МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Кафедра Комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
Аліна САВЧЕНКО.
«_____» _____ 2024р.

КВАЛІФІКАЦІЙНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ЗДОБУВАЧА ОСВІТНЬОГО СТУПЕНЯ **"БАКАЛАВР"**

Тема: " Мобільний застосунок для автоматичного надсилання сповіщень про землетруси з використанням мови програмування Kotlin"

Виконавець: студент групи УС-411 Зінченко Дмитро Вікторович

Керівник: к.т.н., доцент Колісник Олена Василівна

Нормоконтролер: Олександр ШЕВЧЕНКО

Київ – 2024

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних наук та технологій*

Кафедра *Комп'ютерних інформаційних технологій*

Галузь знань, спеціальність, освітньо-професійна програма: 12 "Інформаційні технології", 122 "Комп'ютерні науки", "Інформаційні управляючі системи та технології"

.

ЗАТВЕРДЖУЮ

Завідувач випускової кафедри

_____ Аліна САВЧЕНКО

«____»_____2024р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи студентки

Зінченка Дмитра Вікторовича

(прізвище, ім'я, по батькові)

1. Тема роботи: «Мобільний застосунок для автоматичного надсилання сповіщень про землетруси з використанням мови програмування Kotlin» затверджена наказом ректора від 05.04.2024р. №517/ст.

2. Термін виконання роботи: з 06.05.2024 р. по 16.06.2024 р.

3. Вихідні данні до роботи: Android-додаток для автоматизації та оптимізації процесу надсилання сповіщень про надзвичайні ситуації в світі, а саме – землетруси.

4. Зміст пояснювальної записки: вступ, загальні поняття Android-розробки, основні поняття та види мов програмування, середовище розробки, основні етапи розробки мобільного додатку, аналіз аналогічних додатків у Google Play, аналіз платформи push-сповіщень для взаємодії із застосунком, використання API в реальних умовах, уточнення вимог до системи та дизайн додатку, основний алгоритм написання програми, архітектура програмного забезпечення, опис функціональності системи.

5. Перелік обов'язкового графічного матеріалу: слайди презентації Microsoft Power Point, рисунки та макет застосунку.

6. Календарний план-графік

№ п/п	Завдання	Термін виконання	Підпис керівника
1.	Отримання завдання на дипломну роботу, створення плану дипломної роботи та побудова плану-графіку виконання робіт.	06.05.2024	
2.	Розроблення та затвердження календарного плану виконання дипломної роботи.	07.05.2024	
3.	Проведення консультацій з науковим керівником.	08.05.2024-09.05.2024	
4.	Розробка застосунку для автоматичного надсилання сповіщень про землетруси	09.05.2024-19.05.2024	
5.	Написання Вступу та Розділу 1 дипломної роботи	19.05.2024-22.05.2024	
6.	Написання Розділу 2 дипломної роботи	22.05.2024-26.05.2024	
7.	Написання Розділу 3 та Висновків дипломної роботи	26.05.2024-02.06.2024	
8.	Кінцеве оформлення дипломної роботи згідно з вимогами	02.06.2024-03.06.2024	
9.	Оформлення та друк пояснювальної записки.	04.06.2024	
10.	Створення презентації, доповіді та підготовка до захисту дипломної роботи.	05.06.2024-07.06.2024	
11.	Підготовка матеріалів дипломної роботи для передачі секретарю ДЕК (папка, конверт, рецензія, відгук).	08.06.2024	

7. Дата видачі завдання: «6» травня 2024 р.

Керівник дипломної роботи Олена КОЛІСНИК
(П.І.Б.)

Завдання прийняв до виконання Дмитро ЗІНЧЕНКО
(П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Мобільний застосунок для автоматичного надсилання сповіщень про землетруси з використанням мови програмування Kotlin»: 107 с., 12 рис., 1 табл., 19 літературних джерел.

Об'єктом дослідження: мобільний застосунок для автоматичного надсилання сповіщень про землетруси.

Предмет дослідження: розробка мобільного застосунку для автоматичного надсилання сповіщень про землетруси з використанням мови програмування Kotlin.

Мета роботи: аналіз проблематики сповіщень про землетруси, розробку програмного забезпечення для автоматичного виявлення землетрусів та надсилання сповіщень користувачам за допомогою мобільного додатку.

Методи дослідження: аналіз аналогів, методи створення бази даних, методи проектування користувацького інтерфейсу, методи створення фонових процесів, методи реалізації Clean Architecture.

Отримані результати та їх новизна: розроблено Android-додаток для автоматичного надсилання сповіщень про землетруси, алгоритм написання якого можна використовувати при розробці подібних програм. Новизна полягає в тому, що даний застосунок використовує дані з відкритих джерел про землетруси та надсилає користувачам миттєві сповіщення з метою надання безпеки та інформування про небезпечні ситуації в реальному часі.

Результати кваліфікаційної роботи рекомендується використовувати під час проведення наукових досліджень та в практичній діяльності для створення подібних застосунків з використанням наданих алгоритмів та методів розробки.

ANDROID-ЗАСТОСУНОК, МОВА ПРОГРАМУВАННЯ, СЕРЕДОВИЩЕ РОЗРОБКИ, АРХІТЕКТУРА, PUSH-СПОВІЩЕННЯ, COROUTINEWORKER, ONE SIGNAL, ANDROID STUDIO, WORK MANAGER, HTTP-ЗАПИТ, ПРОГРАМНИЙ ІНТЕРФЕЙС.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

API	Application Programming Interface – це програмний інтерфейс, що дозволяє пов'язувати між собою різні програми. Створений для спрощення та прискорення розробки. Містить набори методів, класів, бібліотек та функцій
SSL	Відкритий програмний продукт, розроблений як універсальна бібліотека для криптографії, що використовує протоколи Secure Sockets Layer і Transport Layer Security
SQLite	Полегшена реляційна система керування базами даних. Втілена у вигляді бібліотеки, де реалізовано багато зі стандарту SQL-92.
OpenGL	Open Graphics Library – відкрита графічна бібліотека, що визначає незалежний від мови програмування крос-платформовий програмний інтерфейс (API)
DVM	Dalvik Virtual Machine – заснована на регістрах віртуальна машина, створена як частина мобільної платформи Android.
USGS	United States Geological Survey – науково-дослідна урядова організація США
EMSC	European-Mediterranean Seismological Centre – міжнародна організація для оцінки сейсмічної небезпеки, а також швидкого визначення параметрів руйнівних землетрусів
IDE	Integrated Development Environment – комплексне програмне рішення для розробки програмного забезпечення
SWT	Standard Widget Toolkit – бібліотека з відкритим вихідним кодом для розробки графічних інтерфейсів користувача на мові Java
HTML	HyperText Markup – стандартизована мова розмітки документів для перегляду вебсторінок у браузері

ВСТУП

Сучасний світ стикається з непередбачуваними природними явищами, серед яких землетруси відіграють особливу роль у загрозі для безпеки людей і майна. З метою мінімізації ризиків і попередження можливих трагедій стає надзвичайно важливим розробка ефективних інструментів сповіщення та попередження. У цьому контексті мобільні додатки стають невід'ємною складовою системи безпеки суспільства.

Розробка мобільних застосунків, які автоматично надсилають сповіщення про землетруси, дозволяє значно підвищити рівень готовності та реагування населення на такі ситуації. Вибір мови програмування Kotlin для розробки такого застосунку обумовлений її сучасністю, надійністю та ефективністю в контексті створення додатків для платформи Android.

Актуальність кваліфікаційної роботи полягає у необхідності створення ефективного та швидкодіючого мобільного застосунку для автоматичного надсилання сповіщень про землетруси. З урахуванням постійного росту кількості користувачів мобільних пристроїв та їх популярності в різних сферах життя, важливість швидкого та достовірного інформування про небезпечні події, такі як землетруси, стає більш актуальною.

Для досягнення мети роботи необхідно вирішити наступні задачі:

1. Провести аналіз предметної області для визначення необхідних функціональних та нефункціональних вимог до системи.

2. Дослідити і обґрунтувати вибір існуючіх рішень, що існують на ринку, визначити переваги та недоліки, а також визначити новизну пропонованого застосунку.

3. Вибрати методи і алгоритми, інструменти та технології для розробки додатку, обґрунтувати вибір мови програмування Kotlin.

4. Описати архітектуру програмної системи, розробити діаграми та обґрунтувати вибір архітектурних рішень.

5. Спроектувати основні компоненти системи та визначити їх взаємодію.

6. Розробити структуру бази даних, описати сутності та їх взаємозв'язки.

7. Спроектувати інтерфейс користувача, створити прототипи основних екранів та описати їх функціональні можливості.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Загальні поняття Android-розробки

Розробка застосунків для Android – це процес створення програмного забезпечення, яке використовує за основу операційну систему Android. У розробці акцент здебільшого робиться на програмуванні, тестуванні та налагодженні застосунків, які користувачі можуть завантажити та використовувати на своїх пристроях.

Платформа Android складається з різних компонентів. Серед них – стандартні застосунки, які вже встановлені на пристроях, такі як Повідомлення або Календар. Також у склад платформи входять *програмні інтерфейси (API)*, які керують зовнішнім виглядом і поведінкою застосунків, а також різноманітні допоміжні файли та бібліотеки. При створенні власних застосунків розробники мають доступ до цих API для управління їх функціоналом. Кожен застосунок Android працює у власному процесі [4].

Щоб краще зрозуміти процес розробки застосунків для Android, перш за все важливо зрозуміти архітектуру цієї платформи. Архітектура Android складається з 4 основних рівнів.

Ядро Linux

Уявіть, що ядро Linux – це двигун автомобіля. Так само, як двигун надає енергію автомобілю, ядро Linux є основною складовою операційної системи Android. Ядро забезпечує основні системні послуги, такі як безпека та управління пам'яттю.

Рівень бібліотек

На рівні бібліотек вже є попередньо побудовані бібліотеки, такі як *SQLite*, *OpenGL* та *SSL*, які розробники можуть використовувати для створення своїх застосунків. Набір бібліотек подібний до набору інструментів для будівництва. SQLite, OpenGL та SSL – це інструменти в цьому наборі. Ці інструменти використовуються для керування базами даних, візуалізації графіки та гарантії безпечного мережевого зв'язку.

Рівень виконання

Рівень виконання відповідає за роботу коду застосунків. Він містить різні інструменти, зокрема *Android Runtime (ART)* та *Dalvik Virtual Machine (DVM)*. Простими словами, рівень виконання – це «диригент у оркестрі». Так само, як диригент керує оркестром і об'єднує різні інструменти для створення музики, рівень виконання об'єднує різні частини програми для створення працюючого застосунку.

Прикладний рівень

Це те, що користувач бачить на своєму пристрої. Це так зване «обличчя застосунку». Іншими словами, це те, що користувачі встановлюють і з чим взаємодіють.

Activity (Діяльність): Android застосунок містить одну або декілька діяльностей (activities). Кожна діяльність представляє собою контейнер, що містить користувацький інтерфейс та код для його виконання.

Наміри (intents) утворюють систему повідомлень в Android. Намір складається з дії, яку треба виконати (наприклад, переглянути, редагувати і т. д.) та даних до неї.

Дія – це завдання, яке слід виконати при отриманні наміру разом з даними. Наміри використовуються для запуску діяльностей та комунікації між різними частинами Android системи. Застосунок може приймати або відправляти наміри. При відправленні наміру фактично надсилається повідомлення системі для виконання певної дії, наприклад, запуску нової діяльності в поточному застосунку або відкриття іншої програми. Проте, просте надсилання наміру не гарантує автоматичного виконання. Для цього потрібно зареєструвати *приймач намірів (intent receiver)*, який отримує намір і повідомляє Android системі, що слід зробити: виконати завдання в

новій діяльності або запустити іншу програму. У випадку, якщо існує декілька приймачів для прийому наміру, можна створити інструмент, що дозволить користувачеві обрати необхідну дію.

Оскільки один намір може мати декілька приймачів, користувач самостійно обирає дію, яку слід виконати. Наприклад, під час тривалого натискання на зображення в галереї відображається інструмент вибору, який пропонує відправити зображення через електронну пошту або соціальні мережі, редагувати або видалити його і т. д. Якщо система не може знайти відповідний намір, а інструмент вибору не був створений розробником, то застосунок завершиться аварійно і виведе повідомлення про помилку виконання. Тому важливо переконатися, що вибори для комерційних або інших дій у даному застосунку створені належним чином [17, с. 129].

Віджети та видові вікна: Видове вікно (view) представляє собою основний елемент керування інтерфейсом у вигляді прямокутної області, де можна малювати і обробляти події.

Приклади видових вікон: *контекстне меню (ContextMenu)*, *меню (Menu)*, *поверхня малювання (SurfaceView)*. Віджети є більш розширеними елементами інтерфейсу, наприклад, прапорці з перемикачами, де можна вибрати один з декількох можливих станів. Віджети являють собою елементи керування, з якими взаємодіє користувач. До віджетів відносяться *кнопка (Button)*, *вибір дати (DatePicker)*, *галерея (Gallery)*, *прапорець (CheckBox)* та інше. Отже, видові вікна і віджети є елементами керування користувацьким інтерфейсом, проте видові вікна здатні виконувати не одну, а декілька функцій.

1.2. Аналіз існуючих мобільних застосунків

Насамперед визначимо критерії, за якими будемо оцінювати застосунки.

Надійність і точність інформації

Застосунок повинен надавати точну та достовірну інформацію про землетруси, включаючи їх магнітуду, місцезнаходження та потенційний вплив.

Швидкість сповіщення

Застосунок повинен надавати миттєві сповіщення про землетруси, щоб користувачі могли вжити необхідних заходів безпеки вчасно.

Геолокаційна служба

Застосунок повинен мати можливість використовувати геолокацію для надання інформації про землетруси, які відбуваються в безпосередній близькості до користувача.

Повний спектр функцій

Застосунок повинен мати широкий спектр функцій, включаючи картографічні дані, історію землетрусів, поради щодо безпеки та можливість встановлення нагадувань.

Оновлення та підтримка

Застосунок повинен регулярно оновлюватися для вдосконалення функціоналу та забезпечення сумісності з останніми версіями операційних систем.

Користувацький інтерфейс

Застосунок повинен мати зрозумілий та зручний інтерфейс, що дозволяє швидко та легко отримувати доступ до необхідної інформації.

