МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)
16 червня 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавр

зі спеціальності 122 – Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Патріотичний додаток, прогноз погоди»
здобувача групи ІН-03 Темченко Андрія Миколайовича

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

_____Андрій ТЕМЧЕНКО
(підпис)

Керівник,
старший викладач кафедри
комп'ютерних наук,
Кандидат технічних наук Артем КОРОБОВ _____
(підпис)

Суми – 2024

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«Затверджую»

В.о. завідувача кафедри

_ _____ Ігор ШЕЛЕХОВ

(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»

здобувача групи ІН-03 Темченко Андрія Миколайовича

1. Тема роботи: « Патріотичний додаток, прогноз погоди »
   затверджую наказом по СумДУ від *«22» квітня 2024 р. № 0414-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 17 червня 2024 року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз актуальності розробки додатку для прогнозу погоди, конкурентоспроможності та цільової аудиторії, постановка й формування завдань. 2) Проектування системи, розробка дизайну, розробка функціоналу. 3) Огляд та вибір засобів програмної реалізації 4) Програмна реалізація системи додатку для інформування про погоду. 5) Аналіз результатів.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____
6. Консультанти до проекту (роботи), із зазначенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «__» _____ 20__ р.

Завдання прийняв на виконання _____ Керівник _____

(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, конкурентоспроможності, постановка й формування завдань дослідження*	10.04.2024 - 14.04.2024	
2	*Проектування системи, розробка дизайну.*	14.04.2024 – 22.04.2024	
3	*Огляд та вибір інструментів програмної реалізації*	22.04.2024 – 26.04.2024	

4	*Розробка інформаційної системи додатку*	26.05.2024 - 08.06.2024	
5	*Аналіз отриманих результатів*	08.06.2024 – 10.06.2024	
6	*Оформлення пояснювальної записки до кваліфікаційної роботи*	30.05.2024 – 16.06.2024	

Здобувач вищої освіти _____ Керівник _____

(підпис) (підпис)

АНОТАЦІЯ

Записка: 58 стор., 12 рис., 0 табл., 1 додатки, 10 використаних джерел.

Обґрунтування актуальності теми роботи – розробка патріотичного додатку, що надає прогноз погоди, сприяє зміцненню національної ідентичності та підвищенню обізнаності громадян про погодні умови в різних регіонах країни. Використання патріотичних символів та мотивів у дизайні і функціоналі таких додатків сприяє формуванню позитивного іміджу держави та вихованню патріотичних почуттів серед молоді. Крім того, технологічний розвиток мобільних додатків надає можливості для покращення якості життя населення шляхом оперативного та точного надання інформації про погоду. Це особливо актуально в умовах змін клімату та непередбачуваних погодних явищ, коли своєчасне попередження може запобігти збиткам та врятувати життя.

Об’єкт дослідження - патріотичний мобільний додаток, який надає користувачам актуальний прогноз погоди з використанням патріотичних символів та мотивів.

Мета роботи – дослідження, аналіз та розробка Android додатку для прогнозу погоди, який надаватиме користувачам актуальну та точну інформацію про погодні умови з використанням патріотичних символів і мотивів. Додаток сприятиме зміцненню національної ідентичності, забезпечуючи зручний інтерфейс для перегляду прогнозів[4].

Методи дослідження - аналіз існуючих додатків про перегляд погоди, інструменти розробки та проектування.

Результати - розроблено застосунок, з зручним та зрозумілим функціоналом, точною інформацією про погодні умови та швидкою роботою, допомога якого знадобиться, як звичайним громадянам, так і військовим за для захисту країни.

АДАПТИВНИЙ ДИЗАЙН, ІНФОРМАЦІЙНА СИСТЕМА, KOTLIN, XML, API

ЗМІСТ

ВСТУП

Прогноз погоди відіграє важливу роль у сучасному суспільстві і є не лише джерелом інформації, а й засобом підготовки до повсякденних подій та планування діяльності. Точний прогноз здатен впливати на емоції, настрій і навіть фізіологічний стан людей, адже знання про майбутню погоду допомагає уникнути небажаних сюрпризів. Прогноз погоди – це універсальна інформація, яка дозволяє людям з різних куточків світу знаходити спільну мову та розуміти природні умови одне одного. Він також відіграє важливу роль у розвитку суспільства, особливо в таких сферах, як сільське господарство, транспорт і туризм, впливаючи на їхню ефективність і безпеку.

Додаток прогнозу погоди став невід'ємною частиною повсякденного життя багатьох українців у сучасному світі. Згідно з даними досліджень, понад 90% опитаної громади регулярно користуються прогнозами погоди, щоб планувати свій день і підготуватися до погодних умов[10]. Такі додатки не лише забезпечують зручний доступ до актуальної інформації, але й інтегрують патріотичні елементи, що сприяє підвищенню національної свідомості та гордості за свою країну. Ці додатки також сприяють екологічній освіті, надаючи користувачам інформацію про зміну клімату та екологічні ініціативи.

Завдяки патріотичним додаткам прогнозу погоди, люди можуть бути в курсі погодних змін в будь-який час і в будь-якому місці, що робить інформацію більш доступною та інтегрованою в повсякденне життя. Це також відкрило можливість українським розробникам ділитися своїми інноваціями з аудиторією по всьому світу, демонструючи високий рівень технологічних досягнень та культурної самобутності України.

Постановки задачі

Робота присвячена розробці патріотичного додатку прогнозу погоди, що надасть користувачам зручний доступ до актуальної метеорологічної інформації, інтегровані патріотичні елементи та додаткові функції, які покращують досвід користувача.

Патріотичний додаток прогнозу погоди — це програма, яка дозволяє користувачам отримувати точний прогноз погоди, налаштовувати персональні сповіщення, отримувати інформацію про кліматичні умови та екологічні ініціативи, а також взаємодіяти з інтуїтивно зрозумілим інтерфейсом, оформленим у національних кольорах і стилі.

У цьому звіті буде розглянуто основні етапи розробки додатку. Спочатку я проведу аналітичний огляд, щоб визначити вимоги до програми, згідно з якими можна визначити приблизні обсяги роботи та вміло розподілити час, що буде витрачено на той чи інший модуль роботи, а також проаналізувати її конкурентоспроможність. Далі, я розгляну проектування системи, включаючи дизайн інтерфейсів для різних частин додатку та структури системи. У завершальному розділі я розгляну інформаційне та програмне забезпечення системи, що включає вибір технологій та опис програмної реалізації.

У процесі розробки патріотичного додатку прогнозу погоди ми будемо враховувати такі ключові аспекти:

- **Аналітичний огляд:** на цьому етапі я визначу основні вимоги до програми, враховуючи потреби користувачів та цільову аудиторію. Аналіз конкурентоспроможності допоможе визначити унікальні особливості, які виділяють наш додаток на тлі інших погодних програм.
- **Проектування системи:** проектування інтерфейсу є критично важливим для створення зручного користувацького досвіду. Я розроблю інтерфейс для головного екрану, вікна прогнозу та інших вікон, що можуть знадобитися користувачу при користуванні додатком. Також буде розроблено діаграму класів, яка відобразить структуру додатку.

- **Інформаційне та програмне забезпечення системи:** на цьому етапі я розповім про те, які саме інструменти та технології для розробки додатку були обрані. Буде описано програмну реалізацію з акцентом на важливі деталі, які забезпечують ефективність та стабільність додатку.

1. АНАЛІТИЧНИЙ ОГЛЯД

1.1 Основні етапи створення патріотичного додатку прогнозу погоди

Додатки прогнозу погоди займають важливе місце в сучасному суспільстві, оскільки вони надають користувачам доступ до актуальної метеорологічної інформації, дозволяючи планувати свою діяльність і уникати небажаних сюрпризів від погоди. Вони роблять прогноз погоди більш доступним і інтегрованим у повсякденне життя, дозволяючи людям отримувати інформацію будь-де та будь-коли. Для того щоб розробити такий додаток, потрібно дотримуватися таких етапів:

- **Етап визначення ідеї та планування –** це є першим кроком визначення концепції додатку, адже додатки можуть мати різну спрямованість, такі як розважальні, бізнес-додатки, освітні додатки тощо. Також на початковому етапі важливо скласти графік розробки, встановити дедлайни та створити покроковий план реалізації проєкту.
- **Етап проектування –** це другий, але не менш важливий етап у створенні мобільного додатку. На цьому етапі закладається основа, на якій буде будуватися весь проєкт. Тут визначається не лише зовнішній вигляд додатку, але й його функціонування. Включає розробку дизайну інтерфейсу, визначення архітектури системи та технічних характеристик.
- **Етап розробки** – це третій, ключовий етап створення мобільного додатку, де ідеї перетворюються в реальність. На цьому етапі пишеться програмний код, створюються функції та модулі, які забезпечуватимуть роботу додатку. Особлива увага приділяється точності прогнозу, інтеграції з метеорологічними сервісами та знаходження якісного API[8].
- **Тестування** – це один з найважливіших етапів створення мобільного

додатку, адже на цьому етапі визначається, наскільки додаток готовий до випуску та використання. Проводиться виявлення та виправлення помилок, тестується продуктивність та загальна якість роботи додатку. Важливо виконати різні види тестування, включаючи функціональне тестування, перевірку на різних пристроях та операційних системах, а також оцінку користувацького інтерфейсу.

- **Запуск та публікація** – цей етап є практично останнім кроком у процесі створення додатку. Якщо після успішного завершення усіх попередніх етапів і позитивного результату тестування наш додаток готовий до випуску, то настав час зробити його доступним для широкої аудиторії користувачів. Це включає підготовку додатку для публікації в магазині додатків, налаштування маркетингових матеріалів і відповіді на запити щодо випуску.
- **Оновлення та підтримка життєдіяльності** – цей етап вважається кінцевим у створенні додатку, але він фактично є постійним процесом у житті нашого застосунку. Для того, щоб забезпечити постійну актуальність та задоволення потреб користувачів, необхідно буде виправляти помилки, додавати новий функціонал, розробляти оновлення і налагоджувати зворотній зв’язок та підтримку користувачів. Такий підхід дозволяє тримати додаток у цілковитому порядку та забезпечувати високу якість користування протягом усього його життєвого циклу.

