та відновлює їх у список. Такий підхід дозволяє забезпечити збереження улюблених місць між сеансами використання додатку та забезпечує сталість даних.

Реалізація шаблону MVC у додатку.

Шаблон MVC реалізований в цьому додатку за допомогою трьох основних класів згідно діаграми класів у другому розділі(): APIRequest виступить як Модель (Model), FavouriteList як Контролер (Controller), а APIThread як Інтерфейс (View) для відображення та взаємодії з користувачем. Далі наведено докладний опис даних класів.

Клас APIRequest відповідає за взаємодію з OpenWeatherAPI, використовується для виконання запитів до API OpenWeather та отримання

даних про погоду для міста яке ми попередньо обрали. Основні етапи роботи цього класу включають:

– встановлення з’єднання з сервером OpenWeatherMap через URL, включаючи параметри запиту, такі як місто, одиниці вимірювання (metric), та ключ API;
– надсилання POST-запиту до сервера з використанням HttpURLConnection;
– отримання відповіді від сервера, яка представляє собою JSON-об’єкт, що містить інформацію про погоду;
– розбір отриманого JSON для отримання різних параметрів погоди, таких як назва міста, опис, температура, вологість, швидкість вітру та інші;
– збереження отриманих даних у відповідні змінні класу.

Цей клас також містить методи доступу (get та set) для отримання та встановлення різних параметрів погоди (див. рис. 3.10). Наприклад, метод getCity повертає назву міста, а метод setCity встановлює нове значення для цього параметра. Нижче зображені параметри які ми оброблюємо в додатку.

```
JSONObject json = new JSONObject(res);
        this.city = json.getString("name");
        JSONObject weather = (JSONObject)json.getJSONArray("weather").get(0);
        this.description = weather.getString("description");
        this.icon = weather.getString("icon");
        this.temperature = ((JSONObject)json.get("main")).getString("temp");
        this.feelsLike = ((JSONObject)json.get("main")).getString("feels_like");
        this.minimum = ((JSONObject)json.get("main")).getString("temp_min");
        this.maximum = ((JSONObject)json.get("main")).getString("temp_max");
        this.sunrise = ((JSONObject)json.get("sys")).getString("sunrise");
        this.sunset = ((JSONObject)json.get("sys")).getString("sunset");
        this.humidity = ((JSONObject)json.get("main")).getString("humidity");
        this.windSpeed = ((JSONObject)json.get("wind")).getString("speed");
        this.offset = Integer.parseInt(json.getString("timezone"));
        this.lat = Double.parseDouble(((JSONObject)json.get("coord")).getString("lat"));
        this.lng = Double.parseDouble(((JSONObject)json.get("coord")).getString("lon"));
```

Рисунок 3.10 – Можливі погодні параметри API

Загалом, цей клас відповідає за виконання запитів до API та обробку отриманих даних про погоду.

Клас FavouriteList, представляє собою управління списком улюблених елементів, які зберігаються в мобільному пристрої. Давайте розглянемо основні елементи цього класу: Конструктор FavouriteList(Context context): Ініціалізує об’єкт класу, отримуючи Context для доступу до ресурсів та файлової системи. FILENAME: Константа, що визначає ім’я файлу для збереження списку улюблених елементів (favslist.txt). Цей клас використовується для зручного управління та збереження списку улюблених елементів в додатку (див. рис. 3.11).

```
public void saveToFile() throws IOException {
    FileOutputStream outputStream = mContext.openFileOutput(FILENAME,
        Context.MODE_PRIVATE);
    writeListToStream(outputStream);
}
public void readFromFile() throws IOException {
    FileInputStream inputStream = mContext.openFileInput(FILENAME);
    try (BufferedReader reader = new BufferedReader(new
        InputStreamReader(inputStream))) {
      mFavsList.clear();
      String line;
      while ((line = reader.readLine()) != null) {
        mFavsList.add(line);
      }
    }
    catch (FileNotFoundException ex) {
    }
}
```

Рисунок 3.11 – Зчитування та збереження міст

Клас APIThread, наслідується від класу Thread і використовується для асинхронного виконання операцій API-запиту та завантаження зображення із відповіді. Мета його створення – використання для виклику API-запиту в окремому потоці для уникнення блокування основного потоку і завантаження зображення (див. рис. 3.12), яке використовується у мобільному додатку для відображення погоди. Потрібно звернути увагу на те, що використання окремого