Безпека даних

Застосунок повинен гарантувати конфіденційність та безпеку особистих даних користувачів, забезпечуючи відповідні заходи захисту інформації.

Рейтинги та відгуки

Перегляд рейтингів та відгуків користувачів дає оцінку ефективності застосунку.

Релевантні застосунки по запиту Earthquake в Google Play Market: *My Earthquake Alerts – Map, Earthquake Networks, LastQuake, Earthquake, Volcanoes & Earthquakes.* Також ці застосунки мають отримувати інформацію із достовірних джерел, наприклад United States Geological Survey та European-Mediterranean Seismological Centre [16, с. 120].

United States Geological Survey (USGS) – це агентство, яке займається вивченням геологічних явищ в Сполучених Штатах. Окрім цього, вони надають глобальну

інформацію про землетруси та інші природні явища, що відбуваються у всьому світі. Система моніторингу землетрусів, розроблена USGS, вважається однією з найбільш авторитетних та надійних у світі.

European-Mediterranean Seismological Centre (EMSC) – це некомерційна організація, яка спеціалізується на моніторингу та реєстрації землетрусів, які відбуваються в Європі та басейні Середземного моря. Їхній головний фокус – регіон Європи та Середземномор'я, однак вони також надають інформацію про землетруси по всьому світу. EMSC відомий своєю широкою мережею моніторингу та швидким реагуванням на події.

1. Надійність та точність інформації

- “My Earthquake Alerts – Map” користується даними від USGS та EMSC, що гарантує високу надійність та точність;
- “Earthquake Networks” використовує дані від USGS та EMSC, тому інформація має бути достовірною;
- “LastQuake” також використовує дані від EMSC, що забезпечує надійність інформації;
- “Earthquake” користується даними від USGS, тому можна очікувати достовірну інформацію;
- “Volcanoes & Earthquakes” надає інформацію про землетруси та вулканічні виверження, на основі даних від EMSC та USGS.

2. Швидкість сповіщення

- “My Earthquake Alerts – Map” швидко оновлює інформацію про землетруси та надсилає сповіщення;
- “Earthquake Networks” має режим живого спостереження, що дозволяє отримувати оперативні сповіщення;
- “LastQuake” також має можливість оперативного сповіщення про землетруси;
- “Earthquake” надає швидке оновлення інформації про землетруси;
- “Volcanoes & Earthquakes” має швидку реакцію на вулканічні виверження та землетруси.

3. Геолокаційна служба

- Всі вказані застосунки мають можливість використовувати геолокацію для надання інформації про землетруси в безпосередній близькості до користувача.

4. Повний спектр функцій

- “My Earthquake Alerts – Map”, “Earthquake Networks”, “LastQuake”, “Earthquake”, “Volcanoes & Earthquakes”, у всіх цих застосунках доступні картографічні дані, історія землетрусів, поради щодо безпеки та можливість встановлення нагадувань.

5. Оновлення та підтримка

- За звичайними умовами регулярні оновлення та підтримка надаються усіма цими застосунками.

6. Користувацький інтерфейс

- Інтерфейси кожного із застосунків є зрозумілими та зручними для користування.

7. Безпека даних

- Кожен із застосунків гарантує конфіденційність та безпеку особистих даних користувачів та дозволяє ознайомитися з політикою конфіденційності.

8. Рейтинги та відгуки

- Кожен з цих застосунків має позитивні відгуки та високі рейтинги у Google Play Market.

Всі застосунки мають дуже добрі відгуки та показники в цілому, тому складно відразу визначити, який з них являється найкращим. Однак, певні недоліки все ж таки існують. Давайте їх розглянемо.

Недоліки

“My Earthquake Alerts – Map”

Деякі користувачі відзначають нестабільну роботу застосунку, що може включати в себе проблеми із завантаженням даних, відображенням інформації або низьку продуктивність під час використання.

“Earthquake Networks”

Деякі користувачі повідомляють про можливі проблеми з точністю інформації в деяких випадках, що може включати в себе неточності в магнітуді, місцезнаходженні або інших параметрах землетрусів.

“LastQuake”

Користувачі зазначають проблеми з оновленням та відображенням даних, що може означати невчасне оновлення інформації про землетруси або неповну доступність даних у деяких випадках.

“Earthquake”

Застосунок може бути обмеженим у функціоналі порівняно з іншими аналогами, що може включати в себе менший набір доступних функцій або обмежену кількість інформації, яка надається користувачам.

“Volcanoes & Earthquakes”

Можливість спостереження за виверженнями вулкана може мати менше користі для деяких користувачів, оскільки вони менш поширені чи важливі з точки зору безпеки, порівняно з іншими природними явищами, такими як землетруси [23].

Зважаючи на отриману інформацію, “My Earthquake Alerts – Map” є найкращим варіантом серед наведених застосунків. Незважаючи на деякі скарги щодо нестабільності в роботі, він має широкий функціонал, надійне джерело інформації (USGS та EMSC) і швидкі сповіщення про землетруси. Тому, він став фундаментальним джерелом для нашого подальшого дослідження і створення власного застосунку.

1.3. Середовище розробки Android Studio та його аналоги

Android Studio, представлене Еллі Паверс на конференції Google I/O 16 травня 2013 року, є інтегрованим середовищем розробки (IDE) для платформи Android. Компанія Google випустила перший стабільний реліз Android Studio 1.0 8 грудня 2014 року, замінюючи плагін ADT для платформи Eclipse. Це середовище, побудоване на базі вихідного коду IntelliJ IDEA Community Edition від JetBrains, розвивається в рамках відкритої моделі розробки та поширюється під ліцензією Apache 2.0.

Android Studio підтримує розробку застосунків не лише для смартфонів і планшетів, але й для інших пристроїв на базі Wear OS, Android TV, Google Glass і Android Auto. Для старих проєктів, розроблених за допомогою Eclipse і ADT Plugin, доступний інструмент для автоматичного імпорту в Android Studio [2].

Середовище надає засоби для типових завдань розробки застосунків для Android, включаючи тестування на різних версіях платформи і проєктування для пристроїв з різними розмірами екрану. Android Studio має кілька додаткових функцій порівняно з IntelliJ IDEA, таких як нова система складання, тестування і розгортання застосунків на основі Gradle і підтримка безперервної інтеграції.

Щоб прискорити розробку, Android Studio містить колекцію стандартних елементів інтерфейсу та візуальний редактор для їх компонування. Також доступний майстер створення власних елементів оформлення для створення нестандартних інтерфейсів. Крім того, середовище має функцію завантаження типових прикладів коду з GitHub.

Також включені розширені інструменти рефакторингу, адаптовані під особливості платформи Android, перевірки сумісності з попередніми версіями, виявлення проблем з продуктивністю, моніторинг споживання пам'яті та оцінка зручності використання. У редактор доданий режим швидкого внесення правок. Система підсвічування, статичного аналізу та виявлення помилок, розширена підтримкою **Android API**. Інтегрована підтримка оптимізатора коду **ProGuard** та вбудовані засоби генерації цифрових підписів є присутніми в Android Studio. Інтерфейс для управління перекладами на інші мови теж доступний [19].

Наразі передбачені такі функції:

- живі макети (layout): редагувальник WYSIWYG, що дозволяє живе кодування і подання програми в реальному часі;
- консоль розробника: надання порад щодо оптимізації, допомога з перекладом, стеження за напрямком, аналіз метрик Google Analytics;
- бета-релізи та поетапні випуски;
- використання системи збирання Gradle;

- Android-орієнтований рефакторинг та швидкі виправлення;
- інструменти Lint для оцінки продуктивності, юзабіліті, сумісності версій та інших проблем;
- використання можливостей ProGuard та цифрових підписів для програм;
- шаблони для створення розширених дизайнів та компонентів Android;
- розширений редактор макетів, що дозволяє перетягувати компоненти користувацького інтерфейсу та переглядати макети на різних конфігураціях екранів одночасно.

Eclipse є вільним модульним інтегрованим середовищем розробки програмного забезпечення, що розробляється та підтримується Eclipse Foundation. **IDE** включає такі проєкти, як платформа Eclipse, набір інструментів для програмістів на **Java**, системи контролю версій, конструктори GUI та інше. Це основною мовою написано на Java і може використовуватися для розробки застосунків на Java, а також, за допомогою різних плагінів, на інших мовах програмування, таких як Ada, C, C++, C#, COBOL, Fortran, Perl, PHP, Python, R, Ruby, Scala, Clojure та Scheme. До середовища розробки входять Eclipse ADT (Ada Development Toolkit) для Ada, Eclipse CDT для C/C++, Eclipse JDT для Java, Eclipse PDT для PHP [21].

Початки створення Eclipse сягають **IBM VisualAge** і були спрямовані на розробників Java, що стало початком у створенні Java Development Tools (JDT). Проте користувачі могли розширювати можливості шляхом встановлення плагінів, написаних для платформи Eclipse, що охоплювали різні мови програмування, і також мали змогу створювати та впроваджувати власні плагіни та модулі.

Випущений під ліцензією Eclipse Public License, Eclipse є вільним програмним забезпеченням та став одним з перших інтегрованих середовищ розробки, яке працює без проблем під GNU Classpath і IcedTea.

Eclipse – це фреймворк для створення модульних платформонезалежних застосунків з рядом важливих особливостей:

- можливість розробки програмного забезпечення на різних мовах програмування (основна – Java);

- мультиплатформність;
- модульність, яка дозволяє розширювати функціонал незалежними розробниками;
- відкритий вихідний код;
- розробляється та підтримується фондом Eclipse, в який входять провідні постачальники програмного забезпечення, такі як IBM, Oracle, Borland.

Починаючи з версії 3.0, Eclipse перестав бути монолітним інтегрованим середовищем розробки, яке підтримує розширення, і став набором розширень. Його основу складають фреймворки OSGi та SWT/JFace, на основі яких побудований наступний рівень – платформа і засоби розробки повноцінних клієнтських застосунків RCP (Rich Client Platform). Платформа RCP служить основою для розробки різних програм, як наприклад торент-клієнт Azareus чи File Arranger.

Eclipse написаний на Java, тому є платформонезалежним продуктом, за винятком бібліотеки графічного інтерфейсу SWT, яка розробляється окремо для більшості поширених платформ. Бібліотека SWT використовує графічні засоби платформи (ОС), що забезпечує швидкість та звичний зовнішній вигляд інтерфейсу користувача [18, с. 342].

NetBeans IDE є вільним інтегрованим середовищем розробки (IDE), призначеним для програмування на Java, JavaFX, C/C++, PHP, JavaScript, HTML5, Python та Groovy. Це середовище може бути встановлене як для підтримки окремих мов, так і у повній конфігурації. За замовчуванням NetBeans IDE підтримує розробку для платформ J2SE і J2EE.

Починаючи з початкових версій, NetBeans IDE розповсюджується під ліцензією Apache License. Проєкт спонсорувався компанією Sun Microsystems, потім цю компанію купила – Oracle. У жовтні 2016 року Oracle передала права власності на NetBeans у Apache Software Foundation, яка тепер займається його розробкою та підтримкою.

NetBeans IDE підтримується на платформах Microsoft Windows, GNU/Linux, FreeBSD і Solaris (як SPARC, так і x86).

Останні версії NetBeans IDE пропонують широкі можливості, конкуруючи з найкращими інтегрованими середовищами розробки для Java. Вони включають рефакторинг, профілювання, підсвічування синтаксичних конструкцій, автодоповнення коду, шаблони коду та інші корисні функції. У випуску 7.4, який вийшов у жовтні 2013 року, розширено засоби розробки для використання технологій HTML5, додана підтримка створення гібридних HTML5-застосунків для платформ Android і Apple iOS з використанням фреймворку Apache Cordova, а також були реалізовані засоби використання HTML5 в проєктах Java EE і PHP. Крім того, була представлена експериментальна підтримка майбутнього випуску JDK8.

Microsoft Visual Studio є серією продуктів від компанії Майкрософт, які включають інтегроване середовище розробки програмного забезпечення та різноманітні інструменти для програмістів. Ці продукти дозволяють розробляти як консольні, так і графічні програми, включаючи ті, що використовують технологію Windows Forms, а також веб-сайти, веб-застосунки та веб-служби на платформах, що підтримуються Microsoft, таких як Windows, Windows Mobile, Windows Phone, Windows CE, .NET Framework, .NET Compact Framework та Microsoft Silverlight.

Версія Visual Studio з кодовим ім'ям Dev14 була випущена 20 червня 2015 року. Однією з важливих нововведень є розширена підтримка різних цільових платформ: крім основної платформи Windows, тепер існує можливість будувати проєкти для IOS та Android. Ця розширена функціональність відкриває нові можливості для розробників, дозволяючи їм створювати застосунки для широкого спектру пристроїв і платформ, що ще більше розширює сферу застосування Visual Studio [21].

Xcode – це інтегроване середовище розробки (IDE), створене компанією Apple, яке дозволяє розробляти програмне забезпечення за допомогою таких технологій, як GCC, GDB, Java та інші. Наразі це єдиний інструмент для написання "універсальних" (Universal Binary) застосунків для Mac OS X. Xcode включає в себе обширну документацію розробника від Apple та Interface Builder – інструмент для створення графічних інтерфейсів.

Пакет *Xcode* містить модифіковану версію вільного набору компіляторів GNU Compiler Collection і підтримує мови програмування C, C++, Objective-C, Swift, Java,

AppleScript, Python і Ruby, а також інші, включаючи Cocoa, Carbon і Java. Крім цього, сторонні розробники забезпечили підтримку GNU Pascal, Free Pascal, Ada, C#, Perl, Haskell і D. Для налагодження Xcode використовує GDB як back-end для свого засобу відлагодження.

У серпні 2006 року Apple анонсувала інтеграцію DTrace, фреймворку динамічного трасування від Sun Microsystems, який був випущений як частина OpenSolaris, в Xcode під назвою Xray. Пізніше Xray був перейменований в Instruments.

1.4. Kotlin – головний програмний інструмент

Розробка застосунків для Android стала необхідною складовою сучасного інформаційного середовища. Проте перед тим, як розпочати створення застосунку, важливо розглянути мови програмування для Android. Вибір може мати значний вплив на проєкт. Кожна мова має свої особливості, переваги та обмеження, які визначатимуть ефективність розробки, продуктивність застосунку та зручність підтримки у майбутньому [15, с. 75].