1.2 Аналіз вимог

Мобільні додатки є невід'ємною частиною сучасного життя, що задовольняють потреби, бажання та запити користувачів, сприяючи комфортному способу життя. Вони охоплюють широкий спектр сфер, від послуг доставки і таксі до державних та фінансових сервісів, а також від мобільних ігор і соціальних мереж до освітніх платформ і рішень для бізнесу. Клієнти та користувачі завжди оцінюють зручність та інтуїтивно зрозумілий інтерфейс мобільних додатків. Вони віддають перевагу додаткам, які забезпечують стабільність роботи, швидку відповідь на взаємодію, мінімалістичний інтерфейс без інформаційного перевантаження,

персоналізацію та забезпечують безпеку даних.

Отже, визначимо основні вимоги до додатку прогнозу погоди:

Інтерфейс – додаток має мати зрозумілий та інтуїтивно зрозумілий інтерфейс для всіх користувачів. Він повинен забезпечити просту навігацію і контрастність, щоб ефективно взаємодіяти з елементами. Весь текст в додатку має бути представлений англійською мовою для міжнародної доступності.

Адаптивність - додаток повинен бути сумісним з різними пристроями на базі Android і оптимізованим для різних розмірів екранів, що є популярними серед користувачів.

Виявлення за місцезнаходженням - цей функціонал дозволяє додатку автоматично визначати поточне місцезнаходження користувача за допомогою GPS або іншого засобу локації на пристрої. Це забезпечує точність та зручність для користувачів, оскільки додаток може автоматично адаптувати інформацію до місцевих умов та прогнозів.

Мануальний пошук по населеному пункту - ця функція дозволяє користувачам вручну вибирати населений пункт чи місце, для якого вони хочуть отримати прогноз погоди. Вона надає користувачам можливість ввести назву місця, щоб отримати актуальну інформацію про погоду для обраного регіону.

Ці дві функції разом забезпечують користувачам зручність в доступі до погодної інформації, незалежно від того, чи вони використовують автоматичне визначення місцезнаходження або використовують мануальний пошук за населеним пунктом.

Безпека - додаток повинен забезпечувати захист персональних даних користувачів і мати доступ до файлів та місцезнаходження тільки з їхнього дозволу.

Ці критерії допоможуть зрозуміти вимоги, необхідні для створення функціонального, зручного, безпечного та привабливого для користувачів додатку, що поліпшує їхній досвід з прогнозу погоди.

1.3 Аналіз конкурентоспроможності

Аналіз конкурентоспроможності є важливим етапом для оцінки можливостей та переваг в сегменті ринку. Цей процес дозволяє розробникам отримати усвідомлення про сильні та слабкі сторони конкурентів, а також ідентифікувати можливості для покращення свого продукту. Для аналізу конкурентоспроможності було вибрано наступні популярні додатки для прогнозу погоди:

- Weather Channel
- AccuWeather
- Weather Underground

Weather Channel

Weather Channel - це один із найпопулярніших додатків для прогнозу погоди на Android. Цей мобільний застосунок надає широкий вибір функцій без реклами, включаючи детальні прогнози, розширені карти, повідомлення про небезпеку погоди, інтерактивні графіки та зміну одиниць вимірювання. Приклад вигляду додатку можна побачити на рис 1.3.1 (фото були взяті зі сторінки додатку в google play, посилання:
https://play.google.com/store/apps/details?id=com.weather.Weather&hl=en).

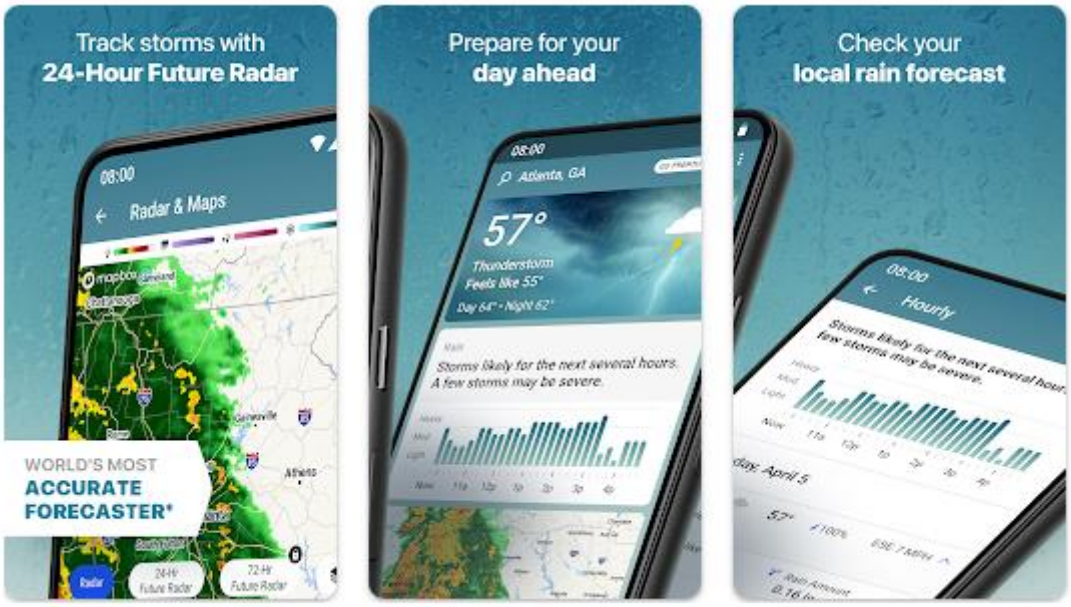


Рис 1.3.1 – Інтерфейс та функції додатку Weather Channel

З основних переваг Weather Channel має:

- **Сильні сторони:**

1. Широкий охоплення різноманітних місць і прогнозів погоди:

Weather Channel забезпечує користувачів широким спектром інформації щодо погоди по всьому світу. Додаток включає прогнози для міст, селищ та регіонів у багатьох країнах, що робить його вельми корисним для мандрівників, подорожуючих бізнесменів та просто цікавих осіб, які хочуть бути в курсі погодних умов у будь-якому місці.

2. Надійна точність прогнозу:

Weather Channel славиться своєю точністю прогнозу погоди. Він використовує передові технології та алгоритми для збору, обробки та аналізу погодних даних з різних джерел, що дозволяє надавати користувачам актуальну і достовірну інформацію про температуру, опади, вітер та інші погодні умови.

Ці дві ключові сильні сторони роблять Weather Channel одним з найпопулярніших і надійних додатків для отримання прогнозу погоди, що відповідає насущним потребам користувачів у всьому світі.

Розглянемо детальніше слабкі сторони додатку Weather Channel, зокрема інтерфейс:

1. Перенавантажений інтерфейс з інформацією:

Однією з основних слабких сторін додатку Weather Channel є його інтерфейс, який може виглядати перенавантаженим з інформацією. У зв'язку з широким охопленням погодних умов та великою кількістю функцій, доступних для користувачів, інтерфейс додатку може здаватися дещо складним для розуміння на перший погляд.

2. Потреба в оптимізації взаємодії з користувачем:

В разі перенавантаженого інтерфейсу може виникнути проблема з оптимізацією взаємодії з користувачем. Важливо зробити інформацію більш доступною та легко зрозумілою шляхом зменшення кількості тексту та додаткових функцій на головному екрані, а також впровадження інтуїтивно зрозумілих елементів управління.

3. Потенційне перевантаження візуальної частини:

Інтерфейс може бути також схильний до перевантаження візуальної частини, що може змішувати користувачів та ускладнювати сприйняття важливої інформації. Необхідно збалансувати кількість візуальних елементів та важливих даних, щоб покращити зручність використання додатку.

Подумаємо про можливості для покращення додатку Weather Channel через адаптивніший інтерфейс:

1. Персоналізація користувацького досвіду:

Однією з ключових можливостей для покращення є впровадження більш адаптивного інтерфейсу. Це дозволить користувачам налаштовувати додаток згідно зі своїми особистими вподобаннями та потребами. Наприклад, можна розглянути можливість вибору відображення конкретних типів погоди або пріоритетних метеопараметрів на головному екрані.

2. Налаштування відповідно до місцезнаходження:

Інтерфейс може стати більш адаптивним шляхом автоматичного визначення місцезнаходження користувача і надання актуальної інформації про погоду в цьому регіоні. Це може включати місцеві прогнози, попередження про погодні умови та поради щодо погоди, специфічні для конкретного місця.

3. Персоналізовані сповіщення та поради:

Вдосконалення можливостей персоналізації також може охоплювати індивідуальні сповіщення про погоду та персоналізовані поради з погоди. Наприклад, додаток може запропонувати рекомендації щодо одягу або активностей на відкритому повітрі в залежності від прогнозу.

AccuWeather

AccuWeather - це один із провідних додатків для прогнозу погоди на платформі Android. В цьому додатку доступний широкий вибір метеорологічних інформаційних ресурсів безкоштовно, і він відомий своєю точністю та надійністю у передбаченні погоди. AccuWeather має майже всі необхідні функції для задоволення потреб користувачів, включаючи відображення поточних погодних умов, детальні годинні та щоденні прогнози, карту погоди, сповіщення про негоду, налаштування одиниць вимірювання, а також можливість персоналізації. Приклад вигляду додатку можна побачити на рис 1.3.2 (фото були взяті зі сторінки додатку в google play, посилання:

https://play.google.com/store/apps/details?id=com.accuweather.android).

Рис 1.3.2 – Інтерфейс та функції додатку AccuWeather

З основних переваг AccuWeather має:

- **Сильні сторони:**

1. Висока точність прогнозу: AccuWeather відомий своєю достовірністю у передбаченні погодних умов, що дозволяє користувачам отримувати актуальну інформацію.

2. Інтерактивні карти та графіки: Додаток пропонує користувачам можливість використовувати динамічні карти та графіки, які дозволяють візуалізувати погодні умови зручним і зрозумілим способом.

Розглянемо детальніше слабкі сторони додатку AccuWeather, зокрема інтерфейс:

Можливість відображення реклами: Деякі користувачі можуть відчути дискомфорт від рекламних матеріалів, які можуть впливати на їхній користувацький досвід і зручність використання додатку.

Подумаємо про можливості для покращення додатку AccuWeather через адаптивніший інтерфейс:

Розширення функціоналу для інтеграції з іншими платформами та сервісами: AccuWeather може покращити свій функціонал шляхом інтеграції з іншими платформами або сервісами, що дозволить користувачам отримувати доступ до погодних умов у різних контекстах та на різних пристроях.

Цей аналіз демонструє, що AccuWeather має сильні сторони у вигляді точних прогнозів та інтерактивних інструментів візуалізації, але також має певні можливості для покращення, зокрема управління рекламою та розширення інтеграційного функціоналу.