потоку для виконання API-запитів підвищує ефективність додатку та забезпечує зручність взаємодії з користувачем. Відсутність блокування основного потоку гарантує, що додаток залишається швидким навіть у випадку операцій, що займають багато часу. Завдяки цьому підходу використання додатку стає більш ефективним та зручним для користувача.

```
        public void run() {
            super.run();
            this.request = new APIRequest(query);
            this.imageBitmap = getImageBitmap("https://openweathermap.org/img/wn/" +
request.getIcon() + ".png");
        }
        private Bitmap getImageBitmap(String url) {
            Bitmap bm = null;
            try {
                URL aURL = new URL(url);
                URLConnection conn = aURL.openConnection();
                conn.connect();
                InputStream is = conn.getInputStream();
                BufferedInputStream bis = new BufferedInputStream(is);
                bm = BitmapFactory.decodeStream(bis);
                bis.close();
                is.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
            return bm;
        }
```

Рисунок 3.12 – Отримання зображення з API

Тут відбувається створення об’єкту URL на основі переданого URL-адреси зображення:

– відкриття з’єднання за допомогою openConnection();

– з’єднання ініціалізується та підключається до вказаного URL;

– отримання потоку введення (InputStream) з з’єднання;

– створення потоку введення (BufferedInputStream);

– декодування потоку за допомогою BitmapFactory.decodeStream(bis).

Реалізація запиту через стандартну бібліотеку HttpURLConnection сервісу OpenweatherAPI (див. рис. 3.13).

```
URL url = new URL("https://api.openweathermap.org/data/2.5/onecall" + +query + "&units=metric&appid=" +"?lat="+ "&lon=" +"&exclude=current,minutely,hourly" + this.apiKey);
```

Рисунок 3.13 – Виборчий виклик

Треба відмітити що в сучасній розробці виклики краще організовувати за допомогою retrofit чи іншої альтернативи.

Передача параметрів у графік за допомогою бібліотеки GraphView від jjjoe64 (див. рис. 3.14).

```
private void updateGraph(GraphView graphView, List<WeatherData> weatherDataList)
{
   LineGraphSeries<DataPoint> series = new LineGraphSeries<>();
   Calendar calendar = Calendar.getInstance();
   for (WeatherData weatherData : weatherDataList) {
      Date date = new Date(weatherData.getTimestamp() * 1000);
      calendar.setTime(date);
      double x = calendar.get(Calendar.DAY_OF_MONTH);
      double y1 = weatherData.getTemperature().getMinTemperature();
      double y2 = weatherData.getTemperature().getMaxTemperature();
      series.appendData(new DataPoint(x, y1), true, weatherDataList.size());
      series.appendData(new DataPoint(x, y2), true, weatherDataList.size());
   }

   graphView.addSeries(series);
}
```

Рисунок 3.14 – Параметри графіку

Цей метод updateGraph використовується для оновлення графіка в GraphView на основі списку WeatherData. Основна ідея полягає в тому, що для кожного об’єкта WeatherData зі списку вираховується дата, мінімальна та максимальна температури, і ці значення додаються до серії точок графіка.

Основні етапи методу:

– створення порожньої серії для графіка (LineGraphSeries<DataPoint> series);

– проходження по кожному об’єкту WeatherData у списку;

– отримання дати з WeatherData та конвертація її в день місяця;
– отримання мінімальної та максимальної температур з WeatherData;
– додавання двох точок графіка для кожного дня: одна точка з мінімальною температурою (y1), інша з максимальною температурою (y2);
– додавання серії точок до GraphView.

3.5 Огляд роботи додатку

Після запуску додатку ми потрапляємо на головний екран, на якому розташовані дві вкладки. За замовчуванням відкрита вкладка «Збережені» (див. рис. 3.15), де ми маємо швидкий доступ до перегляду потрібних нам міст.