Мова програмування Kotlin вразила спільноту Android своєю передовістю та ефективністю. У 2017 році Google зробила Kotlin офіційною мовою розробки для Android, і це рішення прийнялося з великим задоволенням серед багатьох розробників. Давайте детальніше розглянемо, чому Kotlin вважається сучасним і перспективним вибором для створення застосунків для Android [8].

Чому Kotlin став офіційною мовою для Android:

Google обрав Kotlin як офіційну мову розробки для Android з декількох ключових причин:

1. Повна сумісність з Java, що дозволяє розробникам поступово впроваджувати Kotlin у наявні проєкти, не починаючи все з нуля.
2. Безпека типів, що дозволяє виявляти помилки на етапі компіляції і знижує ризик виникнення багів у застосунку.

3. Широкий набір сучасних функцій, таких як лямбди, функції вищого порядку і розширення, що робить код більш читабельним і дозволяє розробникам писати більш компактний і виразний код.
4. Зменшення кількості повторюваного (шаблонного) коду, що спрощує підтримку і розробку проєктів.

Однією з ключових переваг Kotlin є його нативна інтеграція з Android Studio – офіційним середовищем розробки для Android від Google. Це означає, що можна створювати застосунки на Kotlin з максимальним рівнем зручності та підтримки.

Android Studio надає розробникам широкий набір інструментів, а Kotlin доповнює їх та робить процес створення застосунків більш ефективним. Це дає доступ до автозаповнення коду, інтегрованих засобів налагодження та аналізу продуктивності, що спрощує розробку і підвищує якість кінцевого продукту.

Отже, Kotlin є сучасною, безпечною та продуктивною мовою програмування для Android, що забезпечує розробникам всі необхідні інструменти для створення якісних застосунків [20].

Класична Java. Ця мова програмування має важливе значення для розробки Android-застосунків. Java була першою офіційною мовою програмування для Android, тому Google обрав її з метою зробити розробку застосунків на цій платформі більш доступною і поширеною. За допомогою Java створено велику кількість застосунків, які досі успішно функціонують на мільйонах пристроїв.

Однією з важливих переваг Java для розробки на Android є її надійність. Застосунки, розроблені на цій мові, зазвичай працюють дуже стабільно і безпечно, що є важливим фактором для користувачів і забезпечує захист даних. Крім того, Java має велику спільноту розробників, що означає доступність до великої кількості ресурсів, підтримки, бібліотек і фреймворків, що значно спрощує розробку. Також, Java забезпечує портативність застосунків, що означає можливість легко переносити додатки на різні платформи, що підвищує переносимість вашого коду і знижує витрати на розробку.

Давайте розглянемо деякі складнощі, з якими можна зіткнутися під час використання Java порівняно з сучаснішими мовами, такими як Kotlin. Один з

основних складнощів полягає в тому, що для досягнення тієї ж функціональності в Java часто потрібно писати більше коду. Це може призвести до збільшення обсягу коду та складнішої структури застосунка, що у свою чергу може зробити розробку менш ефективною та ускладнити підтримку.

І на завершення, Java іноді відстає у впровадженні нових функцій порівняно з більш сучасними альтернативами. Тому існує ризик того, що використання останніх інновацій буде неможливим.

Давайте розглянемо інші мови програмування та визначимо їхні слабкі та сильні сторони:

Мова C++. Цю потужну мову програмування можна використовувати для розробки Android-застосунків за допомогою Android NDK (Native Development Kit). У яких випадках варто використовувати C++ з Android NDK?

C++ та Android NDK стають більш важливими, коли існують такі потреби:

- Забезпечити високу продуктивність застосунку. Наприклад, для ігор або графічних застосунків. Використання C++ дозволяє отримати доступ до низькорівневих системних ресурсів, що може покращити продуктивність вашого застосунку.
- Використовувати наявні бібліотеки на C/C++, які не мають нативної підтримки в Java або Kotlin. За допомогою Android NDK можна інтегрувати ці бібліотеки у свій проєкт Android.
- У деяких випадках можна використовувати цю мову для написання нативних функцій і бібліотек, які можна викликати з Java або Kotlin, що дозволяє підвищити швидкість виконання для критично важливих частин вашого застосунку.

Однак робота з цією мовою вимагає високого рівня експертизи. Які обмеження і проблеми можна очікувати:

1. Розробка на C++ є складнішою і трудомісткою, ніж на більш високорівневих мовах, таких як Java або Kotlin. Вам потрібно мати досвід у програмуванні на C++.

2. У відміну від Java, C++ не має автоматичного збирача сміття, тому вам самим доведеться керувати пам’яттю. Це може призвести до витоків пам’яті та помилок, пов’язаних з ними.

3. Код, написаний на C++, може бути платформозалежним, тому вам доведеться піклуватися про підтримку різних архітектур і версій Android.

4. Налагодження коду на C++ може бути складним завданням, особливо при використанні NDK. Виявлення і усунення помилок може бути складніше порівняно з використанням вищих мов програмування.

Використання C++ з Android NDK відкриває можливості для оптимізації продуктивності та інтеграції з наявними бібліотеками, але це також потребує додаткових зусиль і знань. Тому перед вибором цієї мови важливо ретельно оцінити вимоги вашого проєкту та рівень підготовки вашої команди розробників.

Мова C# з Xamarin. Цю мову програмування можна використовувати для створення Android-застосунків за допомогою платформи Xamarin. Давайте розглянемо переваги та недоліки використання C# для розробки Android-застосунків і як Xamarin розширює його можливості.

Переваги

1. Крос-платформеність: C# з Xamarin дозволяють створювати застосунки для Android, iOS та інших платформ зі спільним кодом, що економить час і ресурси.

2. Широкі бібліотеки .NET: розробники можуть скористатися багатою екосистемою .NET і сторонніми бібліотеками, що прискорює розробку.

3. Досвід розробника: для розробників, знайомих із C#, використання цієї мови з Xamarin може бути більш комфортним.

Недоліки

1. Складність взаємодії з нативним кодом: під час роботи з нативними API Android може виникати складність і потреба у додатковій роботі з ними.

2. Ресурсоємність: Xamarin-застосунки можуть мати більший обсяг і вимагати більше ресурсів пристрою порівняно з нативними застосунками.

3. Затримка оновлень: нові версії Android можуть вимагати оновлення Xamarin, що може призвести до затримок у доступі до останніх функцій.

Xamarin надає такі способи розширення можливостей C#.

- Xamarin.Forms: цей інструмент дозволяє створювати крос-платформні користувацькі інтерфейси зі спільним кодом для різних платформ, що скорочує витрати на розробку і забезпечує однаковий дизайн.
- Xamarin.Essentials: надає набір бібліотек із доступом до функцій пристрою (камера, GPS і датчики) за однаковим API для всіх платформ.
- Xamarin.Android: розробники можуть використовувати його для більш низькорівневої взаємодії з Android API і створення кастомних компонентів.

Python та інші інструменти. Використання Python разом із іншими інструментами для розробки Android-застосунків відкриває альтернативні можливості для розробників. Давайте розглянемо, як Python у поєднанні з бібліотекою Kivy, а також інші інструменти, такі як HTML, CSS, JavaScript та Adobe PhoneGap, можуть бути використані для створення Android-застосунків.

Використання Python з Kivy. Kivy – це крос-платформена бібліотека для розробки мобільних застосунків. Вона дозволяє створювати інтерактивні та мультимедійні застосунки з використанням Python. Проте варто пам'ятати, що продуктивність Python може бути не такою високою, як у Java або Kotlin, тому Kivy найкраще підходить для невеликих і середніх проєктів [11, с. 119].

HTML, CSS і JavaScript – ці веб-технології можна використовувати для створення гібридних застосунків для Android за допомогою фреймворків, таких як **Adobe PhoneGap** (також відомий як Apache Cordova). Це дозволяє розробникам використовувати знайомі веб-технології для створення застосунків, які можуть працювати на різних платформах, включаючи Android.

Adobe PhoneGap – це фреймворк, який надає доступ до нативних функцій пристрою через JavaScript API. Він дозволяє створювати крос-платформні застосунки з спільним кодом, що спрощує розробку та зменшує витрати.

Вибір між Python з Kivy і HTML/CSS/JavaScript з Adobe PhoneGap залежить від ваших вподобань, навичок і вимог проєкту. Python з Kivy може бути більш кращим для мультимедійних застосунків, тоді як HTML/CSS/JavaScript може забезпечити крос-платформеність і використання веб-технологій.

Аналіз основних характеристик мов програмування для розробки Android-додатків:

Таблиця 1.1

Порівняння мов програмування

Мова	**Продуктивність**	**Складність розробки**	**Підтримка спільноти**	**Додаткова інформація**
Kotlin	Високий рівень	Низька складність	Активна та зростаюча	Офіційна мова програмування для розробки на Android
Java	Середній рівень	Середня складність	Велика спільнота	Має довгу історію використання на платформі Android
C++	Високий рівень	Висока складність	Велика спільнота	Використовується для розробки нативних додатків та ігор
C#	Середній рівень	Середня складність	Значна підтримка	Використовується для крос-платформеного програмування.
Python	Різна (висока для CPython)	Низька складність	Велика спільнота	Найчастіше використовується з бібліотекою Kivy

У підсумку, перед розробкою застосунку для платформи Android, вибір мови програмування є ключовим етапом, який визначить ефективність процесу розробки та якість кінцевого продукту. Kotlin і Java є двома найбільш популярними мовами для розробки Android-застосунків, кожна з яких має свої переваги і особливості.

Необхідно уважно враховувати ваші власні знання та досвід, а також потреби проєкту при виборі мови. Kotlin відомий своєю сучасністю і продуктивністю, тоді як

Java є класичним вибором з великою спільнотою та довгою історією на платформі Android [14].

Крім цього, не слід забувати про інші альтернативи, такі як C++, C#, Python з бібліотекою Kivy та використання веб-технологій з Adobe PhoneGap, які також можуть бути відповідними для певних проєктів з урахуванням їх унікальних особливостей.

1.5. Push-сповіщення для взаємодії із застосунком

Для більшості мобільних застосунків стає необхідністю мати сервіс обміну повідомленнями. Незалежно від типу створеного застосунку, він має функцію обміну інформацією між програмою та користувачем.

Ця функція, хоча й корисна для будь-якого застосунку, але водночас ускладнює роботу розробників. Саме через це різні розробники використовують різні платформи для інтеграції сервісів обміну повідомленнями у свої програми. Таким чином, розглянемо декілька оптимальних платформ, які можна використовувати для цієї мети.

One Signal – це платформа, яка відповідає зростаючим потребам сервісів обміну повідомленнями у сучасних умовах швидкого розвитку онлайн-бізнесу. Вона пропонує високоякісні послуги обміну повідомленнями, і саме завдяки ефективно працюючим сервісам на цій платформі компанія стала однією з провідних на ринку постачальників послуг обміну повідомленнями [5, с. 500].

Більшість клієнтів – це підприємства. Вони співпрацюють з понад мільйоном розробників по всьому світу, які використовують їхні послуги обміну повідомленнями, щодня відправляючи понад 8 мільярдів повідомлень.

One Signal пропонує широкий спектр функцій, які користувачі можуть використовувати:

Push-сповіщення для мобільних застосунків

One Signal надає потужні інструменти для відправлення push-сповіщень на мобільні пристрої, що дозволяє додаткам ефективно спілкуватися зі своїми користувачами.

Push-сповіщення для веб-застосунків

Крім мобільних застосунків, One Signal дозволяє відправляти push-сповіщення на веб-сторінки, щоб залучати користувачів і повідомляти їх про важливі події.

Служби обміну повідомленнями в застосунках

One Signal може служити як потужний інструмент для обміну повідомленнями всередині додатків, що дозволяє створювати динамічні сповіщення та взаємодіяти з користувачами.

Управління SMS

Платформа також надає можливості управління SMS, що дозволяє відправляти текстові повідомлення користувачам безпосередньо з додатків.

Послуги та управління електронною поштою

One Signal дозволяє автоматизувати та керувати розсилкою електронних листів, що сприяє збільшенню залучення користувачів через електронну пошту.

Незважаючи на те, що One Signal володіє вражаючим набором функцій, варто мати на увазі, що ця платформа, хоч і є лідером у своєму сегменті, може не відповідати усім потребам додатку. У таких випадках можна розглянути альтернативні платформи, які пропонують більше функціональності або відповідають конкретним вимогам проєкту [17, с. 167].

Firebase – це керований бекенд-сервіс, розроблений і підтримуваний компанією Google. Ця платформа надає широкий спектр інструментів і сервісів, що дозволяють розробникам створювати та підтримувати додатки без необхідності власного управління інфраструктурою серверів і хостингу.

Основні переваги Firebase полягають у простоті використання і готовності до використання під ключ різних функцій, що включають:

Push-сповіщення

Firebase дозволяє надсилати push-сповіщення на мобільні пристрої користувачів, що дозволяє ефективно залучати аудиторію та сповіщати про важливі події.

Служби обміну повідомленнями в додатках

Firebase пропонує інструменти для реалізації обміну повідомленнями всередині додатків, що дозволяє створювати інтерактивні та персоналізовані досвіди для користувачів.

Швидка розробка з інтеграцією готових функцій

Firebase має набір готових функцій, таких як аутентифікація користувачів, бази даних, сховище файлів, аналітика тощо, що дозволяє розробникам швидко створювати і запускати додатки з мінімальними зусиллями.

Інтеграція з алгоритмами машинного навчання

Firebase дозволяє використовувати інтелектуальні функції, такі як машинне навчання, для покращення функціональності додатків і персоналізації взаємодії з користувачами.

Аутентифікація та управління користувачами

Firebase має вбудовані засоби аутентифікації користувачів і управління доступом, що дозволяє безпечно ідентифікувати користувачів і керувати їхніми правами доступу.

Хмарні сервіси і хостинг

Firebase надає хмарні сервіси для зберігання даних, виконання функцій на стороні сервера і надання доступу до різних сервісів Google Cloud.

Аналітика та звіти

Firebase забезпечує інструменти для збору та аналізу даних про використання додатків, що дозволяє розробникам отримувати цінну інформацію для покращення продуктів.

Отже, Firebase є потужною платформою, що дозволяє розробникам створювати інноваційні та високопродуктивні додатки без зайвих складнощів у керуванні інфраструктурою.