Weather Underground

Weather Underground - це популярний сервіс прогнозу погоди, який відомий завдяки активній спільноті користувачів, що надає зворотній зв'язок і відгуки, а також доступ до графіків та статистики погодних умов. Приклад вигляду додатку можна побачити на рис 1.3.3 (фото були взяті зі сторінки додатку в google play, посилання: https://play.google.com/store/apps/details?id=com.wunderground.android).

Рис 1.3.3 – Інтерфейс та функції додатку Weather Underground

З основних переваг Weather Underground має:

- **Сильні сторони:**

1. Спільнота користувачів: Weather Underground використовує активну спільноту користувачів, яка надає важливий зворотний зв'язок щодо якості прогнозів і функціональності додатку. Це дозволяє підтримувати актуальність і точність інформації.

2. Графіки та статистика: Користувачі мають можливість використовувати графіки та статистичні дані для детального аналізу погодних умов і їх змін на протязі часу, що допомагає в кращому розумінні погодних тенденцій.

Розглянемо детальніше слабкі сторони додатку Weather Underground, зокрема інтерфейс:

Можливість більшої залежності від користувачів для точності прогнозів: Залежність від користувальницького зворотного зв'язку може призвести до варіабельності у точності прогнозів, особливо у регіонах з меншою кількістю активних користувачів.

Подумаємо про можливості для покращення додатку Weather Underground через адаптивніший інтерфейс:

Підвищення зацікавленості користувачів через розширення спільнотних функцій: Weather Underground може покращити свою платформу шляхом розширення спільнотних функцій, таких як форуми, обговорення та спільні події, що збільшить взаємодію між користувачами і підвищить загальний інтерес до платформи.

Цей аналіз підкреслює важливість спільнотного аспекту для Weather Underground, вказує на його потенціал для покращення точності прогнозів через взаємодію з користувачами, а також наголошує на можливостях для розвитку спільнотних функцій для залучення більш широкої аудиторії.

2.ВИБІР МЕТОДУ ВИРІШЕННЯ ЗАДАЧІ

2.1 Вибір засобів програмної реалізації

Вибір інструментів програмної реалізації визначає основу проекту, забезпечуючи технічну базу та визначаючи можливості і обмеження кінцевого продукту. У моєму випадку для розробки додатку прогнозу погоди були обрані такі інструменти: Android Studio (версія - Android Studio Hedgehog | 2023.1.1 Patch 2), Kotlin, XML, Figma. Кожен з них має свої переваги, які роблять їх ідеальними для реалізації цього проекту[3].

Середовище розробки

Android Studio є офіційним інтегрованим середовищем розробки (IDE) для створення додатків на платформі Android. Воно пропонує широкий спектр інструментів для розробників, включаючи вбудовані засоби для налагодження, тестування та аналізу коду. Завдяки інтеграції з Gradle, Android Studio спрощує управління залежностями і процесом збірки проекту.

Рис 2.1.1 – Середовище розробки Android Studio

Основними перевагами Android Studio є:

Інтегроване середовище розробки (IDE): Android Studio надає розробникам повноцінне інтегроване середовище для створення Android-додатків з усіма необхідними інструментами і ресурсами.

Підтримка Kotlin і Java: Android Studio підтримує сучасну мову Kotlin і традиційну Java, що дозволяє розробникам вибирати найбільш підходящий інструмент для реалізації своїх проектів[1].

Емулятори Android: Інтеграція з Android Emulator дозволяє тестувати додатки на різних віртуальних пристроях з різними версіями Android, що

допомагає впевнитися в сумісності додатка з різними платформами.

Засоби для профілювання і налагодження: Android Studio надає потужні інструменти для аналізу продуктивності додатків, виявлення проблем швидкодії і оптимізації роботи програми.

Графічний дизайнер і редактор інтерфейсів: Інтеграція з Figma і іншими графічними інструментами дозволяє створювати і редагувати інтерфейси додатків безпосередньо в середовищі Android Studio.

Підтримка Android SDK і Google Play Services: Android Studio автоматично оновлює Android SDK і підтримує нові функції, що дозволяє використовувати останні можливості платформи Android.

Інтеграція з системами керування версіями: Підтримка систем керування версіями (наприклад, Git) дозволяє ефективно керувати використанням історії змін у проекті.

Підтримка мультиплатформеності: Можливість розробки не лише для телефонів, а й для планшетів, Android TV, Wear OS та інших платформ Android.

Спільнота розробників і документація: Android Studio має активну спільноту користувачів і багато документації, що дозволяє швидко отримувати відповіді на запитання і розв'язувати проблеми.

Розширені можливості налаштування: Велика кількість плагінів і додаткових інструментів дозволяє налаштовувати робоче середовище під конкретні потреби розробника.

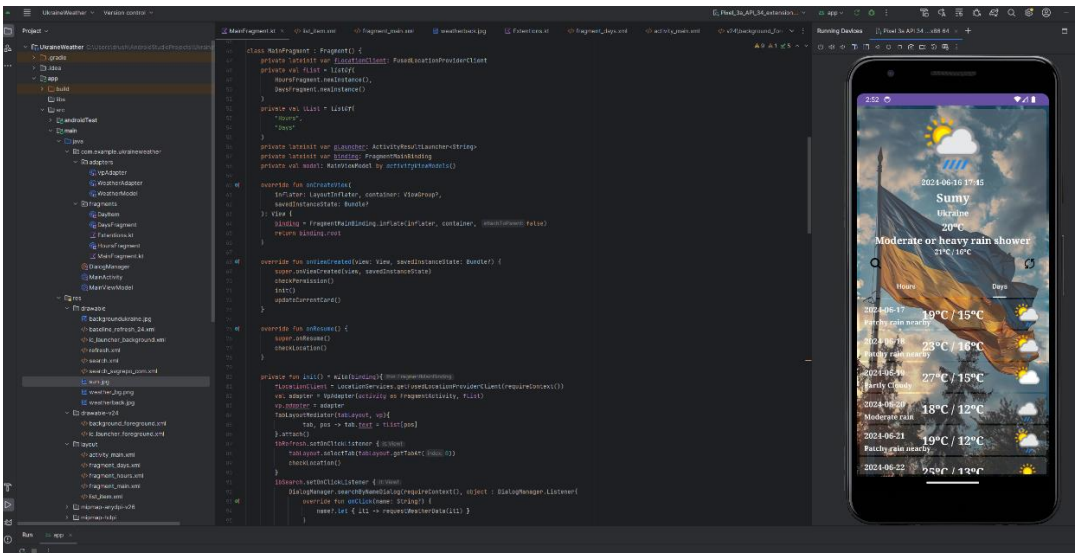


Рис 2.1.2 – Інтерфейс Android Studio

Ці переваги роблять Android Studio незамінним інструментом для професійної розробки Android-додатків, забезпечуючи швидку і ефективну роботу над проектами будь-якої складності.

Мова програмування

Мова програмування Kotlin вибрана як основна для цього проекту. Kotlin є офіційно підтримуваною мовою для розробки Android-додатків і має кілька переваг порівняно з Java, таких як більш компактний синтаксис, покращену безпеку типів і кращу сумісність з сучасними мовними функціями.[2]Це дозволяє розробникам писати більш надійний і зрозумілий код, що сприяє швидшому і безпомилковому розвитку додатка.

Рис 2.1.3 – Мова програмування Kotlin

- **Розширена безпека типів:** Kotlin дозволяє виявляти більше типових помилок ще на етапі компіляції, завдяки чому програми стають більш надійними і менше схильними до критичних помилок у реальному часі.
- **Менша кількість коду:** Синтаксис Kotlin є більш конденсованим порівняно з Java, що дозволяє писати менше коду для досягнення тих самих результатів, що сприяє підвищенню продуктивності розробників.

- **Підтримка нульових значень:** Kotlin пропонує безпечну обробку нульових значень, що дозволяє уникнути NullPointerException, які часто виникають в Java-коді.

- **Інтероперабельність з Java:** Kotlin може взаємодіяти без проблем з існуючим Java-кодом, що робить його ідеальним вибором для проектів, які потребують поступової міграції з Java.

- **Розширена підтримка Android і веб-фреймворків:** Kotlin підтримується Android і багатьма веб-фреймворками, що робить його ідеальним вибором для розробки API, які інтегруються з цими платформами.
- **Розширені функціональні можливості:** Kotlin підтримує функціональне програмування, включаючи зручні лямбда-вирази, функції вищих порядків і обробку подій, що спрощує написання чистого і ефективного коду.
- **Вбудована підтримка корутин:** Kotlin має вбудовану підтримку корутин для асинхронного програмування, що дозволяє ефективно керувати багатозадачними операціями і підвищує швидкодію додатку.
- **Активна підтримка Google:** Kotlin офіційно підтримується Google для розробки Android-додатків, що забезпечує актуальність і підтримку нових функцій платформи.

Ці переваги роблять Kotlin відмінним вибором для створення високоякісних, ефективних та надійних додатків для прогнозу погоди, що задовольняють високі стандарти безпеки і продуктивності.

Розмітка Інтерфейсу (XML)

Розмітка Інтерфейсу (XML) для додатка прогнозу погоди використовується для опису візуального представлення користувацького інтерфейсу. Використання XML дозволяє структурувати компоненти UI таким чином, що

спрощує їх модифікацію та налаштування. Це дозволяє розділити логіку додатка від його інтерфейсу, що полегшує одночасну розробку функціональності та дизайну, що в свою чергу прискорює процес створення програмного забезпечення.