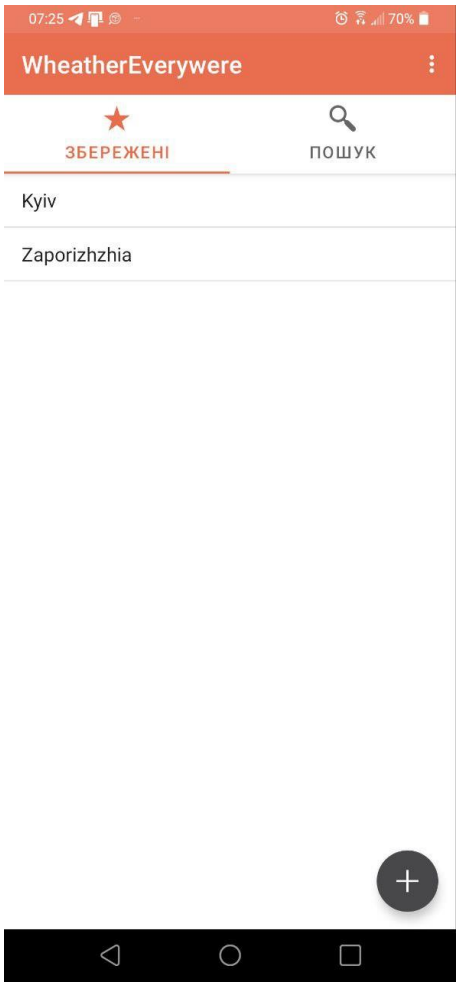

Рисунок 3.15 – Головний екран

Ця функція дозволяє користувачеві зручно відстежувати та переглядати інформацію про погоду для попередньо збережених міст, забезпечуючи легкий і швидкий доступ до важливої інформації.

У другій вкладці «Пошук» (див. рис. 3.16), користувач може ввести назву міста, натиснути кнопку, і отримати повну інформацію про погоду у вказаному місті. Ця функціональність надає зручний інтерфейс для користувача, який може швидко та легко отримати інформацію про погоду для будь-якого міста, необхідного йому. Такий підхід робить додаток більш універсальним і корисним для широкого кола користувачів, незалежно від їхнього розташування.

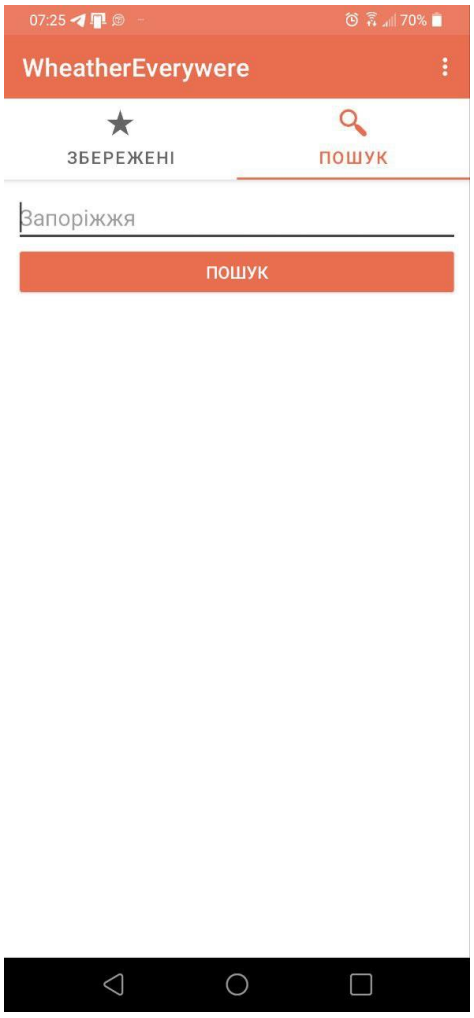

Рисунок 3.16 – Екран пошуку

Після вибору міста у вкладці «Збережені» або при ручному пошуку, користувач переходить на вікно з інформацією про поточну погоду та графіком,

який відображає мінімальну та максимальну температуру впродовж наступних 7 днів (див. рис. 3.17). В цілому користувач може переглянути наступні погодні дані: поточну температуру разом з мінімальною та максимальною протягом дня, відсоток туманності, швидкість вітру, графічне зображення стану хмар, а також час сходу та заходу сонця. Температура виводиться від поточної дати, що надає зручний та зрозумілий для користувача перегляд прогнозу на найближчий тиждень. Ця функціональність покликана забезпечити користувача не лише поточною інформацією, а й прогнозом на майбутнє, що дозволяє більш ефективно планувати свої дії та взаємодіяти з додатком.