Pusher – це інноваційна платформа, спрямована на креативних розробників, що мають бажання реалізувати свої унікальні ідеї у вигляді додатків і потребують можливостей роботи з даними та функціональними сервісами в режимі реального часу. Вона пропонує інтегроване рішення для уникнення складнощів, пов'язаних з управлінням серверами та клієнтами для обміну повідомленнями.

API-інтерфейси Pusher значно спрощують процес розробки, дозволяючи легко інтегрувати їх у застосування та отримувати всі необхідні сервіси.

Особливості Pusher включають:

1) повідомлення push;

2) оновлення у режимі реального часу;

3) сумісність з різними платформами розробки додатків, незалежно від типу (мобільні або веб-застосунки);

4) система наскрізного шифрування для надання безпеки користувачам;

5) інтегроване кероване підключення до WebSocket;

6) параметри масштабованості для забезпечення ефективності та розширюваності.

CleverTap – це платформа, спрямована на поліпшення взаємовідносин з клієнтами, оскільки вона зосереджується на двох ключових аспектах: обміну сповіщеннями в реальному часі та аналітиці поведінки користувачів. Ці послуги дозволяють покращувати маркетингові стратегії бізнесу та оптимізувати взаємодію з клієнтами.

Основні характеристики платформи CleverTap включають:

1) CleverTap дозволяє відправляти повідомлення на різні канали зв'язку для залучення користувачів;

2) платформа забезпечує впевненість у безпеці ідентифікації користувачів;

3) CleverTap збирає та аналізує історію взаємодії користувачів з додатком для уточнення маркетингових стратегій;

4) платформа дозволяє відслідковувати ключові події та взаємодію користувачів з додатком для розуміння їхнього поведінкового шаблону;

5. CleverTap забезпечує стабільну та надійну роботу для забезпечення безперебійної роботи сервісу.

Часом вам може знадобитись платформа, яка не лише надає різноманітні сервіси, але й спеціалізується на послугах обміну повідомленнями, щоб краще відповідати вашим потребам. Twilio Notify – саме така платформа, якій можна довіритися. Вона має високопрофесійні функції і дозволяє інтегрувати всі необхідні функції вашого додатка через один виклик API, що значно спрощує роботу розробників [3].

Особливості Twilio Notify:

1) Twilio Notify забезпечує можливість відправки SMS-повідомлень по всьому світу;

2) платформа підтримує відправку push-повідомлень на мобільні пристрої;

3) Twilio Notify дозволяє відправляти масові сповіщення одночасно багатьом користувачам;

4) платформа надає можливість керувати списком відправлених повідомлень;

5) Twilio Notify дозволяє керувати реєстрацією користувачів та їхніми сервісами.

В різноманітних програмах виникають різні вимоги до їх функціоналу. Проте одна функція, яка практично завжди потрібна у будь-якому додатку – це можливість отримування сповіщень.

Різні програмісти використовують інструменти, такі як One Signal, для впровадження функцій обміну повідомленнями у свої додатки, оскільки в іншому випадку це може бути складно здійснити. Однак One Signal не є ідеальним вибором для всіх випадків.

Тому ми розглянули декілька альтернатив One Signal з подібними функціями, які можна використовувати. Деякі з цих альтернатив мають додаткові функції, що роблять їх кращими у конкретних випадках використання.

РОЗДІЛ 2
РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ

2.1. Огляд API для отримання даних про землетруси

Обирання відповідного API для мобільного застосунку є критичним аспектом розробки програмного забезпечення, оскільки воно безпосередньо впливає на функціональність, ефективність та користувацький досвід додатка. У контексті розробки мобільного застосунку для отримання даних про землетруси, обрання оптимального API стає ключовою задачею для забезпечення надійності та швидкості отримання необхідної інформації.

Зручність використання API визначається його здатністю відповідати специфічним вимогам мобільного додатка, забезпечуючи оптимальну швидкість, стабільність і безпеку передачі даних. Наприклад, мобільні застосунки часто мають обмежені ресурси пристроїв, тому API повинне бути оптимізоване для ефективного використання цих обмежень.

Важливо також враховувати, що вибір API повинен базуватися на його спроможності забезпечувати достатню кількість і якість даних про землетруси, що необхідні для задоволення вимог функціональності вашого додатка. Наприклад, API повинне надавати не лише базову інформацію про землетруси, але й детальні параметри, такі як глибина, магнітуда, координати та інші. У даній області існує кілька API, які можна розглядати для використання у мобільному додатку. Кожне з цих API має сво унікальні особливості. Наприклад, доступність даних в реальному часі, обсяг і якість інформації, а також рівень документації та підтримки, що надаються розробникам.

Оглянемо декілька API для отримання даних про землетруси, включаючи їх функціональність, можливості та способи використання. Такий підхід дозволить здійснити інформований вибір API, який найкращим чином підходить для реалізації задуманого мобільного застосунку [11, с. 112].

USGS Earthquake API

API, наданий Геологічною службою США (USGS) є чудовим інструментом для отримання даних про землетруси з метою проведення прогностичного аналізу, оцінки ризику та дослідження тенденцій.

Розробник може використовувати це API для отримання надійних та актуальних даних про землетруси безпосередньо у своєму застосунку. API охоплює міжнародні землетруси, надає дані в реальному часі та підтримує підписку на географічні області, що дозволяє розробникам швидко отримувати та використовувати дані про землетруси у своїх проєктах. Крім того, API має словник даних із стандартизованими ключами та мітками, що спрощує розуміння та обробку наданих даних про землетруси.

Інші характеристики API включають доступ до історичних та реальних даних про землетруси, а також надання даних в реальному часі на підставі підписки на географічні області.

Цей API може бути використаний для створення погодного додатка з функцією прогнозу/попередження про землетруси, який використовує USGS Earthquake API для надсилання сповіщень у разі прогнозованих або виявлених землетрусів.

Додаткові варіанти використання цього API включають створення калькулятора страхових внесків, який використовує USGS Earthquake API для надання оцінки внесків на майно або області на основі їх ризику зазнати землетрусу [10].

EMSC Earthquake API

API Європейського середземноморського сейсмічного центру (EMSC) надає дані про землетруси в реальному часі. Це включає інформацію про найновіші землетруси разом з характеристиками та географічним розташуванням подій. API дозволяє здійснювати пошук землетрусів за різними параметрами, такими як магнітуда, глибина та час.

Механізм виявлення землетрусів EMSC ґрунтується на використанні систем, що базуються на участі громадськості, де свідки землетрусу перетворюються на сенсори реального часу для сейсмічного моніторингу. Інформаційна система EMSC включає в себе декілька компонентів: веб-сайт для робочого столу, веб-сайт для мобільних пристроїв, мобільний додаток під назвою LastQuake та Twitter-бот quakebot.

Коли люди відчувають раптовий трус, вони негайно звертаються до Інтернету (соціальні мережі, блоги, новини і т. д.), щоб отримати інформацію про цей трус. Ця швидка концентрація осіб, які шукають інформацію, призводить до масштабного збільшення трафіку на веб-сайт EMSC, одночасного запуску мобільного додатка LastQuake та зростання кількості твітів із ключовим словом "землетрус" на різних мовах.

Ці три методи взаємодоповнюються і дозволяють EMSC виявляти землетруси протягом 15–120 секунд від початку землетрусу без аналізу сейсмічних сигналів. Для локалізації землетрусу без аналізу сейсмічних даних EMSC потребує інформації про IP-адреси відвідувачів веб-сайту, місцезнаходження мобільних відвідувачів веб-сайту, місцезнаходження користувачів додатка LastQuake та описове місце в профілі користувача на Twitter.

Значимість цього механізму полягає в тому, що він дозволяє швидко виявляти землетруси без використання традиційних сейсмічних методів аналізу.

IRIS (Incorporated Research Institutions for Seismology) DMC (Data Management Center) API

IRIS DMC API є інтерфейсом програмування додатків, що надає доступ до сейсмічних даних та послуг. IRIS є міжнародною науковою організацією, яка об'єднує більше 150 установ досліджень з сейсмології з усього світу.

Основна мета IRIS DMC API полягає у забезпеченні доступу до глобальних сейсмічних даних для досліджень, моніторингу та розробки додатків, пов'язаних із землетрусами. Цей API дозволяє користувачам отримувати доступ до різних видів сейсмічної інформації, включаючи дані про землетруси, сейсмічні станції, сейсмограми та інші параметри [9]. Розглянемо основні характеристики IRIS DMC API.

Доступ до глобальних даних про землетруси

API надає доступ до інформації про останні землетруси з усього світу, включаючи їх магнітуду, глибину, час та місце.

Доступ до сейсмічних станцій

Користувачі можуть отримувати дані про сейсмічні станції, їх розташування та набори даних сейсмограм.

Різноманітність форматів даних

API підтримує різноманітні формати виводу даних, такі як QuakeML, SAC, MiniSEED та інші, що дозволяє користувачам обробляти дані у відповідному форматі для їхніх потреб.

Розширені можливості пошуку і фільтрації

Користувачі можуть виконувати розширений пошук та фільтрацію даних за різними параметрами, такими як географічні координати, часовий інтервал, магнітуда та інші.

Підтримка розширених аналітичних запитів

API дозволяє виконувати складні аналітичні запити для обробки та аналізу сейсмічних даних, включаючи розрахунок параметрів землетрусів і відтворення сейсмограм.

Вибір найкращого API для отримання даних про землетруси може залежати від конкретних потреб користувача і характеристик кожного API. Однак, у контексті надійності, доступності даних, функціональності та підтримки спільноти розробників, найбільш підходящим вибором може бути USGS Earthquake Hazards Program API.

USGS Earthquake Hazards Program API є одним з найрозповсюдженіших та найбільш довірених джерел сейсмічних даних у світі. Основні переваги USGS API включають.

Надійність даних

USGS відомий своєю високою якістю та достовірністю сейсмічних даних, які використовуються як науковцями, так і розробниками.

Доступність і актуальність

API надає доступ до оновлюваних даних про землетруси у реальному часі, що дозволяє використовувати їх у застосунках, які вимагають оперативного оновлення інформації.

Функціональність

USGS API дозволяє отримувати різні параметри землетрусів, такі як магнітуда, глибина, місце події та інші важливі дані для наукових досліджень і розробки застосунків.

Підтримка і документація

USGS надає добре документоване API з активною спільнотою розробників, що сприяє вирішенню проблем і обміну досвідом.

Отже, USGS Earthquake Hazards Program API є найкращим вибором для тих, хто шукає надійне джерело сейсмічних даних з широким спектром функціональностей і високою підтримкою спільноти розробників. Вибір API також може залежати від географічного контексту та конкретних вимог проєкту, але USGS API зазвичай визнаний як один з найкращих ресурсів у цій галузі.

2.2. Проектування мобільного застосунку

Міцна базова архітектура є ключовим фактором масштабованості додатка. Внесення таких змін, як заміна API на оновлену і оптимізовану структуру, вимагає практично повного переписування додатка [6].

Це пояснюється тим, що код тісно пов'язаний з модулем відповіді на певну дію користувача. Використання чистої архітектури допомагає вирішити цю проблему. Це найкращий підхід для великих додатків з великою кількістю функцій і принципами SOLID. Цю концепцію запропонував Роберт С. Мартін (відомий як Дядько Боб) у своєму блозі «Чистий код» у 2012 році.

Принципи SOLID є основними принципами розробки програмного забезпечення, які дозволяють створити якісний код, котрий буде добре масштабуватися та підтримуватися [12, с. 88].

S – принцип єдиної відповідальності (*Single Responsibility Principle*). Кожен клас повинен мати лише одну область відповідальності.

O – принцип відкритості-закритості (*Open Closed Principle*). Класи повинні бути відкритими для розширення, але закритими для змін.

L – принцип підстановки Лісков (*Liskov Substitution Principle*). Має бути можливість підставити будь-який підтип (клас-спадкоємець) замість базового (батьківського) типу (класу), при цьому робота програми не повинна змінитися.

I – принцип розділення інтерфейсів *(Interface Segregation Principle*). Цей принцип означає, що не потрібно змушувати клієнта (клас) реалізувати інтерфейс, з яким він не має стосунку.

D – принцип інверсії залежностей (*Dependency Inversion Principle*). Модулі верхнього рівня не повинні залежати від модулів нижнього рівня. Обидва рівні повинні залежати від абстракції. Абстракція не повинна залежати від деталей. Деталі повинні залежати від абстракцій.

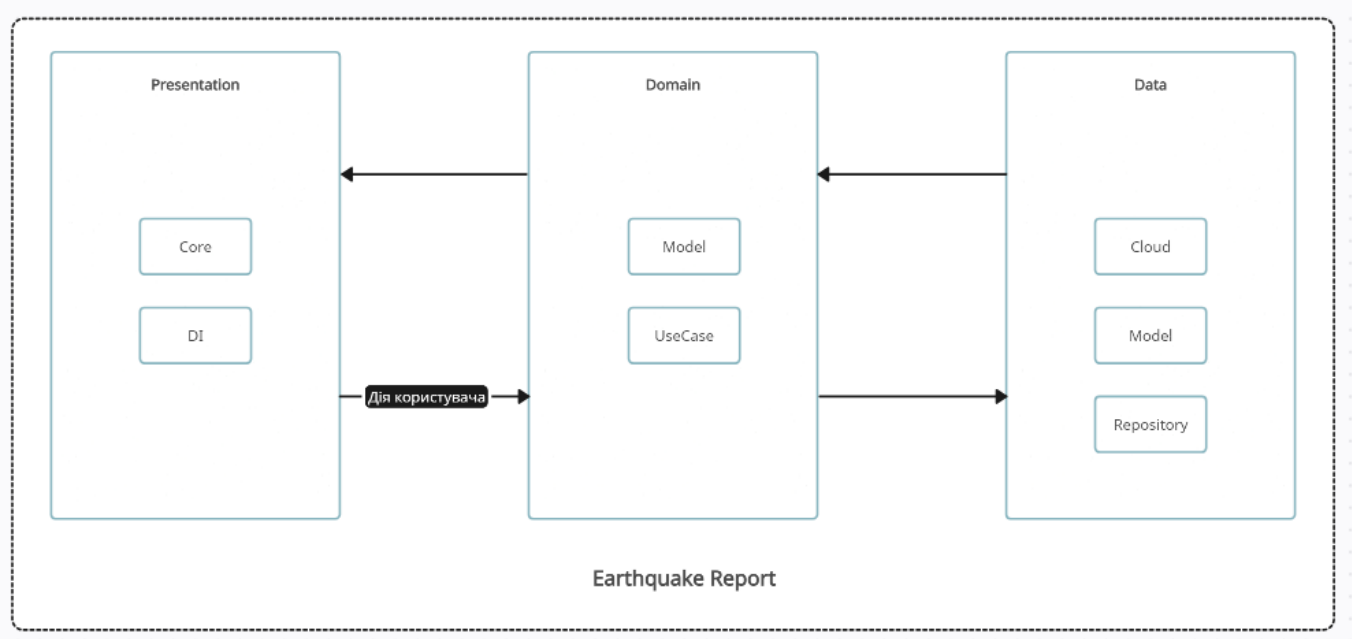

Рис. 2.1. Структура рівнів застосунку

Проєкт має бути розділений на три рівні: Presentation, Domain, Data.