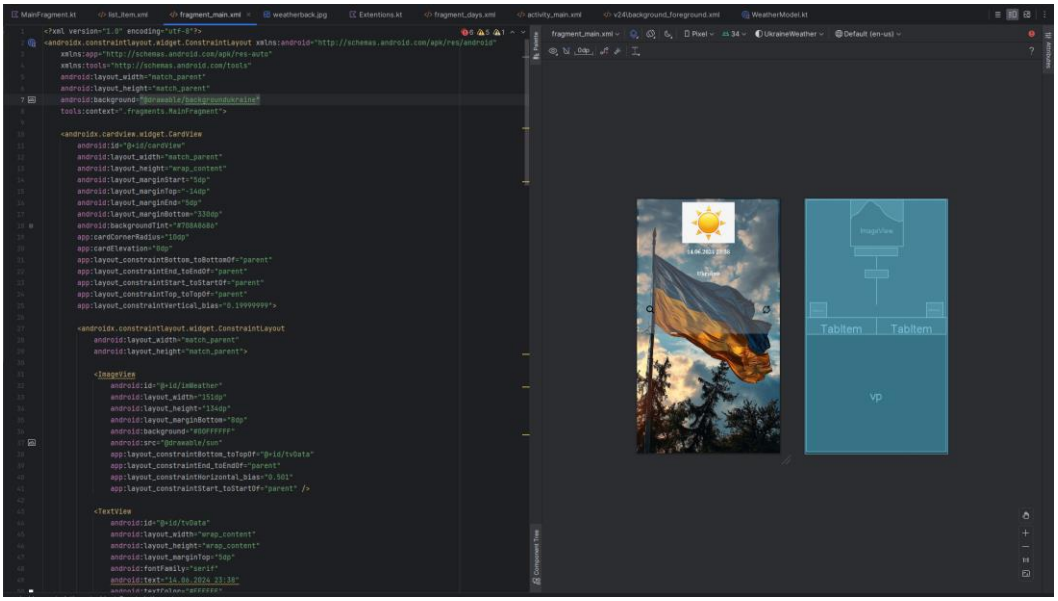


Рис 2.1.4 – XML, розмітка користувацького інтерфейсу

Переваги використання XML-розмітки для додатка прогнозу погоди:

- **Читабельність:** XML-файли легко зрозуміти і редагувати, що сприяє командній роботі та підтримці коду.
- **Розділення логіки та дизайну:** Розмітка дозволяє чітко відокремити логічну частину додатка від його візуального вигляду, що сприяє підтримці та розвитку проекту.
- **Підтримка різних пристроїв:** Завдяки XML-розмітці можна створювати адаптивний інтерфейс, який коректно відображатиметься на різних пристроях з різними розмірами екранів.

- **Масштабованість**: Можливість обробляти великі обсяги даних та складні ієрархії без втрати продуктивності.

- **Валідність та перевірка**: Використання DTD (Document Type Definition) або XML Schema для визначення структури даних дозволяє забезпечити їх валідність та цілісність.
- **Платформна незалежність**: XML є текстовим форматом, що дозволяє передавати дані між різними системами та платформами без необхідності додаткових перетворень.
- **Міжнародні стандарти**: XML є стандартом, визначеним W3C (World Wide Web Consortium), що забезпечує його стабільність та довготривалу підтримку.

Figma

Figma – це онлайн-інструмент для дизайну та прототипування, який використовується для створення веб-дизайну, мобільних додатків, інтерфейсів користувача, графічного дизайну та інших проектів.

Figma пропонує багато інструментів для створення векторних малюнків, інтерфейсів, роботи з текстом, масштабування та розміщення об'єктів, атакож інструменти для прототипування і спільної роботи[5].
Інтерфейс Figma інтуїтивно зрозумілий і легкий у використанні, щоробить його популярним серед дизайнерів будь-якого рівня.
Однією з ключових особливостей Figma є можливість спільної роботи над проектами. Кілька користувачів можуть одночасно працювати над одним проектом, бачити зміни в реальному часі та взаємодіяти один з одним.
Figma підтримує інтеграцію з багатьма іншими інструментами та сервісами,

такими як Slack, Trello, Zeplin тощо, що спрощує спільнуроботу та обмін даними між різними інструментами[6].

Figma дозволяє створювати інтерактивні прототипи, які допомагаютьвизначати поведінку і функціонал додатків або веб-сайтів перед їх розробкою.
Всі дані в Figma зберігаються в хмарі, що забезпечує доступ до них з будь-якого пристрою. Платформа також забезпечує високий рівень безпеки для збережених даних.
Figma є потужним інструментом для дизайнерів та команд, які шукають ефективні способи створення та спільної роботи над проектами в онлайн-середовищі.

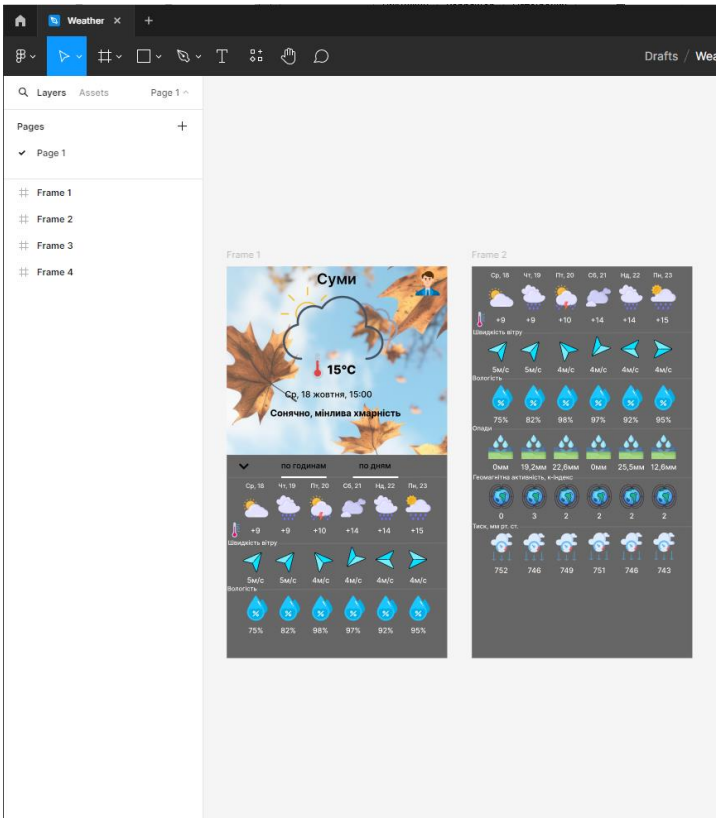


Рис 2.1.5 – Программа Figma

3. ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Розробка інтерфейсу

Розробка ефективного інтерфейсу є критичною для будь-якого мобільного додатка, зокрема для програми прогнозу погоди. Інтерфейс повинен бути не лише зрозумілим і привабливим, але й забезпечувати зручну взаємодію користувача з додатком.

Використання Android Studio для розробки інтерфейсу дозволяє максимально точно описати структуру та вигляд елементів за допомогою XML розмітки. Мінімалістичний дизайн важливий для спрощення орієнтації користувача і полегшення взаємодії з додатком. Він зменшує кількість візуальних відволікаючих елементів, зосереджуючи увагу на основних функціях програми [7](рис 3.1.1).

Рис 3.1.1 – Інтерфейс додатку прогноз погоди

Зайдній фон та колір комірок з інформацією були ретельно підібрані – патріотичний прапор України та сірий колір комірок що створює легкий огляд

фону та інфомації зображеної на екрані. За для того, щоб забезпечити зручну читабельність та зручність взаємодії, шрифт був пофарбований в білий колір. Це додає контрастності та виділяє їх на світлому фоні, навіть при недостатній яскравості дисплею.

Інтерфейс вікна завантаження програми

При відкритті додатку користувача зустрічає вікно завантаження програми(рис 3.1.2).

Рис 3.1.2 – Вікно завантаження програми

Данний екран є першою взаємодією користувача з додатком, який виконує важливу функцію для сворення першого враження від додатку. Інтерфейс завантажувального вікна розроблений у відповідності до загального стилю додатку, що передбачає світлий фон та сучасний логотип.

Світлий фон створює відчуття легкості та простору, надаючи інтерфейсу естетично приємний вигляд, що сприяє позитивному користувацькому досвіду з першої миті взаємодії. Логотип додатку, розташований в центрі вікна, привертає увагу.

Логотип, розроблений за допомогою інструменту Photoshop, є втіленням патріотизму та сучасних трендів. Він поєднує в собі герб, символ свободи та гідності, що надає логотипу привабливості, і символ сонця з хмарами, що підкреслює спрямованість додатку в погоду.

Інтерфейс головного меню

Після вікна завантаження користувач потрапить на головне меню, та головний екран застосунку. (рис. 3.3.1)

Рис 3.3.1 – Інтерфейс головного меню

На основному екрані ми можемо побачити такі елементи: погоду на даний момент на вулиці, дату за якою переглядається прогноз. Місто в якому знаходиться користувач, або місто яке було вибрано в пошуку, максимальну та мінімальну температуру в цю добу, та хмарність неба. Також нижче можемо побачити кнопку пошуку по місті (створена для того, щоб користувач, який не хоче щоб бралася інформація про його місцезнаходження не відображалася),

та кнопку перезавантаження, і кнопки вибору показу погоди по годинам та дням. Кнопки представлені у вигляді векторного зображення, що дозволяє не втрачати якість ілюстрації самих кнопок.

Нижче ми можемо побачити велике меню з прогнозом (датами, часом направленням вітру, температурою, вологістю повітря та картинку з станом неба).

3.2 Опис програмної реалізації

Основна структура додатка заснована на архітектурі з використанням Fragment для розділення функціональності між різними компонентами інтерфейсу. В головній активності, MainFragment, розміщується ViewPager2 з вкладками для переміщення між фрагментами, які представляють різні секції додатка, такі як поточний прогноз, погодні карти та налаштування.

Користувач відкриває додаток, який починається з екрану-заставки (SplashFragment). Через кілька секунд додаток автоматично переходить до головної активності (MainFragment).

У MainFragment додаток перевіряє наявність необхідних дозволів, таких як доступ до місцезнаходження для надання точних прогнозів. Якщо дозволу немає, додаток запитує його у користувача. Після чого додаток починає відслідковувати місцезнаходження користувача і робить поновлення прогнозу погоди.

- **Виконати функцію пошуку по місту –** натиснути на поле пошуку та вести назву міста на англійській мові.
- **Поновити інформацію –** натиснути на кнопку та інформація поновиться .
- **Змінити відображення погоди по дням –** натиснути на кнопку з назвою «Days»
- **Змінити відображення погоди по годинам –** натиснути на кнопку з назвою «Hours»

Розглянемо основні компоненти додатку, їх функції та опис:

1.Основний клас MainFragment

MainFragment є центральним фрагментом додатку, який відповідає за відображення основного інтерфейсу користувача, включаючи поточний прогноз погоди та різні вкладки для перегляду прогнозів на годинному та денному рівнях.

Функції:

Ініціалізація основних компонентів, таких як вкладки і адаптери.

Перевірка та запит дозволів на доступ до місцезнаходження.

Запит та обробка даних про погоду з API[9].

Оновлення інтерфейсу відповідно до отриманих даних.

2. Перевірка дозволів та доступ до місцезнаходження

Додаток перевіряє наявність необхідних дозволів для доступу до місцезнаходження користувача. Якщо дозволи не надані, він запитує їх у користувача.

Функції:

checkPermission(): перевіряє наявність дозволу на доступ до місцезнаходження.

permissionListener(): обробляє результат запиту дозволів.

isPermissionGranted(permission: String): перевіряє, чи надано певний дозвіл.