Рисунок 3.17 – Екран з виводом інформації

Таким чином детально розглянуто процес розробки мобільного додатку WeatherApp, зосереджуючись на області реалізації інтерфейсу користувача (Frontend) та логіки додатку (Backend). Детально описано створення проєкту,

вибір шаблону та використані інструменти під час тестування. Розглянуто процес додавання елементів на інтерфейс користувача, а також налаштування цих елементів за допомогою мови XML Важливим аспектом є зосередження на виборі оптимальних інструментів тестування та їхньому використанні для забезпечення якості розроблюваного продукту. Процес додавання та налаштування елементів інтерфейсу користувача розглядається в контексті покращення зручності взаємодії з додатком.

Розділ більш докладно розкриває розробку Backend-частини. Показано, які додаткові бібліотеки були імпортовані, та які саме змінні відповідають конкретним елементам. Розглянуто роботу кнопки пошуку, процес отримання та обробки інформації про погоду.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було проведено огляд технологій розробки Android-додатків, їх види, було проаналізовано API OpenWeather. В результаті цього аналізу було спроєктовано та реалізовано Android-застосунок.

Однією з ключових особливостей розробленої програмної системи стало вдале використання сервісу OpenWeather для реалізації функціоналу перегляду інформації про погоду.

У рамках реалізації застосунку використовувалось одне з популярних API OpenWeather, а використання бібліотеки GraphView від jjoe64 надало можливість відобразити мінімалістичний але зручний графік температури. Серед іншого, було впроваджено методи для взаємодії з API, зокрема такі як відправка запитів та їх обробка. Додатково було розроблено функціонал для побудови графіку який відображає мінімальну та максимальну температуру на тиждень.

Подальший розвиток розробки може бути пов’язаний з розширенням функціоналу застосунку та впровадженням нових можливостей для поліпшення користувацького досвіду. Також, розгляд можливостей інтеграції з іншими сервісами та вдосконалення процесів взаємодії користувачів з додатком є перспективними напрямками досліджень у майбутньому.

ПЕРЕЛІК ПОСИЛАНЬ

1. Horton J. Android Programming for Beginners. 2018. 766 p.
2. Типи мобільних додатків. URL: https://training.qatestlab.com/blog/technical-articles/types-of-mobile-applications (дата звернення: 25.07.2023).
3. Teague K., Gallicchio N. The Evolution of Meteorology. 2017. 272 p.
4. OpenWeather Guide. URL: https://openweathermap.org/guide (дата звернення: 10.08.2023).
5. About AccuWeather. URL: https://www.encyclopedia.com/books/politics-and-business-magazines/accuweather-inc (дата звернення: 11.08.2023).
6. WeatherStack FAQ. URL: https://weatherstack.com/faq (дата звернення: 13.08.2023).
7. Neil S. Android Studio Flamingo Essentials - Java Edition. 2023. 816 p.
8. MVC (Model View Controller) Architecture Pattern in Android. URL: https://www.geeksforgeeks.org/mvc-model-view-controller-architecture-pattern-in-android-with-example/ (дата звернення: 15.08.2023).
9. Application programming interface (API). URL: https://en.wikipedia.org/wiki/API (дата звернення: 18.08.2023).
10. Ludin S. Garza J. Learning HTTP/2. 2017. 84 p.
11. Gaitatzis T. Learn REST APIs. 2019. 109 p.
12. OpenWeather One Call Data Documentation. URL: https://openweathermap.org/api/one-call-3 (дата звернення: 24.09.2023).
13. Larman C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. 2004. 736 p.
14. Miles R., Hamilton K. Learning UML 2.0. 2006. 283 p.

ДОДАТОК А

Клас MainActivity

```
public class MainActivity extends AppCompatActivity {
    private ActivityMainBinding binding;
    private FavouriteList favsList;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        SectionsPagerAdapter sectionsPagerAdapter = new SectionsPagerAdapter(this,
getSupportFragmentManager());
        ViewPager viewPager = findViewById(R.id.view_pager);
        viewPager.setAdapter(sectionsPagerAdapter);
        TabLayout tabs = findViewById(R.id.tabs);
        tabs.setupWithViewPager(viewPager);
        tabs.getTabAt(0).setIcon(R.drawable.ic_star);
        tabs.getTabAt(1).setIcon(R.drawable.ic_search);
        favsList = new FavouriteList(this);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.context_menu, menu);
        return true;
    }
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
```

```
    int id = item.getItemId();
    if (id == R.id.about_us) {
        startActivity(new Intent(MainActivity.this,AboutActivity.class));
        return true;
    }
    return super.onOptionsItemSelected(item);
}
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == 0) {
        if(resultCode == Activity.RESULT_OK){
            favsList.addItem(data.getStringExtra("cityName"));
            try {
                favsList.saveToFile();
            }
            catch (IOException ex) {
                ex.printStackTrace();
            }
            FavsFragment.adapter.notifyDataSetChanged();
        }
        if (resultCode == Activity.RESULT_CANCELED) {
        }
    }
} //onActivityResult
@Override
protected void onResume() {
    super.onResume();
    try {
        favsList.readFromFile();
```

```
            if(FavsFragment.adapter!=null){
                FavsFragment.adapter.notifyDataSetChanged();
            }
        }
        catch (IOException ex) {
            ex.printStackTrace();
        }
    }
    @Override
    protected void onPause() {
        super.onPause();
        try {
            favsList.saveToFile();
        }
        catch (IOException ex) {
            ex.printStackTrace();
        }
    }
    public FavouriteList getFavsList() {
        return favsList;
    }
}
```