Data Layer

Цей рівень містить дані. Інтерфейси API, бази даних та реалізація репозиторіїв знаходяться саме на цьому рівні (див. Додаток Б).

У чистій архітектурі рівень даних виконує функцію керування потоком даних та отримання інформації з різних джерел, таких як бази даних, веб-сервіси та файли. Крім того, він відповідає за постійне зберігання даних і сприяє доступу до них. Рівень даних також служить інтерфейсом для безперешкодної взаємодії між іншими рівнями додатка та даними. Він діє як абстрактний рівень, захищаючи дані від прямої взаємодії з іншими частинами застосунку.

Domain Layer

На цьому рівні містяться моделі (сутності, що містять дані) та інтерфейси репозиторіїв. Одночасно в цьому рівні зберігається бізнес-логіка проєкту разом з використанням Use Case.

Рівень домену виступає як центр бізнес-логіки, охоплюючи завдання, такі як конвертація даних, фільтрація, злиття та сортування. Його завданням є перетворення сирих даних, отриманих з рівня даних, в оброблений формат, який зручно і керовано обробляється рівнем презентації (див. Додаток В).

Use Case представляє собою процеси або функції, які є необхідними для виконання операцій. Взявши за приклад банківську діяльність, Use Case може включати відкриття рахунку, створення депозиту, закриття рахунку, замовлення картки і так далі. Ці Use Case є ключовими складовими, без яких функціонування банку буде неможливим.

Intercator (або Interactor) визначається як механізм, який об'єднує кілька Use Case для виконання певних операцій або функцій. У термінах об'єктно-орієнтованого програмування, Intercator може бути класом, в якому описані різні Use Case. Такий підхід дозволяє зробити код більш компактним та організованим, об'єднуючи пов'язані операції в один компонент.

Presentation Layer

Рівень презентації виступає як інтерфейс між користувачем та додатком, обробляючи введення користувача та відображаючи відповідні результати. Цей рівень зазвичай складається з графічного інтерфейсу користувача, такого як веб-сторінка, мобільний додаток або програма для настільного комп'ютера. У архітектурі MVVM рівень презентації включає об'єкти виду (view) та моделі виду (view model). Види, які представлені активностями (Activities) або фрагментами (Fragments), повинні залишатися простими та вільними від бізнес-логіки. Якщо у видів існує будь-яка бізнес-логіка, важливо відповідно рефакторити ці класи (див. Додаток А).

Компонент DI (Dependency Injection) забезпечує включення всіх залежностей при запуску додатка, таких як мережеві з'єднання, моделі видів (View Models), прецеденти тощо. DI реалізується за допомогою інструментів, таких як Dagger, Hilt, Koin або шаблону Service Locator, залежно від типу додатка. Я вибрав Hilt, оскільки його легше зрозуміти й впровадити, ніж Dagger [7].

Навіщо використовувати ViewModels? Згідно з документацією Android, ViewModel зберігає та керує даними користувацького інтерфейсу з урахуванням життєвого циклу. За його допомогою дані залишаються недоторканими при зміні конфігурації, наприклад, під час обертання екрану.

Рівень персистентності повинен включати класи, що відповідають за зберігання та отримання даних. У нашому випадку ці класи представлені об'єктами бібліотеки Realm.

Мережевий рівень має включати класи, які взаємодіють з даними з сервера. Ми будемо використовувати класи Retrofit 2. Retrofit 2 надає засоби створення повноцінного REST-клієнта, який може виконувати HTTP-запити типу POST, GET, PUT, DELETE. Для визначення типу та інші параметри запиту використовуються аннотації. Наприклад, для вказівки GET-запиту необхідно використовувати аннотацію **@GET** перед методом, для POST-запиту – **@POST** і так далі. У дужках після аннотації вказується адреса, до якої буде відправлено запит.

Переміщення даних з будь-якого з цих рівнів до бізнес-рівня вимагає прив'язки об'єктів збереження та мережевих об'єктів до домену.

Рівень домену має бути розділений від рівнів даних. Рівень даних не повинен знати про модель домену. Домен запитує дані з рівня даних, рівень даних отримує дані, пропускає їх через маппер і таким чином повертає модель домену.

Загальна модель для рівнів даних при отриманні даних з різних джерел порушувала б принцип розділення завдань. Моделі персистентності і мережеві моделі представляють різні частини системи і повинні бути відображені різними моделями. Домен не повинен знати про це, тому дані, що запитуються, мають бути зіставлені з об'єктами домену перед тим, як вони перетинають межу та повертаються назад до домену.

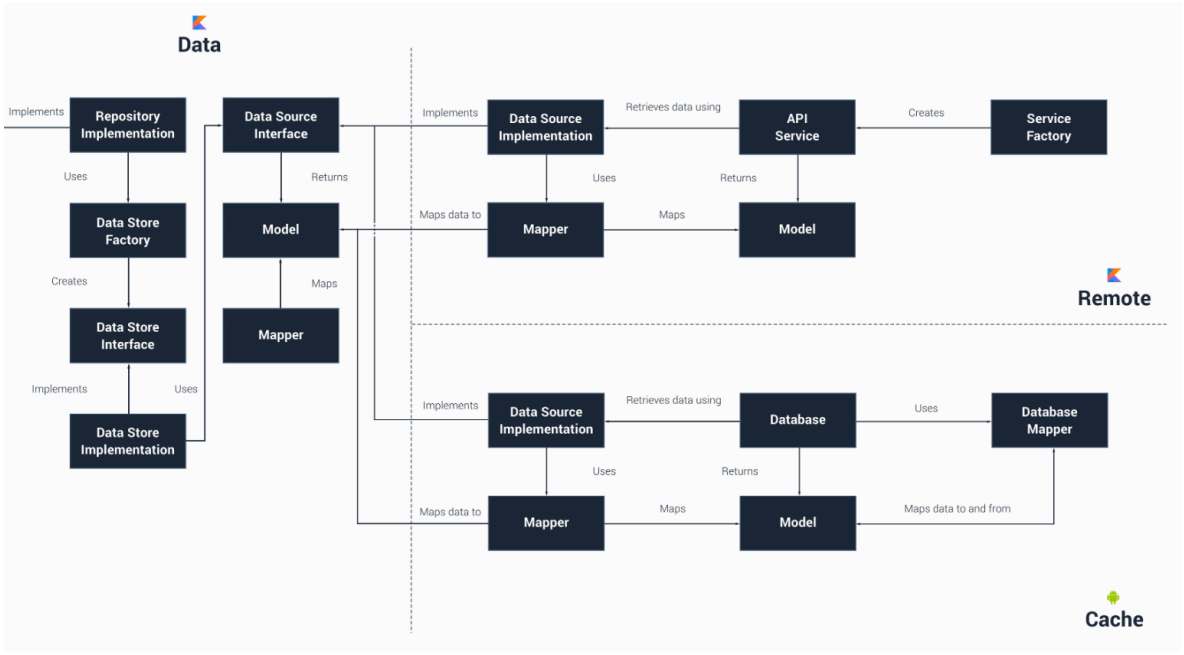

Рис. 2.2. Загальна модель відображення даних

2.3. Інтеграція обраного API з мобільним застосунком

Інтеграцію API USGS для отримання інформації про землетруси у мобільному додатку здійснимо за допомогою бібліотеки Retrofit 2 в середовищі Android. Retrofit – це потужний інструмент для роботи з HTTP-запитами в Android, який дозволяє зручно використовувати REST API.

Для початку необхідно додати Retrofit до проєкту Android. Встановити залежність Retrofit у файл **build.gradle** застосунку.

implementation 'com.squareup.retrofit2:retrofit:2.9.0'

implementation 'com.squareup.retrofit2:converter-gson:2.9.0'

Для того, щоб ефективно використовувати цю бібліотеку, потрібно визначити базове посилання. Базовий URL завжди має закінчуватись слешем "/". Він встановлюється за допомогою методу baseUrl(). Якщо в запиті вказано повний URL, то базовий URL буде проігноровано.

Для того, щоб отримувати списки даних використовується наступне посилання: https://earthquake.usgs.gov/fdsnws/event/1/query?format=geojson&limit=10. Базове посилання матиме такий вигляд https://earthquake.usgs.gov/ з тією метою, щоб в майбутньому стало легше підтримувати застосунок та швидко вносити зміни в HTTP-запити.

Екземпляр Retrofit має наступний вигляд:

```
object RetrofitInstance {
    private val retrofit = Retrofit.Builder()
        .baseUrl("https://earthquake.usgs.gov/")
        .addConverterFactory(GsonConverterFactory.create())
        .build()
    val api: ApiService = retrofit.create(ApiService::class.java)
}
```

Далі слід створити інтерфейс, в якому буде метод, який здійснюватиме необхідний запит та встановити анотацію @GET, в яку потрібно вказати решту посилання, тобто fdsnws/event/1/query?format=geojson&limit=10.

```
interface ApiService {
    @GET("fdsnws/event/1/query?format=geojson&limit=10")
    suspend fun getEarthQuakeReport(): Response<EarthquakeReportData>
}
```

Реалізація буде відбуватися на Data рівні, тому відповідно ці класи мають бути в пакеті: com.earthquakereport.***data***.cloud.api.

Після цього треба розробити модель, яка дозволить маніпулювати даними. Для її коректного створення необхідно відвідати посилання, зазначене в USGS API,

аналізувати типи даних, що містяться у форматі JSON, а також переглянути інформацію, що зберігається в конкретних змінних.

Рис. 2.3. Дані на сервері в json форматі

Найбільш інформативні поля включають: mag (магнітуда), place (місце), time (час коли відбувалася певна подія), url (посилання на обширну інформацію). Для того, щоб швидко створити необхідну модель в Kotlin, можна скопіювати весь json та вставити в плагін JSON To Kotlin Class.

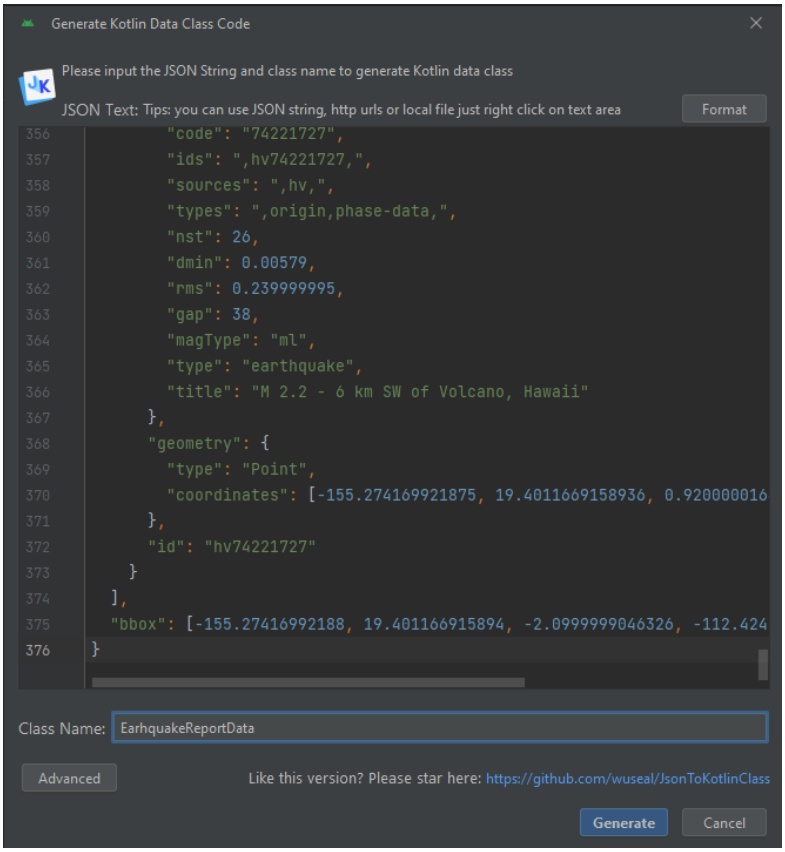

Рис. 2.4. Генерація моделі за допомогою плагіну

В результаті буде створено повноцінну модель EarthquakeReportData, котра міститиме такі дані, як: bbox, features, metadata, type. А вже безпосередньо в класі Feature ми можемо дістати ті дані, які необхідні для відображення інформації застосунку.

```
data class EarthquakeReportData(
    val bbox: List<Float>,
    val features: List<Feature>,
    val metadata: Metadata,
    val type: String
)

data class Feature(
    val geometry: Geometry,
    val id: String,
    val properties: Properties,
```

```
    val type: String
)
```

Таких даних як mag, place, time, url в Feature немає, тому що існує такий клас як Properties, в якому вже є всі необхідні дані. Інші дані легко можна буде використовувати в майбутньому.

```
data class Properties(
    val alert: Any,
    val cdi: Any,
    val code: String,
    val detail: String,
    val dmin: Any,
    val felt: Any,
    val gap: Any,
    val ids: String,
    val mag: Float,
    val magType: String,
    val mmi: Any,
    val net: String,
    val nst: Any,
    val place: String? = null,
  val url: String
...)
```

Тепер можна успішно виконати запит із типом EarthquakeReportData, в якому отримаємо всі вище зазначені дані.

```
        override suspend fun getEarthquakeReportData(): EarthquakeReportData? {
    return try {
        RetrofitInstance.api.getEarthQuakeReport().body()
    } catch (e: Exception) {
        e.printStackTrace()
```

```
        null
    }
}
```