3. Отримання даних про місцезнаходження

Додаток використовує сервіси локації для отримання поточного місцезнаходження користувача.

Функції:

getLocation(): отримує поточне місцезнаходження користувача та запитує погодні дані для цього місця.

isLocationEnabled(): перевіряє, чи ввімкнено сервіси локації на пристрої.

4. Запит та обробка даних про погоду

Додаток використовує Volley для виконання HTTP-запитів до API погоди та обробки отриманих даних.

Функції:

requestWeatherData(city: String): відправляє запит на сервер для отримання прогнозу погоди.

parseWeatherData(result: String): обробляє отриманий JSON з даними про погоду.

parseDays(mainObject: JSONObject): обробляє прогноз на декілька днів.

parseCurrentData(mainObject: JSONObject, weatherItem: WeatherModel): обробляє поточні погодні дані.

5. Оновлення інтерфейсу

Після отримання даних про погоду додаток оновлює інтерфейс для відображення поточного прогнозу.

Функції:

updateCurrentCard(): оновлює основну картку з поточним прогнозом.

init(): ініціалізує вкладки та інші елементи інтерфейсу.

Опис інтерфейсу

Інтерфейс реалізовано в мінімалістичному стилі, що дозволяє користувачам легко орієнтуватися та взаємодіяти з елементами. Використання фрагментів і вкладок дозволяє розділити функціональність між різними екранами, забезпечуючи зручний доступ до прогнозів на годинному та денному рівнях.

1. Клас WeatherAdapter

WeatherAdapter відповідає за відображення списку погодних даних у вигляді карток у RecyclerView. Це дозволяє ефективно відображати велику кількість погодних даних з можливістю прокрутки. Адаптери забезпечують ефективне відображення та оновлення даних у користувацькому інтерфейсі. Основним класом у цьому пакеті є WeatherAdapter, який використовує патерн ViewHolder для оптимізації роботи з RecyclerView.

Функції:

onCreateViewHolder(parent: ViewGroup, viewType: Int): Holder: створює новий ViewHolder, коли RecyclerView потребує нового елемента.

onBindViewHolder(holder: Holder, position: Int): прив'язує дані до ViewHolder на вказаній позиції.

Comparator: внутрішній клас, який використовується для порівняння елементів списку для оптимізації оновлень.

2. Клас Holder

Holder є внутрішнім класом WeatherAdapter, який відповідає за відображення окремого елемента списку.

Функції:

bind(item: WeatherModel): прив'язує дані з об'єкта WeatherModel до відповідних елементів інтерфейсу.

Ініціалізація обробника кліків, який передає об'єкт WeatherModel слухачу (listener).

3. Інтерфейс Listener

Listener дозволяє обробляти події кліків на окремих елементах списку.

Функції:

onClick(item: WeatherModel): метод, який викликається при натисканні на елемент списку.

4. Модель даних WeatherModel

WeatherModel є модель даних, яка містить всю необхідну інформацію про погодні умови для відображення в інтерфейсі.

Властивості:

time: час або дата прогнозу.

condition: погодні умови (наприклад, ясно, хмарно).

currentTemp: поточна температура.

maxTemp: максимальна температура.

minTemp: мінімальна температура.

imageUrl: URL зображення, яке відображає погодні умови.

3.3 Тестування

Для тестування додатку я використав метод мануального тестування, який дозволяє детально перевірити роботу всіх функцій і елементів інтерфейсу користувача. Нижче наведено опис процесу мануального тестування, який я виконав на своєму смартфоні.

Пристрій: Pixel_3A_API_34(Android 12).

Підготовка

- **Встановлення додатку:** Завантажив та встановив APK-файл додатку на свій смартфон.
- **Налаштування середовища:** Переконався, що на пристрої встановлено необхідні версії Android та наявний стабільний доступ до Інтернету.

Тестування основних функцій:

1. **Запит на доступ до геопозиції**

Мета: Перевірити функціонал запитів на доступ геолокації

Процедура: Зайти в додаток, натиснути дозволити доступ до геолокації, якщо такий запит наявний.

Результат: Відбулося відображення запиту додатку щодо дозволу доступу до медіафайлів. Всі медіафайли зчитались та були відображені у вигляді списку(рис 3.3.1).

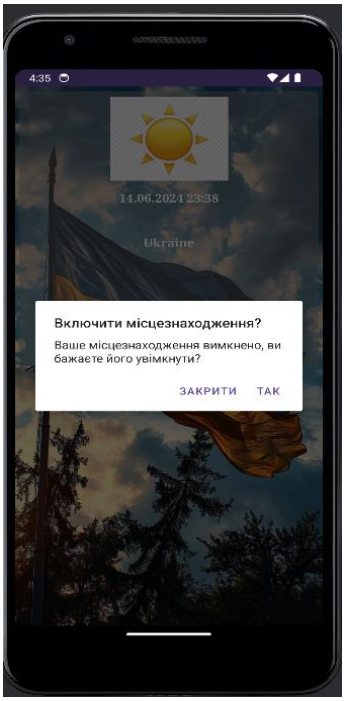


Рис 3.3.1 – Результати тестування №1

ВИСНОВОК

Створення патріотичного прогнозу погоди було комплексним проектом, який включав кілька етапів розробки та використання різноманітних технологій.

Цей додаток призначений для надання користувачам актуальної інформації про погодні умови з патріотичним дизайном, що підкреслює національні кольори та символіку.

Для реалізації проекту було використано наступні інструменти та технології: Figma, Kotlin, Android Studio та API. На початковому етапі ми використовували Figma для створення прототипів інтерфейсу користувача.

Це дозволило нам швидко візуалізувати наші ідеї, обговорити їх з командою та внести необхідні корективи до дизайну. Figma забезпечила ефективну співпрацю дизайнерів та розробників, дозволяючи створити привабливий та функціональний інтерфейс. Основною мовою програмування для розробки додатку було обрано Kotlin.

Це сучасна мова, яка має всі переваги Java, але є більш компактною та безпечною. Використання Kotlin дозволило нам скоротити кількість коду та зменшити кількість можливих помилок.

Для розробки та налагодження додатку ми використовували Android Studio – офіційне інтегроване середовище розробки для Android. Android Studio забезпечила всі необхідні інструменти для створення високоякісного додатку, включаючи редактор коду, вбудовані емулатори та інструменти для налагодження. Для отримання актуальних даних про погоду ми інтегрувалися з погодним API.

Це дозволило нашому додатку отримувати та відображати точну інформацію про поточні погодні умови, прогноз на кілька днів вперед та інші метеорологічні

дані. Використання API забезпечило динамічність додатку та його актуальність. Процес розробки був спланований таким чином, щоб максимально ефективно використовувати наявні ресурси та забезпечити високу якість кінцевого продукту. Усі етапи проекту були ретельно сплановані та виконані, починаючи від створення дизайну у Figma, написання коду на Kotlin в Android Studio та інтеграції з погодним API.

Результатом нашої роботи став функціональний та естетично привабливий додаток патріотичного прогнозу погоди, який не лише забезпечує користувачів необхідною інформацією, але й додає нотку національної гордості.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Програмування з нуля на Kotlin українською (146) Програмування з нуля на Kotlin українською - YouTube
2. Основи мови програмування на Kotlin Основи Kotlin | krypton.com.ua
3. Основи Android Studio Основи Android Studio. Уроки для початківців. W3Schools українською (w3schoolsua.github.io)
4. Android з 0 до junior ANDROID УКРАЇНСЬКОЮ. Старт курсу, урок перший, Android Studio (youtube.com)
5. Figma з нуля. Вчимось працювати у фігма Figma з нуля. Вчимось працювати у фігма - UXPUB UA Дизайн-спільнота
6. Figma українською | Що таке Фігма і як вона працює? Figma українською | Що таке Фігма і як вона працює? (youtube.com)
7. XML для початківців Підручник XML для початківців (guru99.com)
8. Що таке API | І де їх шукати| Що таке API | І де їх шукати
9. WeatherApi|WeatherApi
10. Додатки для прогнозу погоди | Додатки для прогнозу погоди

Додаток – Код модулів додатку

Main Fragment

```
package com.example.ukraineweather.fragments                    ⚠9 ⚠1 ✓5 ^ v

import android.Manifest
import android.app.Instrumentation.ActivityResult
import android.content.Context
import android.content.Intent
import android.content.pm.PackageManager
import android.location.LocationManager
import android.os.Bundle
import android.provider.Settings
import android.util.Log
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.PixelCopy.Request
import android.view.View
import android.view.ViewGroup
import android.widget.TableLayout
import android.widget.Toast
import androidx.activity.result.ActivityResultLauncher
import androidx.activity.result.contract.ActivityResultContracts
import androidx.core.app.ActivityCompat
import androidx.fragment.app.FragmentActivity
import androidx.fragment.app.activityViewModels
import androidx.privacysandbox.tools.core.model.Method
import com.android.volley.toolbox.RequestFuture
import com.android.volley.toolbox.StringRequest
import com.android.volley.toolbox.Volley
import com.example.ukraineweather.DialogManager
import com.example.ukraineweather.MainViewModel
import com.example.ukraineweather.R
import com.example.ukraineweather.adapters.VpAdapter
import com.example.ukraineweather.adapters.WeatherModel
import com.example.ukraineweather.databinding.ActivityMainBinding
import com.example.ukraineweather.databinding.FragmentMainBinding
import com.google.android.gms.location.FusedLocationProviderClient
import com.google.android.gms.location.LocationRequest
import com.google.android.gms.location.LocationServices
import com.google.android.gms.location.Priority
import com.google.android.gms.tasks.CancellationTokenSource
import com.google.android.material.tabs.TabLayoutMediator
```

MainFragment.kt

```kotlin
import com.squareup.picasso.Picasso
import org.json.JSONObject

const val API_KEY = "caad7e36fee3471caad05018240506"

class MainFragment : Fragment() {
    private lateinit var fLocationClient: FusedLocationProviderClient
    private val fList = listOf(
        HoursFragment.newInstance(),
        DaysFragment.newInstance()
    )
    private val tList = listOf(
        "Hours",
        "Days"
    )
    private lateinit var pLauncher: ActivityResultLauncher<String>
    private lateinit var binding: FragmentMainBinding
    private val model: MainViewModel by activityViewModels()

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

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

    override fun onResume() {
        super.onResume()
        checkLocation()
    }

    private fun init() = with(binding){ this: FragmentMainBinding
```