ДОДАТОК Б

Клас WeatherDisplayActivity

```
public class WeatherDisplayActivity extends AppCompatActivity {
    private TextView cityTextView;
    private TextView tempTextView;
    private TextView humidityTextView;
    private TextView descTextView;
    private TextView windTextView;
    private TextView feelsLikeTextView;
    private TextView minTextView;
    private TextView maxTextView;
    private TextView sunriseTextView;
    private TextView sunsetTextView;
    private ImageView iconImageView;
    private Button addToFavButton;
    private APIThread apiThread;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_weather_display);
        cityTextView = findViewById(R.id.city_text_view);
        tempTextView = findViewById(R.id.temp_text_view);
        descTextView = findViewById(R.id.description_text_view);
        humidityTextView = findViewById(R.id.humidity_text_view);
        windTextView = findViewById(R.id.windspeed_text_view);
        feelsLikeTextView = findViewById(R.id.feels_like_text_view);
        minTextView = findViewById(R.id.min_text_view);
        maxTextView = findViewById(R.id.max_text_view);
```

```
        sunsetTextView = findViewById(R.id.sunset_text_view);
        sunriseTextView = findViewById(R.id.sunrise_text_view);
        iconImageView = findViewById(R.id.icon_image_view);
        addToFavButton = findViewById(R.id.add_to_fav);
        if(getIntent().getBooleanExtra("favsButton", false)){
            addToFavButton.setVisibility(View.VISIBLE);
        }
        apiThread = new APIThread(getIntent().getStringExtra("cityName"));
        apiThread.start();
        try {
            apiThread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        cityTextView.setText("Погода в "+apiThread.getRequest().getCity());
        tempTextView.setText(apiThread.getRequest().getTemperature() + "°C");
        descTextView.setText(apiThread.getRequest().getDescription());
        iconImageView.setImageBitmap(apiThread.getImageBitmap());
        humidityTextView.setText("Туманність:
"+apiThread.getRequest().getHumidity()+"%");
        windTextView.setText("Швидкість повітря:
"+apiThread.getRequest().getWindSpeed()+" km/h");
        feelsLikeTextView.setText("Відчувається:
"+apiThread.getRequest().getFeelsLike() + "°C");
        minTextView.setText("Мінімум: "+apiThread.getRequest().getMinimum() +
"°C");
        maxTextView.setText("Максімум: "+apiThread.getRequest().getMaximum() +
"°C");
        DateFormat obj = new SimpleDateFormat("HH:mm");
        Date sunrise = new
```

```
Date(Long.parseLong(apiThread.getRequest().getSunrise()+"000"));
        Date sunset = new
Date(Long.parseLong(apiThread.getRequest().getSunset()+"000"));
        sunriseTextView.setText("Схід: "+ obj.format(sunrise));
        sunsetTextView.setText("Захід: "+ obj.format(sunset));
    }
    public void onAddToFavClick(View view) {
        Intent returnIntent = new Intent();
        returnIntent.putExtra("cityName", getIntent().getStringExtra("cityName"));
        setResult(Activity.RESULT_OK,returnIntent);
        finish();
    }
    public void onLocationClick(View view) {
Toast.LENGTH_LONG).show();
        Intent intent = new Intent(WeatherDisplayActivity.this, MapsActivity.class);
        intent.putExtra("cityName", getIntent().getStringExtra("cityName"));
        intent.putExtra("lat", apiThread.getRequest().getLat());
        intent.putExtra("lng", apiThread.getRequest().getLng());
        startActivity(intent);
    }
}
```