Обов’язково потрібно додавати обробку виключень **try catch** для того, щоб у разі нестабільного підключення або відсутності інтернет підключення, в програмі не виникали збої.

```
override suspend fun getEarthquakeReport(checkFirstOpen: Boolean):
List<EarthquakeDomain> {
    val earthquakeReportData = cloudDataSource.getEarthquakeReportData()
    return if (checkFirstOpen) {
        toEarthquakeDomainMapper.map(
            earthquakeReportData,
            checkFirstOpen
        )
    } else {
        toEarthquakeDomainMapper.map(
            null,
            checkFirstOpen
        )
    }
}
```

У випадку успішного отримання даних, вони записуються в базу даних. У разі якщо не буде доступу до інтернету, або ж щоб не навантажувати застосунок запитами, всю інформацію будемо відображати безпосередньо отримуючи з бази даних. Якщо додаток запущений перший раз, то здійснюватиметься HTTP-запит. Якщо застосунок запущений не перший раз, то в базі даних вже має бути збережена якась інформація. Важливо здійснити перевірку: якщо база даних містить такі самі дані як і в HTTP-запиті, то інформація буде відображатися з бази даних.

```
        override suspend fun map(
    earthquakeReportData: EarthquakeReportData?,
```

```
    checkFirstOpen: Boolean
): List<EarthquakeDomain> {
    return withContext(Dispatchers.IO) {
        try {
            var list: MutableList<EarthquakeDomain> = mutableListOf()
            val dataBase = DataBase()
            if (checkFirstOpen) {
                earthquakeReportData?.features?.forEach { feature ->
                    if (feature.properties.place != null) {
                        dataBase.write(
                            mag = feature.properties.mag,
                            place = feature.properties.place,
                            time = feature.properties.time,
                            url = feature.properties.url
                        )

                        list.add(
                            EarthquakeDomain(
                                mag = feature.properties.mag,
                                place = feature.properties.place,
                                time = feature.properties.time,
                                url = feature.properties.url
                            )
                        )
                    }
                }
            } else list = dataBase.read()
            list
        } catch (e: Exception) {
            e.printStackTrace()
```

```
            emptyList()
        }
    }
}
```

В методі map фільтруються дані та використовуються лише ті, які необхідні. Треба записати *mag, place, time, url* в базу даних, за умови якщо *place != null*. Це дає 100-відсоткову гарантію, що дані не порожні.

```
fun getEarthquakeList(
    checkFirstOpen: Boolean,
    block: (List<EarthquakeDomain>) -> Unit
) {
    viewModelScope.launch(Dispatchers.Main) {
        val result = getListOfEarthquakesUseCase.execute(checkFirstOpen)
        block.invoke(result)
    }
}
data class EarthquakeDomain(
    val mag: Float? = 0F,
    val place: String? = "Error",
    val time: Long? = 0,
    val url: String? = "Error"
)
```

В результаті, отримуємо відфільтрований список елементів EarthquakeDomain за допомогою бібліотеки Retrofit 2. Для відображення інформації на екрані мобільного пристрою необхідно реалізувати інтерфейс програми за допомогою розмітки XML.

2.4. Використання сервісу One Signal для автоматичного надсилання сповіщень

Розглянемо використання сервісу One Signal для автоматичного надсилання сповіщень. One Signal – це потужний сервіс для керування сповіщеннями, який дозволяє розробникам легко та ефективно взаємодіяти з користувачами через різні канали. Розробимо кодову реалізацію для автоматизованого надсилання сповіщень з використанням цього сервісу.

Сервіс One Signal пропонує гнучкі налаштування push-сповіщень за допомогою свого API. Проаналізувавши документацію API One Signal, потрібно створити POST-запит на One Signal, котрий міститиме в себе головні для користувача дані, а саме: *місце та магнітуда*. Для того, щоб ефективно працювати з One Signal, в документації рекомендують брати код з бібліотекою Okhttp, але реалізуємо через Retrofit2.

```
@Headers(
   "Content-Type: application/json",
   "Authorization: Basic **********YzZl "
)
@POST("https://api.onesignal.com/notifications")
fun sendNotification(@Body notificationData: OneSignalNotificationData):
Call<ResponseBody>
```

Цей код реалізує POST-запит на сервер One Signal. Він буде передавати параметри, котрі потрібно помістити в клас OneSignalNotificationData. Також в цьому запиті вказано базове посиланняу зв’язку з тим, що базове посилання котре використовувалося раніше, не підходить до API One Signal. Головною умовою роботи цього запиту є хедери, а саме *Content-Type*: application/json та Authorization: Basic **********YzZl. Перший хедер вказує серверу тип контенту, який надсилається разом із запитом. В даному випадку це означає, що в тілі запиту буде передано дані у форматі JSON. А другий хедер вже отримано з налаштованого кабінету в One Signal

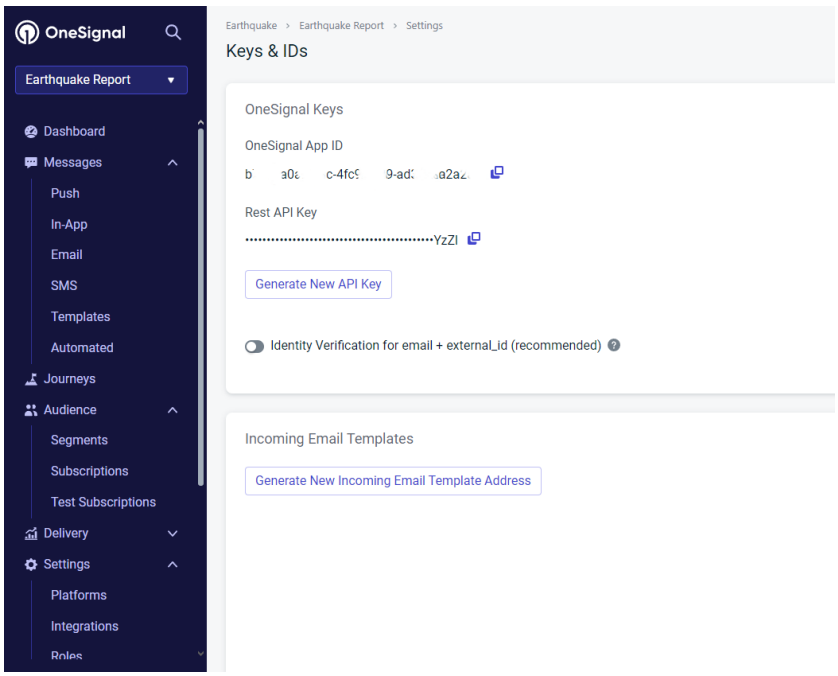

Рис. 2.5. Кабінет розробника в сервісі One Signal

Використовуючи цей кабінет, можна відслідковувати дії користувачів в застосунку: з якої вони країни та скільки разів вони заходили в застосунок, або як довго вони в ньому провели часу. Тому, це не тільки чудовий сервіс для відправки push-сповіщень, а й гарний інструмент для аналізу дій користувачів в застосунку.

Реалізовуючи відправку сповіщень, потрібно розуміти, що вони не мають відправлятися тоді, коли користувач знаходиться в застосунку, адже це сповіщення не матиме жодної цінності, оскільки користувач і так може побачити інформацію на екрані. А отже, якщо push-сповіщення повинно відображатися в той момент, коли користувач не використовує застосунок, необхідно використовувати фонові процеси. В цьому допоможе Work Manager.

Work Manager з'явився як інструмент для відкладеної фонової роботи. Починаючи з першого випуску, він надавав кілька основних API:

1) запуск відкладених завдань дозволяє розвантажувати додаток від завдань, виконання яких не потрібно негайно, наприклад, синхронізація з сервером у певний час доби;

2) обмеження (Constraints) дозволяють запускати завдання лише за певних умов пристрою, наприклад, наявність мережі, режим зарядки або режим сну;

3) можливість реалізації послідовності або ланцюжка завдань, що залежать від успішного виконання попередніх;

4) підтримка як одноразових, так і періодичних завдань.

Крім того, WorkManager надає можливість відстежувати статус виконання завдання: чи знаходиться воно в черзі, виконується, заблоковане або завершене [1].

Для того щоб використовувати Work Manager необхідний *Worker*, але в застосунку реалізуємо *Coroutine Worker*, який підтримує багатопоточність, що дозволить безперешкодно здійснювати запити на сервер One Signal безпосередньо у фоновому режимі. В результаті наслідуючого класу від Coroutine Worker отримано:

```
override suspend fun doWork(): Result {
    return withContext(Dispatchers.IO) {
        val myProcess = RunningAppProcessInfo()
        ActivityManager.getMyMemoryState(myProcess)
        val isInBackground = myProcess.importance !=
RunningAppProcessInfo.IMPORTANCE_FOREGROUND

        try {
            if (isInBackground) {
                getListOfEarthquakesUseCase.execute(true)
                Log.d("DoWork", "success")
            } else throw java.lang.Exception("foreground worker")
            Result.success()
        } catch (e: Exception) {

            Log.d("DoWork", "retry")
            Result.retry()
        }
    }
}
```

Цей код здійснює звичайний запит, але його головна мета полягає в тому, що цей запит повинен працювати лише тоді, коли користувач згорнув застосунок. Таким чином, користувач не побачить інформацію про землетруси на своєму екрані і не буде проінформований про новий землетрус. Якщо ж користувач знаходиться в застосунку, буде використаний метод Result.retry(), який через деякий час знову запустить функцію doWork(). Це буде повторюватися до тих пір, поки користувач не покине застосунок. У такому разі йому буде надіслано сповіщення про новий землетрус, у якому буде вказано місце та сила землетрусу.

Наразі цей код не запускається, тому реалізуємо його періодичну роботу. Відповідно до документації Worker Manager, мінімальна періодичність спрацювання становить 15 хвилин. Отже, ми будемо використовувати цю періодичність для забезпечення максимально точного відображення сповіщення.

Запускається Work Manager за допомогою наступного коду:

```
private fun setTimeWorkRequest() {

    val workManager = WorkManager.getInstance(this)
    val constraints = Constraints.Builder()
       .setRequiredNetworkType(NetworkType.CONNECTED)
       .build()
    val uploadRequest =
       PeriodicWorkRequestBuilder<NotificationWorker>(15,
TimeUnit.MINUTES)
           .addTag("EarthquakeReports")
           .setConstraints(constraints)
           .build()
    workManager.cancelAllWorkByTag("EarthquakeReports")
    workManager.enqueue(uploadRequest)
  }
```

В коді отримується екземпляр Work Manager та відбувається його налаштування, де задаються умови його роботи. Головна умова нашого процесу – це

наявність інтернет підключення і не важливо чи то мережа WI-FI чи мобільна. Адже, саме за допомогою інтернет підключення здійснюватимуться GET/POST запити. Вказується тег EarthquakeReports, для того щоб по цьому тегу закривати всі Worker Manager, які були створені раніше, адже в іншому випадку буде колізія і push-сповіщення можуть надсилатися по декілька разів. При чому, вони будуть абсолютно схожі між собою. В кінці додаємо PeriodicWorkRequestBuilder в Work Manager. Тепер все готово для стабільної роботи push-сповіщень в застосунку.

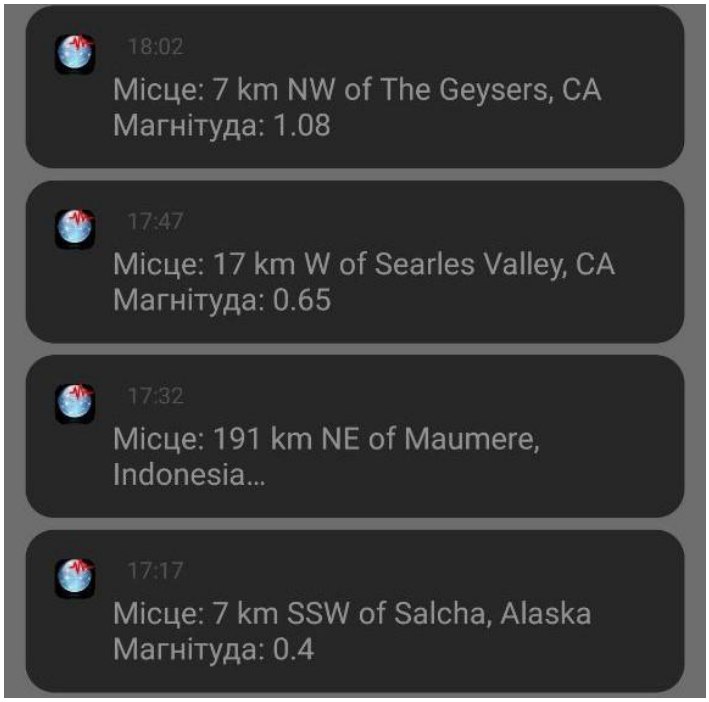

Рис. 2.6. Відображення сповіщень за допомогою сервісу One Signal

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1. Опис функціональності системи

Головна сторінка

На головному екрані застосунку, користувач може переглядати список останніх землетрусів разом з їх магнітудою та датою/часом. Ця інформація дозволяє швидко отримати огляд актуальних подій і зрозуміти їх значення. При натисканні на конкретний землетрус, користувач отримує додаткові деталі, такі як глибина поштовху, точне місцезнаходження та інші важливі параметри. Всі дані про землетруси відображаються через API USGS та реалізовані на базі бібліотеки Retrofit 2. У разі якщо користувач зайшов в застосунок та не має доступу до інтернет він просто побачить ту інформацію, яка була збережена в базі даних. Якщо база даних пуста, то на головному екрані буде виведено повідомлення про помилку. Також в цей момент відбувається підключення до сервісу One Signal, котрий моніторить останні землетруси відправляючи GET запити на сервер USGS і порівнюючи відповідь із вже збереженим у базі даних. Якщо дані між собою мають розбіжності то користувач отримує сповіщення за допомогою POST запиту на API One Signal, але тільки тоді коли він знаходиться поза межами застосунку.

Деталі землетрусу

При натисканні на конкретний землетрус, користувач отримує додаткові деталі, такі як глибина поштовху, точне місцезнаходження та інші важливі параметри. Це дозволяє збільшити розуміння та усвідомлення про те, що відбувається. Реалізація цього відбуватиметься за допомогою вікна WebView.