MainFragment.kt

```
        fLocationClient = LocationServices.getFusedLocationPr(  9  1  5
        val adapter = VpAdapter(activity as FragmentActivity, fList)
        vp.adapter = adapter
        TabLayoutMediator(tabLayout, vp){
                tab, pos -> tab.text = tList[pos]
        }.attach()
        ibRefresh.setOnClickListener { it: View!
            tabLayout.selectTab(tabLayout.getTabAt( index: 0))
            checkLocation()
        }
        ibSearch.setOnClickListener { it: View!
            DialogManager.searchByNameDialog(requireContext(), object : DialogM
                override fun onClick(name: String?) {
                    name?.let { it1 -> requestWeatherData(it1) }
                }
            })
        }
    }

    private fun checkLocation(){
        if(isLocationEnabled()){
            getLocation()
        } else {
            DialogManager.locationSettingsDialog(requireContext(), object : Dia
                override fun onClick(name: String?) {
                    startActivity(Intent(Settings.ACTION_LOCATION_SOURCE_SETTIN
                }
            })
        }
    }

    private fun isLocationEnabled(): Boolean{
        val lm = activity?.getSystemService(Context.LOCATION_SERVICE) as Locati
        return lm.isProviderEnabled(LocationManager.GPS_PROVIDER)
    }

    private fun getLocation(){
        val ct = CancellationTokenSource()
        if (ActivityCompat.checkSelfPermission(
                requireContext(),
```


MainFragment.kt
9 1 5
private fun getLocation(){
val ct = CancellationTokenSource()
if (ActivityCompat.checkSelfPermission(
requireContext(),
Manifest.permission.ACCESS_FINE_LOCATION
) != PackageManager.PERMISSION_GRANTED && ActivityCompat.checkSelfP
requireContext(),
Manifest.permission.ACCESS_COARSE_LOCATION
) != PackageManager.PERMISSION_GRANTED
) {
return
}
fLocationClient
.getCurrentLocation(Priority.PRIORITY_HIGH_ACCURACY, ct.token)
.addOnCompleteListener{ it: Task<Location!>
requestWeatherData(city: "${it.result.latitude},${it.result.long
}
}
private fun updateCurrentCard() = with(binding){ this: FragmentMainBinding
model.liveDataCurrent.observe(viewLifecycleOwner){ it: WeatherModel!
val maxMinTemp = "${it.maxTemp}°C / ${it.minTemp}°C"
tvData.text = it.time
tvCity.text = it.city
tvCurrentTemp.text = it.currentTemp.ifEmpty { maxMinTemp }
tvCondition.text = it.condition
tvMaxMin.text = if(it.currentTemp.isEmpty()) "" else maxMinTemp
Picasso.get().load(path: "https:" + it.imageUrl).into(imWeather)
}
}
private fun permissionListener(){
pLauncher = registerForActivityResult(
ActivityResultContracts.RequestPermission()){ it: Boolean!
Toast.makeText(activity, text: "Permission is $it", Toast.LENGTH_LON
}
}
private fun checkPermission(){
MainFragment.kt > MainFragment > checkLocation
109:10 LF UTF-8 4 spaces

MainFragment.kt

```
155     private fun checkPermission(){                                  ⚠9 ⚠1 ✓5
156         if(!isPermissionGranted(Manifest.permission.ACCESS_FINE_LOCATION)){
157             permissionListener()
158             pLauncher.launch(Manifest.permission.ACCESS_FINE_LOCATION)
159         }
160     }
161
162     private fun isPermissionGranted(permission: String): Boolean {
163         return ActivityCompat.checkSelfPermission(
164             requireContext(),
165             permission
166         ) == PackageManager.PERMISSION_GRANTED
167     }
168
169
170     private fun requestWeatherData(city: String){
171         val url = "https://api.weatherapi.com/v1/forecast.json?key=" +
172                 API_KEY +
173                 "&q=" +
174                 city +
175                 "&days=" +
176                 "10" +
177                 "&aqi=no&alerts=no"
178         val queue = Volley.newRequestQueue(context)
179         val request = StringRequest(
180             com.android.volley.Request.Method.GET,
181             url,
182             {
183                     result -> parseWeatherData(result)
184             },
185             {
186                     error -> Log.d( tag: "MyLog",  msg: "Error: $error")
187             }
188         )
189         queue.add(request)
190     }
191
192     private fun parseWeatherData(result: String) {
193         val mainObject = JSONObject(result)
194         val list = parseDays(mainObject)
```

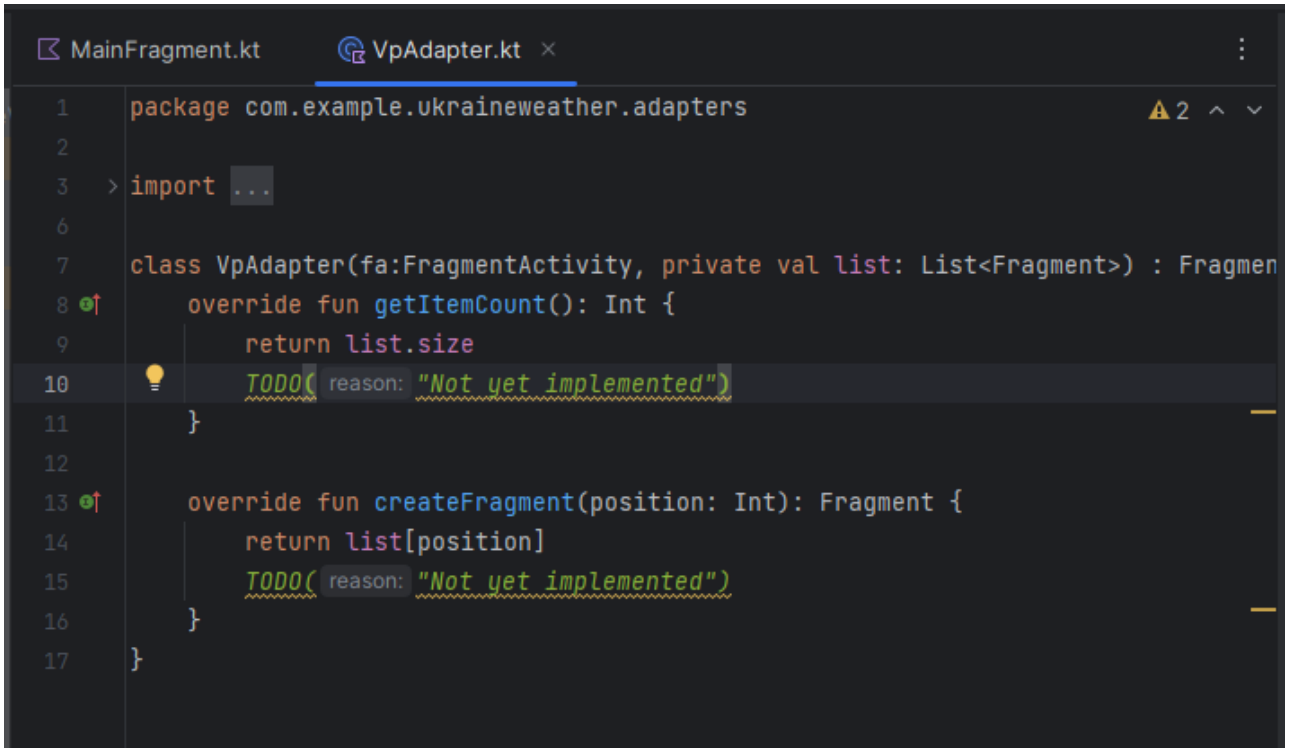
s › MainFragment.kt › MainFragment › checkLocation 109:10 LF UTF-8 4 spaces


MainFragment.kt
9 1 5
private fun parseWeatherData(result: String) {
val mainObject = JSONObject(result)
val list = parseDays(mainObject)
parseCurrentData(mainObject, list[0])
}
private fun parseDays(mainObject: JSONObject): List<WeatherModel>{
val list = ArrayList<WeatherModel>()
val daysArray = mainObject.getJSONObject(name: "forecast")
.getJSONArray(name: "forecastday")
val name = mainObject.getJSONObject(name: "location").getString(name:
for (i in 0 ≤ until < daysArray.length()){
val day = daysArray[i] as JSONObject
val item = WeatherModel(
name,
day.getString(name: "date"),
day.getJSONObject(name: "day").getJSONObject(name: "condition")
.getString(name: "text"), currentTemp: "",
day.getJSONObject(name: "day").getString(name: "maxtemp_c").toF
day.getJSONObject(name: "day").getString(name: "mintemp_c").toF
day.getJSONObject(name: "day").getJSONObject(name: "condition")
.getString(name: "icon"),
day.getJSONArray(name: "hour").toString()
)
list.add(item)
}
model.liveDataList.value = list
return list
}
private fun parseCurrentData(mainObject: JSONObject, weatherItem: WeatherMo
val item = WeatherModel(
mainObject.getJSONObject(name: "location").getString(name: "name"),
mainObject.getJSONObject(name: "current").getString(name: "last_upd
mainObject.getJSONObject(name: "current")
.getJSONObject(name: "condition").getString(name: "text"),
currentTemp: "${mainObject.getJSONObject(name: "current").getString(
weatherItem.maxTemp,
weatherItem.minTemp,
mainObject.getJSONObject(name: "current")
MainFragment.kt > MainFragment > checkLocation
109:10 LF UTF-8 4 spaces