WebView – це компонент Android, який дозволяє вбудовувати веб-сторінки безпосередньо в додаток. Він надає можливість відображати веб-зміст, такий як HTML-сторінки, веб-додатки, онлайн-ресурси тощо, без переходу до зовнішнього веб-браузера [13, с. 103]. Це дозволяє збільшити розуміння та усвідомлення про те, що відбувається. Реалізація цього відбуватиметься за допомогою вікна WebView. WebView – це компонент Android, який дозволяє вбудовувати веб-сторінки безпосередньо в додаток. Він надає можливість відображати веб-зміст, такий як HTML-сторінки, веб-додатки, онлайн-ресурси тощо, без переходу до зовнішнього веб-браузера [13, с. 103].

Основні характеристики WebView.

1. Відображення веб-змісту: WebView відображає веб-сторінки та веб-додатки безпосередньо в межах додатку. Він зберігає всі можливості браузера, такі як рендеринг HTML, виконання JavaScript, завантаження зображень та інші веб-функції.

2. Взаємодія з веб-сторінкою: користувач може взаємодіяти з вмістом WebView так само, як із звичайною веб-сторінкою. Він може клікати по посиланнях, заповнювати форми, переходити на інші сторінки, і т. д.

3. JavaScript підтримка: WebView може виконувати JavaScript код на веб-сторінках, що дозволяє реалізувати більш складані інтерактивні функції.

4. Налаштування поведінки: можливість налаштовувати WebView, включаючи рівень безпеки, кешування, обробку переадресацій, масштабування сторінок та інші параметри.

5. Обробка подій: можливість встановлювати обробники подій для різних подій в WebView, таких як завантаження сторінки, помилки завантаження, кліки по посиланнях тощо.

WebView є потужним інструментом для інтеграції веб-змісту у застосунки Android. Він дозволяє створювати більш потужні інтерфейси, які поєднують локальний та веб-вміст для покращення користувацького досвіду.

Даний застосунок має декілька базових налаштувань, якими зможе керувати кожен користувач. Далі розглянемо кожен з них.

Зміна кольору тексту

У розділі налаштувань користувач може налаштовувати різні аспекти інтерфейсу застосунку. Користувач зможе вибрати бажаний колір тексту для покращення зручності читання. Для того, щоб здійснити реалізацію в коді була не занадто складною, використаємо бібліотеку ColorPickerView. ColorPickerView від skydoves – це бібліотека для Android, яка надає можливість вбудовувати в додатки розширений візуальний інтерфейс для вибору кольорів. Вона дозволяє користувачам зручно обирати колір за допомогою інтерактивного інтерфейсу, що спрощує вибір кольору для налаштування елементів інтерфейсу.

Основні особливості бібліотеки ColorPickerView.

1) вибір кольору за допомогою палітри: бібліотека надає можливість відображати палітру кольорів, де користувач може вибрати бажаний колір шляхом прокручування або торкання по палітрі;

2) компактний інтерфейс: ColorPickerView має досить компактний інтерфейс, який може бути вбудований у різні екрани інтерфейсу вашого додатку без значних зусиль з розміщення;

3) кастомізація вигляду: бібліотека дозволяє змінювати вигляд палітри кольорів, такий як розмір, колір вибраного кольору, тип розташування (горизонтальний або вертикальний) тощо;

4) підтримка кольорових форматів: ColorPickerView підтримує різні формати представлення кольорів, такі як RGB, ARGB, HEX і т. д., що дозволяє працювати з кольорами у зручному форматі для додатку;

5) реакція на події: бібліотека надає можливість обробки подій вибору кольору.

6) сумісність зі стандартними компонентами: ColorPickerView може легко інтегруватися з іншими стандартними компонентами Android і використовуватися для налаштування кольорів елементів користувацького інтерфейсу.

Зміна шрифту

Можливість вибору різних шрифтів для тексту, щоб відповідати особистим вподобанням. Для комфортної зміни шрифту використаємо Spinner. Перш за все, Spinner є важливим компонентом інтерфейсу користувача, який дозволяє вибирати

один елемент із випадаючого списку. Це дозволяє користувачам зручно вибирати опції або значення зі списку доступних варіантів прямо з інтерфейсу додатка.

Кожен елемент в Spinner може бути представлений текстом, об'єктом або навіть складним вмістом, таким як власний макет, що відображає різні аспекти елемента списку, наприклад, текст та значок.

Після вибору елемента зі списку, вибране значення відображається на Spinner, що дозволяє користувачам бачити обраний варіант та швидко змінювати вибір за потреби.

Зміна кольору верхнього поля

Користувач може налаштувати колір верхнього поля застосунку для створення бажаного вигляду. Також можна використати ColorPickerView.

Управління сповіщеннями

Можливість ввімкнення або вимкнення сповіщень від застосунку про надходження нових даних про землетруси. Це дозволяє користувачеві керувати тим, яка інформація відображатиметься активно.

Ці налаштування розроблені для забезпечення максимальної зручності користувача та гнучкості у використанні додатку.

Рекомендації до/після землетрусу

На додатковому екрані з рекомендаціями користувач отримує практичні поради та інструкції щодо дій після відчуття землетрусу. Такі рекомендації можуть включати:

1) пошук безпечного місця – інформація про те, як знайти безпечне місце у разі землетрусу;

2) перевірка на наявність пошкоджень – поради щодо перевірки наявності пошкоджень у приміщенні;

3) огляд навколишнього середовища – інструкції щодо огляду навколишнього середовища та потенційних небезпек.

Ці рекомендації спрямовані на підвищення безпеки та готовності користувача до можливих наслідків землетрусів.

3.2. Створення макетів та фрагментів екрану

3.2.1. Головна сторінка

Головна сторінка (або головний екран) має за мету візуально інформувати користувача про ключові можливості додатку. Її призначання полягає в забезпеченні зручного та швидкого доступу до основної інформації, функцій чи ресурсів. Для відображення інтерфейсу користувача будемо використовувати фрагменти.

Фрагмент в Android є одним з основних будівельних блоків інтерфейсу, який використовується для побудови гнучких і перевикористовуваних компонентів у додатках. Основна його мета полягає в розділенні користувальницького інтерфейсу на менші та самостійні частини, що можуть бути легко керовані та маніпульовані. Але фрагмент сам по собі не може існувати, тому необхідно використовувати Activity в якому він і буде знаходитися.

Для реалізації потрібно використовувати багато однакових блоків, в яких буде відображатися інформація про землетруси, це мають бути певні списки блоків з інформацією. Найкраще для цього підходить RecyclerView [19].

RecyclerView є чудовим інструментом для відображення списків даних у додатках Android. Він забезпечує ефективне управління пам'яттю та ресурсами пристрою шляхом "лінивого" завантаження елементів списку тільки тоді, коли вони стають видимими на екрані. Основні компоненти RecyclerView включають LayoutManager, Adapter і ViewHolder.

LayoutManager відповідає за організацію та розміщення елементів у списку згідно з певними правилами (наприклад, лінійно, у вигляді сітки або каскадом). Adapter використовується для зв'язку даних джерела з RecyclerView, дозволяючи створювати і переробляти відображення елементів списку відповідно до зміни даних. ViewHolder зберігає посилання на вигляди (Views) кожного елемента списку, що допомагає уникнути зайвих викликів findViewById() та поліпшує продуктивність.

Основні переваги використання RecyclerView включають підтримку анімації змін стану елементів, гнучкість у керуванні макетом списку, підтримку різних типів

елементів у списку (наприклад, заголовків, футерів), а також підтримку дотиків та жестів користувача.

Рис. 3.1. Макет головного екрану

Для створення власного блоку, можна створити нову XML розмітку та вже описати вигляд бажаного блоку з інформацією. Також кожному елементу потрібно присвоювати ідентифікатор android:id="@+id/name", для звернення до певного елемента в адаптері RecyclerView. Отже, створимо XML в якому будуть відображатися всі наші дані, які отримуються з HTTP-запиту.

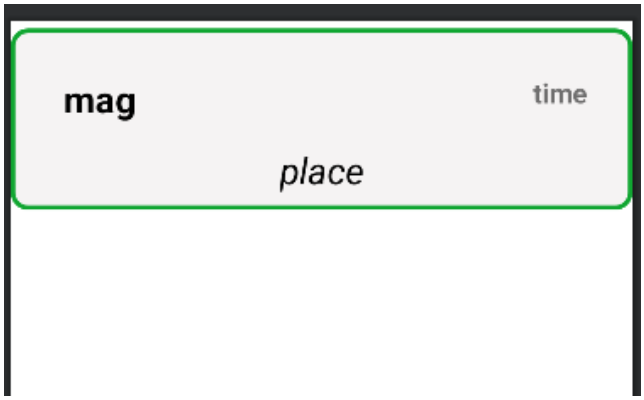

Рис. 3.2. Макет компоненту

Отже, макет компоненту RecyclerView успішно створено, тепер помістимо його в сам RecyclerView за допомогою RecyclerView.Adapter, який слугує певним навігатором, котрий встановлює для кожного поля необхідно інформацію. Реалізація виглядає таким чином:

```
override fun onBindViewHolder(holder: EarthquakeViewHolder, position: Int) {
    val mag = earthquakeList[position].mag
    val time = earthquakeList[position].time
    val sdf = java.text.SimpleDateFormat("yyyy-MM-dd, HH:mm:ss", Locale.UK)
    val date = Date(time!!)
    val fontStyle = earthquakeStorage.getFontStyle()
    val textColor = earthquakeStorage.getTextColor()
    holder.binding.mag.text = String.format("%.1f", mag)
    holder.binding.place.text = earthquakeList[position].place?.replace("?", "a")
    holder.binding.place.setTextColor(Parser.parseColor(textColor))
    Parser.setFontFamily(holder.binding.place, activity, fontStyle)
    holder.binding.time.text = sdf.format(date)
    setBackgroundColor(mag = mag!!, holder = holder)
    clickableItemView(holder = holder, position = position)
}
```

В адаптері для кожного поля *TextView* присвоюється певний текст, зі списку earthquakeList, це саме той список, котрий ми отримали за допомогою @GET запиту. Далі вже відбувається встановлення шрифтів, які збережені в базі даних або стандартні. Також встановлюється спеціальний фон setBackgroundColor для поля holder.binding.mag, в залежності від сили магнітуди, колір фону TextView буде змінюватися на екрані користувача. В кінці встановлюється обробник натискання на об’єкт clickableItemView. В нього подається посилання за допомогою holder, а вже WebView забезпечує відображення сторінки того посилання.

Рис. 3.3. Вигляд головного екрану на мобільному пристрої

3.2.2. Сторінка налаштування

Налаштування дозволяють користувачеві персоналізувати зовнішній вигляд програми відповідно до своїх вподобань.

Компоненти інтерфейсу

- Spinner: цей компонент використовується для вибору однієї опції з списку доступних. У даному випадку Spinner використовується для вибору стилю шрифту тексту.
- ColorPickerView: цей компонент використовується для вибору кольору. У даному випадку ColorPickerView використовується для вибору кольору тексту та кольору верхнього поля.

Функціональні можливості

- Вибір стилю шрифту: користувач може вибрати один із доступних стилів шрифту для тексту.
- Вибір кольору тексту: користувач може вибрати колір тексту за допомогою ColorPickerView.

- Вибір кольору верхнього поля: користувач може вибрати колір верхнього поля за допомогою ColorPickerView.
- Сповіщення: користувач може ввімкнути або вимкнути сповіщення в залежності від його вподобань, натиснувши на поле notifications.

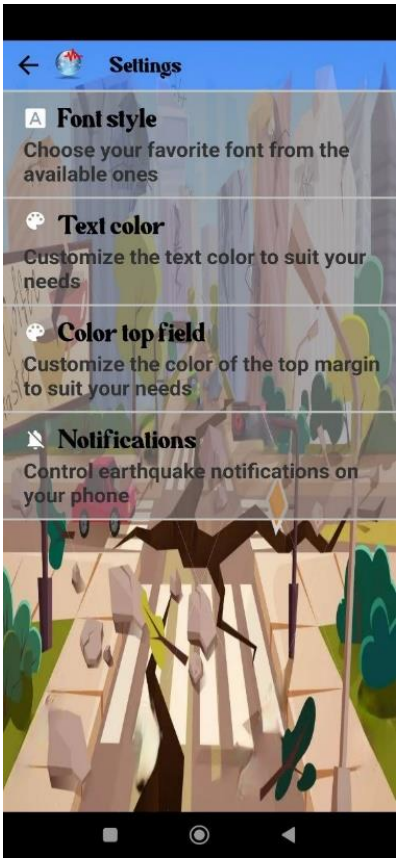

Рис. 3.4. Вигляд налаштувань на мобільному пристрої

Коли користувач активує отримання нотифікацій і дозволяє додатку працювати у фоновому режимі, це передбачає, що застосунок отримує дозвіл на виконання певних завдань поза активним інтерфейсом користувача або навіть у випадку, коли програма перебуває у стані неактивності або екран вимкнено.

Дозвіл на фонову роботу надає додатку змогу виконувати задачі, які є важливими для функціонування або для забезпечення зручності користування. А саме додаток може моніторити певні події або стан системи, щоб відправляти користувачеві повідомлення через механізм сповіщень. Також застосунок потребує регулярного оновлення даних.

3.2.3. Інформаційна сторінка

Інформаційна сторінка, має на меті інформувати населення про правильні дії під час землетрусу. Сторінка містить кілька ключових порад та рекомендацій, які спрямовані на сприяння безпеки в різних ситуаціях.

В разі перебування у приміщенні, рекомендується впасти на землю, укритися під міцними меблями та утримуватися в такому положенні, доки трясіння не припиниться. Якщо особа знаходиться на відкритому просторі, слід негайно переміститися на відкриту ділянку, уникаючи будівель, дерев, вуличних ліхтарів та електричних дротів. У випадку перебування в транспортному засобі, необхідно зупинитися якомога швидше і безпечніше, залишаючись у транспортному засобі до припинення трясіння. Особливо важливими є дії на кухні, де слід негайно вимкнути газ та електрику для запобігання пожежам. Використання ліфтів під час або після землетрусу категорично не рекомендується через ризик застрягання. Після основного землетрусу необхідно бути готовим до повторних поштовхів.