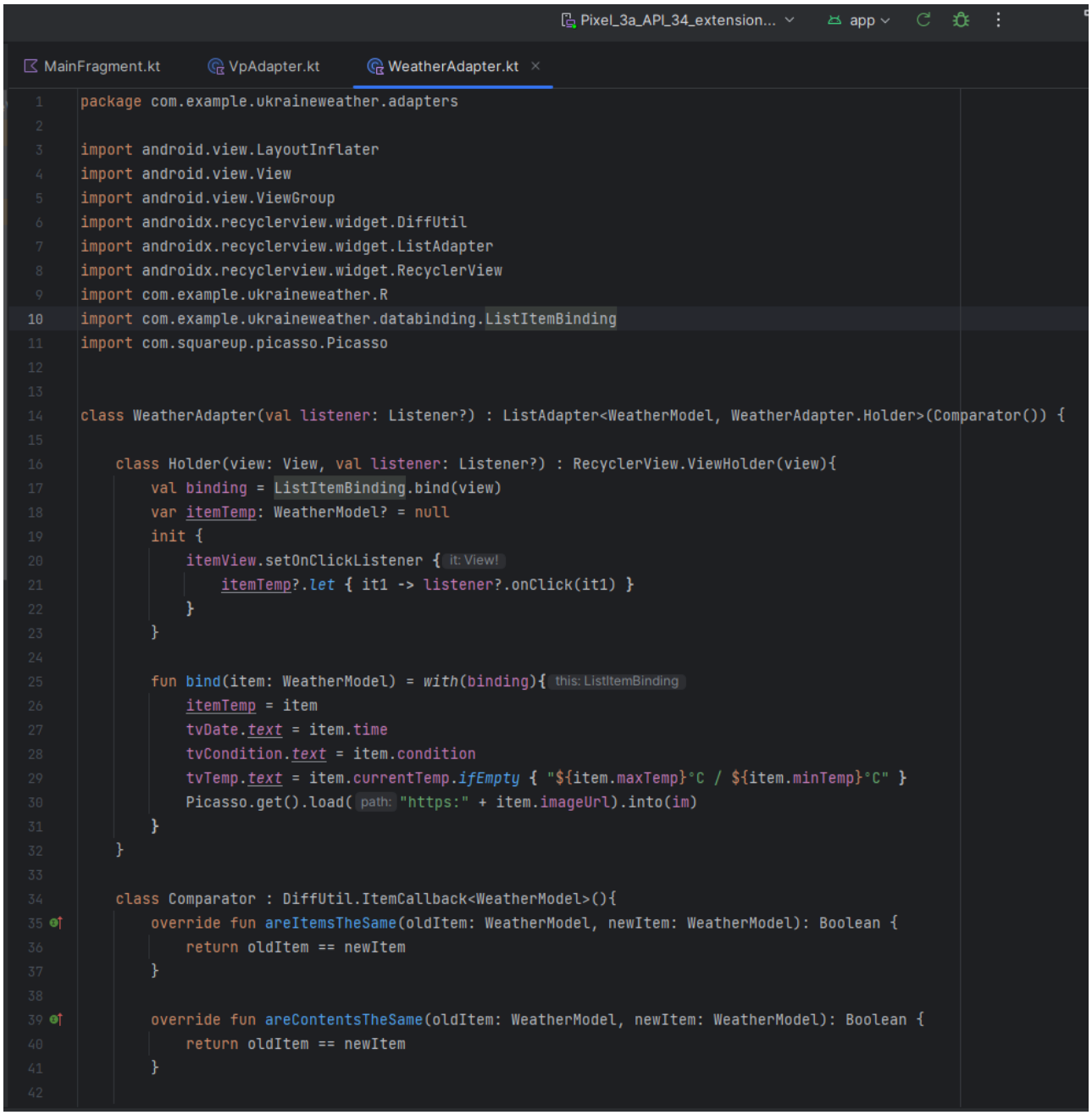
MainFragment.kt

```
                day.getJSONObject( name: "day").getJSONObject(
                    .getString( name: "text"),
                day.getJSONObject( name: "day").getString( name: "maxtemp_c").to
                day.getJSONObject( name: "day").getString( name: "mintemp_c").to
                day.getJSONObject( name: "day").getJSONObject( name: "condition"
                    .getString( name: "icon"),
                day.getJSONArray( name: "hour").toString()
            )
            list.add(item)
        }
        model.liveDataList.value = list
        return list
    }

    private fun parseCurrentData(mainObject: JSONObject, weatherItem: WeatherM
        val item = WeatherModel(
            mainObject.getJSONObject( name: "location").getString( name: "name")
            mainObject.getJSONObject( name: "current").getString( name: "last_up
            mainObject.getJSONObject( name: "current")
                .getJSONObject( name: "condition").getString( name: "text"),
            currentTemp: "${mainObject.getJSONObject( name: "current").getString(
            weatherItem.maxTemp,
            weatherItem.minTemp,
            mainObject.getJSONObject( name: "current")
                .getJSONObject( name: "condition").getString( name: "icon"),
            weatherItem.hours
        )
        model.liveDataCurrent.value = item
    }

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

VpAdapter


MainFragment.kt
VpAdapter.kt
package com.example.ukraineweather.adapters
2
import ...
class VpAdapter(fa:FragmentActivity, private val list: List<Fragment>) : Fragmen
override fun getItemCount(): Int {
return list.size
TODO(reason: "Not yet implemented")
}
override fun createFragment(position: Int): Fragment {
return list[position]
TODO(reason: "Not yet implemented")
}
}

WeatherAdapter



```
42
43      }
44
45      override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): Holder {
46          val view = LayoutInflater.from(parent.context).inflate(R.layout.list_item, parent,  attachToRoot: false)
47          return Holder(view, listener)
48      }
49
50      override fun onBindViewHolder(holder: Holder, position: Int) {
51          holder.bind(getItem(position))
52      }
53
54      interface Listener{
55          fun onClick(item: WeatherModel)
56      }
57  }
```

WeatherModel

MainFragment.kt | VpAdapter.kt | WeatherAdapter.kt | WeatherModel.kt

```
1   package com.example.ukraineweather.adapters
2
3   data class WeatherModel(
4       val city: String,
5       val time: String,
6       val condition: String,
7       val currentTemp: String,
8       val maxTemp: String,
9       val minTemp: String,
10      val imageUrl: String,
11      val hours: String,
12  )
```

DayItem

MainFragment.kt | VpAdapter.kt | WeatherAdapter.kt | WeatherModel.kt | DayItem.kt

```kotlin
package com.example.ukraineweather.fragments

import java.util.concurrent.locks.Condition

data class DayItem(
    val city: String,
    val time: String,
    val condition: String,
    val imageUrl: String,
    val currentTemp: String,
    val maxTemp: String,
    val minTemp: String,
    val hours: String,
)
```

DaysFragment

MainFragment.kt | VpAdapter.kt | WeatherAdapter.kt | WeatherModel.kt | DayItem.kt | DaysFragment.kt

```kotlin
package com.example.ukraineweather.fragments

import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.fragment.app.activityViewModels
import androidx.recyclerview.widget.LinearLayoutManager
import com.example.ukraineweather.MainViewModel
import com.example.ukraineweather.R
import com.example.ukraineweather.adapters.WeatherAdapter
import com.example.ukraineweather.adapters.WeatherModel
import com.example.ukraineweather.databinding.FragmentDaysBinding

class DaysFragment : Fragment(), WeatherAdapter.Listener {
    private lateinit var adapter: WeatherAdapter
    private lateinit var binding: FragmentDaysBinding
    private val model: MainViewModel by activityViewModels()

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

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        init()
        model.liveDataList.observe(viewLifecycleOwner){ it: List<WeatherModel>!
            adapter.submitList(it.subList(1, it.size))
        }
    }

    private fun init() = with(binding){ this: FragmentDaysBinding
        adapter = WeatherAdapter( listener: this@DaysFragment)
        rcView.layoutManager = LinearLayoutManager(activity)
        rcView.adapter = adapter
    }

```

```
20
21      override fun onCreateView(
22          inflater: LayoutInflater, container: ViewGroup?,
23          savedInstanceState: Bundle?
24      ): View {
25          binding = FragmentDaysBinding.inflate(inflater, container,  attachToParent: false)
26          return binding.root
27      }
28
29      override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
30          super.onViewCreated(view, savedInstanceState)
31          init()
32          model.liveDataList.observe(viewLifecycleOwner){ it: List<WeatherModel>!
33              adapter.submitList(it.subList(1, it.size))
34          }
35      }
36
37      private fun init() = with(binding){ this: FragmentDaysBinding
38          adapter = WeatherAdapter( listener: this@DaysFragment)
39          rcView.layoutManager = LinearLayoutManager(activity)
40          rcView.adapter = adapter
41      }
42
43      companion object {
44          @JvmStatic
45          fun newInstance() = DaysFragment()
46      }
47
48      override fun onClick(item: WeatherModel) {
49          model.liveDataCurrent.value = item
50      }
51  }
```

Extentions

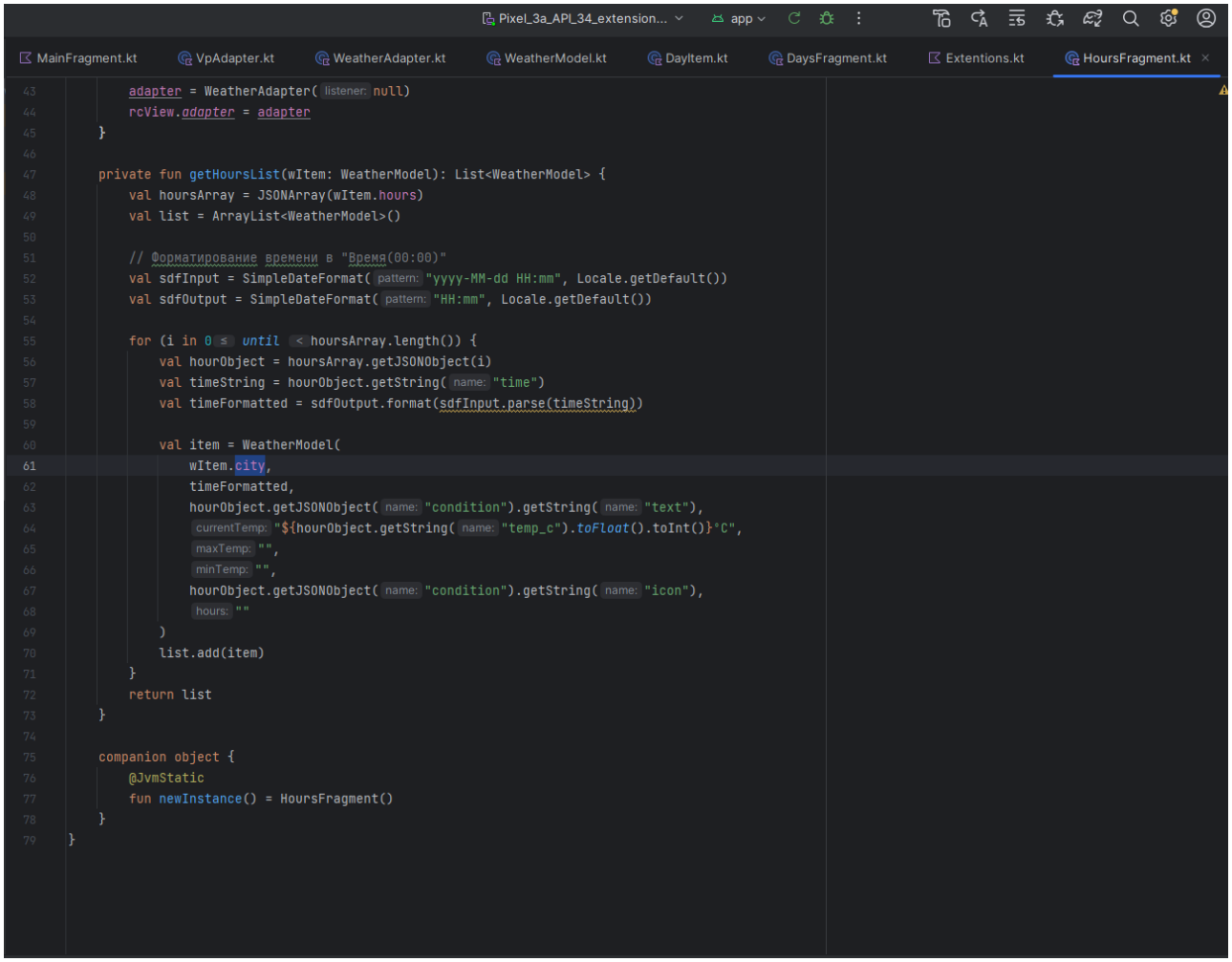
MainFragment.kt | VpAdapter.kt | WeatherAdapter.kt | WeatherModel.kt | DayItem.kt | DaysFragment.kt | Extentions.kt

```kotlin
package com.example.ukraineweather.fragments

import android.content.pm.PackageManager
import androidx.appcompat.app.AppCompatActivity
import androidx.core.content.ContextCompat
import androidx.fragment.app.Fragment

fun Fragment.isPermissionGranted(p: String): Boolean {
    return ContextCompat.checkSelfPermission(
        activity as AppCompatActivity, p) == PackageManager.PERMISSION_GRANTED
}
```

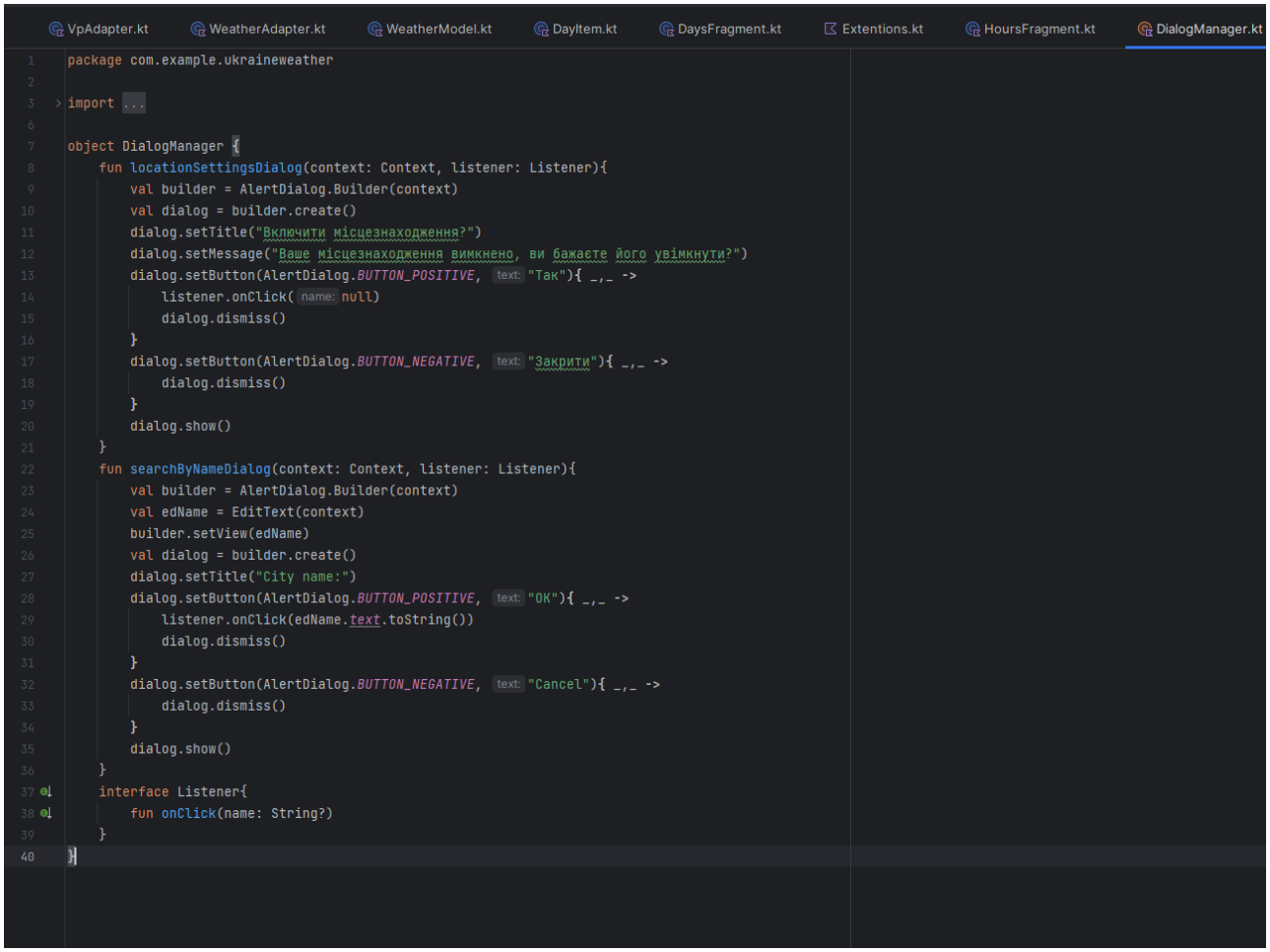
HoursFragment

MainFragment.kt | VpAdapter.kt | WeatherAdapter.kt | WeatherModel.kt | DayItem.kt | DaysFragment.kt | Extentions.kt | HoursFragment.kt

```kotlin
package com.example.ukraineweather.fragments

import ...

class HoursFragment : Fragment() {
    private lateinit var binding: FragmentHoursBinding
    private lateinit var adapter: WeatherAdapter
    private val model: MainViewModel by activityViewModels()

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

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        initRcView()
        model.liveDataCurrent.observe(viewLifecycleOwner) { it: WeatherModel!
            adapter.submitList(getHoursList(it))
        }
    }

    private fun initRcView() = with(binding) { this: FragmentHoursBinding
        rcView.layoutManager = LinearLayoutManager(activity)
        adapter = WeatherAdapter( listener: null)
        rcView.adapter = adapter
    }

    private fun getHoursList(wItem: WeatherModel): List<WeatherModel> {
        val hoursArray = JSONArray(wItem.hours)
        val list = ArrayList<WeatherModel>()

        // Форматирование времени в "Время(00:00)"
        val sdfInput = SimpleDateFormat( pattern: "yyyy-MM-dd HH:mm", Locale.getDefault())
        val sdfOutput = SimpleDateFormat( pattern: "HH:mm", Locale.getDefault())

        for (i in 0 ≤ until < hoursArray.length()) {
            val hourObject = hoursArray.getJSONObject(i)
            val timeString = hourObject.getString( name: "time")
```



Dialog Manager

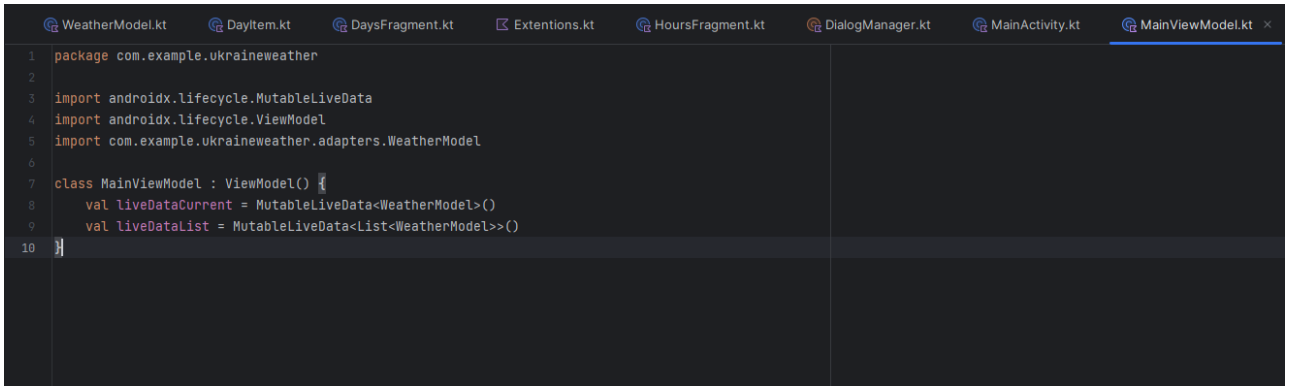


```
package com.example.ukraineweather

import ...

object DialogManager {
    fun locationSettingsDialog(context: Context, listener: Listener){
        val builder = AlertDialog.Builder(context)
        val dialog = builder.create()
        dialog.setTitle("Включити місцезнаходження?")
        dialog.setMessage("Ваше місцезнаходження вимкнено, ви бажаєте його увімкнути?")
        dialog.setButton(AlertDialog.BUTTON_POSITIVE, text: "Так"){ _,_ ->
            listener.onClick( name: null)
            dialog.dismiss()
        }
        dialog.setButton(AlertDialog.BUTTON_NEGATIVE, text: "Закрити"){ _,_ ->
            dialog.dismiss()
        }
        dialog.show()
    }
    fun searchByNameDialog(context: Context, listener: Listener){
        val builder = AlertDialog.Builder(context)
        val edName = EditText(context)
        builder.setView(edName)
        val dialog = builder.create()
        dialog.setTitle("City name:")
        dialog.setButton(AlertDialog.BUTTON_POSITIVE, text: "OK"){ _,_ ->
            listener.onClick(edName.text.toString())
            dialog.dismiss()
        }
        dialog.setButton(AlertDialog.BUTTON_NEGATIVE, text: "Cancel"){ _,_ ->
            dialog.dismiss()
        }
        dialog.show()
    }
    interface Listener{
        fun onClick(name: String?)
    }
}
```

MainActivity

```
package com.example.ukraineweather

import ...

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()
    }
}
```

MainViewModel



```
package com.example.ukraineweather

import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel
import com.example.ukraineweather.adapters.WeatherModel

class MainViewModel : ViewModel() {
    val liveDataCurrent = MutableLiveData<WeatherModel>()
    val liveDataList = MutableLiveData<List<WeatherModel>>()
}
```