Для отримання додаткової та більш детальної інформації користувачі можуть скористатися кнопкою *“More Tips”*. Ця кнопка перенаправляє на офіційну сторінку Геологічної служби США (USGS) за допомогою елемента WebView, де розміщено детальні інструкції та рекомендації щодо дій у випадку землетрусу. Використання інтерактивної кнопки забезпечує доступ до всебічної інформації та сприяє підвищенню рівня обізнаності населення про заходи безпеки.

Рис. 3.5. Рекомендації до/після виникнення землетрусу

Включення інформаційного вікна з довідкою щодо безпеки під час землетрусу в мобільний застосунок має важливе значення для користувачів у разі надзвичайних ситуацій. Це вікно виконує кілька ключових функцій:

Підвищення обізнаності

Надає користувачам важливу інформацію про те, як діяти під час землетрусу, що сприяє підвищенню загального рівня обізнаності про безпеку.

Оперативність отримання інформації

Забезпечує швидкий доступ до рекомендацій у реальному часі, що особливо важливо під час землетрусу, коли кожна секунда може бути вирішальною для збереження життя та здоров'я.

Мінімізація ризиків

Надає конкретні поради щодо дій у різних ситуаціях, таких як перебування в приміщенні, на відкритому просторі, в транспортному засобі чи на кухні.

Підготовка до наступних поштовхів

Інформує про необхідність бути готовими до повторних поштовхів після основного землетрусу, що є важливим аспектом безпеки.

Інтерактивність та доступність

Кнопка “More Tips” дозволяє користувачам легко переходити на офіційні джерела інформації, такі як сторінка Геологічної служби США (USGS), де вони можуть отримати більш детальні та розширені рекомендації.

Психологічна підтримка

Надає чітких і зрозумілих інструкцій може зменшити паніку та страх під час надзвичайної ситуації, допомагаючи користувачам залишатися спокійними та діяти раціонально.

Таким чином, включення інформаційного вікна з довідкою щодо безпеки під час землетрусу в мобільний застосунок не лише сприяє безпеці користувачів, але й підвищує соціальну відповідальність розробників, зміцнює довіру до застосунку та виконує важливу освітню та підготовчу функції.

3.3. Використання API у реальних умовах

В застосунку використано два API, такі як USGS та One Signal. Перед публікацією додатку в Play Market, з метою забезпечення можливості вільного завантаження та використання додатку іншими користувачами на своїх пристроях, необхідно ретельно відпрацювати всі критично важливі функціональні ланцюги [22].

Усе, що бачить користувач, – це лише інтерфейс програми. Але як насправді відбувається її робота? Щоб інформація відобразилася на екрані, спершу необхідно надіслати команду на сервер USGS або отримати дані з локальної бази даних у разі збою або відсутності доступу до мобільної мережі. Коли API USGS повертає всю інформацію у вигляді JSON, у програмі відбувається маппінг даних. Необхідні дані зберігаються у базу даних застосунку, а інші відсіюються. Важливо пам'ятати, що ці дані знаходяться на закритому рівні застосунку data, і рівень presentation не має

прямого доступу до них. Тому бібліотека Hilt забезпечує передачу даних з рівня data до рівня presentation.

Після успішного отримання всіх даних та підготовки до відображення виникає необхідність створення відповідних полів для їх представлення. Це досягається за допомогою XML розмітки в середовищі розробки Android Studio, де створюються вікна, які міститимуть в собі необхідну інформацію. У цих вікнах можуть бути розміщені тексти, кнопки, списки, перемикачі та інші елементи. Одна з цілей розробників полягає у створенні інтерфейсу, який був би зрозумілим та естетично приємним для користувача, оскільки це великою мірою впливає на задоволеність його потреб.

Після підготовки вікна для відображення інформації у розробленому застосунку, необхідно отримати посилання на відповідні поля за допомогою методу findViewById(id). Після цього можна присвоювати текст або встановлювати певні дії, наприклад, під час натискання кнопки або переходу на інше вікно. Це дозволяє отримати та відобразити інформацію, отриману з API USGS, у вікні застосунку.

З API One Signal все значно по іншому. В застосунку воно відповідає за відображення певних сповіщень. Для його реалізації потрібно виконувати запити на сервер, аналогічно до отримання інформації з API USGS, та порівнювати отриману інформацію зі збереженою у базі даних. Це дозволяє вчасно реагувати на нову інформацію із сайту USGS [10]. Але цього буде недостатньо, адже якщо закрити застосунок повністю, він просто припинить свою роботу і в разі виникнення нового землетрусу, користувач не дізнається про це. Отже, тоді застосунок втратить певну свої цінність, бо як такого сповіщення не буде у разі повного припинення роботи застосунку. Для забезпечення стабільної роботи застосунку у фоновому режимі можна використовувати Service та CoroutineWorker.

Service в Android — це компонент, який виконує тривалі операції у фоновому режимі без взаємодії з користувачем. Він підходить для завдань, які потребують постійної роботи або тривалого виконання, навіть коли застосунок неактивний. Існують три основні типи Service: Foreground Service, що виконує помітні для користувача завдання (наприклад, відтворення музики) і потребує відображення

постійного повідомлення у рядку стану. Background Service, що працює у фоновому режимі без взаємодії з користувачем, але має обмеження на використання в Android 8.0 і новіших версіях та Bound Service, що пропонує клієнт-серверний інтерфейс для взаємодії з іншими компонентами застосунку. Використання Service дозволяє виконувати тривалі операції незалежно від стану активності (Activity), що запобігає зупинці завдань при переході між екранами.

CoroutineWorker — це компонент бібліотеки WorkManager, який дозволяє виконувати фонові завдання за допомогою корутин. CoroutineWorker підтримує всі можливості WorkManager, включаючи умови виконання завдань (наявність мережі, рівень заряду батареї тощо) та надійність, зокрема можливість продовження завдань після перезапуску пристрою. Завдяки корутинам, написання коду для фонових задач стає більш простим і зрозумілим, покращуючи стійкість та підтримуваність коду. Використання CoroutineWorker забезпечує асинхронність та ефективність виконання фонових завдань з меншими зусиллями [2].

Таким чином, поєднавши Service та CoroutineWorker, можна отримати стабільну роботу у фоновому режимі із певним інтервалом спрацювання, що є дуже важливим, адже необхідно постійно оновлювати інформацію про поштовхи. В результаті роботи CoroutineWorker здійснюється оновлення інформації в бази даних та якщо остання інформація в базі даних про землетруси відрізняється з найновішою отриманою інформацією з серверу то в результаті користувач отримає сповіщення, в якому і буде відображено інформацію про новий землетрус.

Для того, щоб переконатися у коректній роботі застосунку, необхідно переконатися у правильності отримання інформації. Якщо розбіжностей в застосунку і на сайті USGS немає, то все працює належним чином.

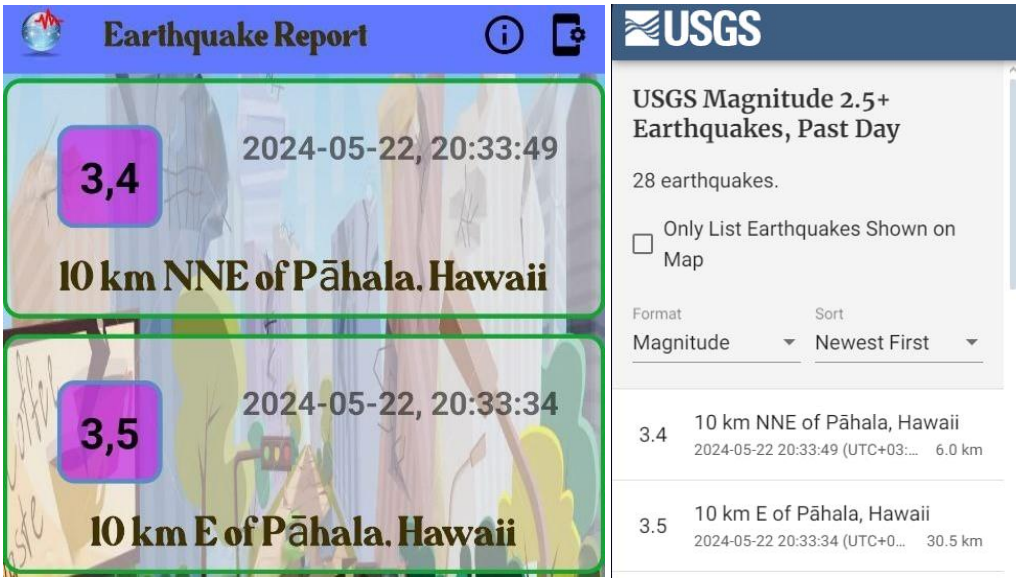

Рис. 3.6. Інформація в застосунку та на сайті USGS

На рис. 3.6. наведено порівняння інформації про землетруси, відображеної в мобільному застосунку та на офіційному сайті Геологічної служби США (USGS). Обидва джерела демонструють ідентичні дані про дві сейсмічні події, що відбулися поблизу Пахала, Гаваї.

Застосунок відображає наступну інформацію, що є ідентичною до даних офіційного сайту USGS:

1) магнітуда 3,4 означає, що землетрус стався 10 км на північ-північний схід від Пахала, Гаваї, з часом події 2024-05-22, 20:33:49;

2) магнітуда 3,5 надає інформацію про те, що землетрус стався 10 км на схід від Пахала, Гаваї, з часом події 2024-05-22, 20:33:34.

Відповідність даних між мобільним застосунком та сайтом USGS свідчить про правильну реалізацію логіки отримання та обробки інформації у застосунку. Це підтверджує, що застосунок коректно використовує API USGS для завантаження даних про землетруси в режимі реального часу, забезпечуючи точність та актуальність відображуваної інформації.

У технічному аспекті, правильна реалізація включає:

1) завантаження даних через API – застосунок успішно виконує запити до API USGS, отримуючи необхідну інформацію про землетруси;

2) парсинг і обробка даних – отримана інформація правильно обробляється та парсується, зберігаючи точність часу події, магнітуди та інших параметрів;

3) відображення даних – інформація коректно відображається у зрозумілому для користувача форматі, що забезпечує високу інформативність і легкість сприйняття.

Таким чином, рис. 3.6. демонструє успішну інтеграцію та синхронізацію мобільного застосунку з офіційними даними USGS, підтверджуючи правильність реалізації та надійність використовуваних алгоритмів.

ВИСНОВКИ

У кваліфікаційній роботі розроблено мобільний застосунок на основі мови програмування Kotlin, спрямований на автоматизацію та оптимізацію процесу надсилання сповіщень про землетруси. Дослідження показало, що сучасні системи сповіщення мають певні обмеження щодо часу реагування та ефективності передачі інформації. Запропонований застосунок вирішує ці проблеми, забезпечуючи своєчасне інформування користувачів про надзвичайні ситуації.

Об’єктом дослідження виступав мобільний застосунок для автоматичного надсилання сповіщень про землетруси, а предметом — процес його розробки з використанням мови програмування Kotlin. У ході роботи розглянуто технічні аспекти розробки програмного забезпечення, включаючи інтеграцію з API USGS для отримання даних про землетруси та використання технологій, таких як Service та CoroutineWorker, для забезпечення надійного фонової роботи застосунку.

Однією з важливих особливостей проекту є використання Clean Architecture, яка сприяє створенню масштабованого, легко підтримуваного та тестованого коду. Clean Architecture дозволяє чітко розділити логіку застосунку на рівні презентації, домену та даних, забезпечуючи незалежність кожного рівня та полегшуючи внесення змін у майбутньому.

Також важливою технологією, використаною в проекті, є Hilt DI**,** що забезпечує ефективне та зручне управління залежностями у застосунку. Використання Hilt спрощує процес впровадження залежностей, покращує тестованість коду та зменшує кількість шаблонного коду, необхідного для налаштування залежностей.

Результати роботи свідчать про успішну реалізацію поставлених завдань та досягнення цілей проекту. Розроблений мобільний застосунок демонструє високу ефективність у надсиланні сповіщень про землетруси, що підтверджується відповідністю даних між застосунком і офіційним сайтом USGS. Таким чином, даний проєкт вносить значний вклад у розвиток систем сповіщення про надзвичайні ситуації та підвищує рівень готовності суспільства до реагування на природні загрози.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Android Architecture Component. URL: https://developer.android.com/topic/architecture (дата звернення 08.05.2024 р). – Назва з екрана.
2. Android CoroutineWorker. URL: https://developer.android.com/develop (дата звернення 09.05.2024 р). – Назва з екрана.
3. Android Developers. URL: https://developer.android.com/studio/debug/dev-options (дата звернення 09.05.2024 р). – Назва з екрана.
4. Android Service. URL: https://developer.android.com/develop/background-work/services (дата звернення 10.05.2024 р). – Назва з екрана.
5. Axelrod A. Complete Guide to Test Automation Techniques, Practices, and Patterns for Building and Maintaining Effective Software Projects / A. Axelrod. – USA, 2018. – 542 s.
6. Clean Architecture. URL: https://medium.com/codex/introduction-to-clean-architecture-2437c6987ec (дата звернення 09.05.2024 р). – Назва з екрана.
7. Hilt DI (Dependency Injection). URL: https://developer.android.com/training/dependency-injection/hilt-android (дата звернення 13.05.2024 р). – Назва з екрана.
8. Kotlin Programming Language. URL: https://kotlinlang.org/ (дата звернення 14.05.2024 р). – Назва з екрана.
9. OpenQuake : Open-source Earthquake Hazard and Risk Software. URL: https://www.globalquakemodel.org/product/openquake-engine (дата звернення 16.05.2024 р). – Назва з екрана.
10. USGS Earthquake Hazards Program. URL: https://www.usgs.gov/programs/earthquake-hazards (дата звернення 15.05.2024 р). – Назва з екрана.

11. Абрамов, А. А., Бабушкин, А. В., Беляев, А. С. – Розробка мобільного додатку для оповіщення про землетруси // Вісник Національного технічного університету «Харківський політехнічний інститут», 2021. – 112-119 с.
12. Інструменти і середовища розробки мобільних додатків. URL: https://ppt-online.org/341525 (дата звернення 17.05.2024 р). – Назва з екрана.
13. Копитко М. Ф. Основи програмування мовою Java / М. Ф. Копитко, К. С. Іванків; Нац. ун-т «ЛНУ ім. Івана Франка». − Львів: Вид-во Нац. ун-ту «ЛНУ ім. Івана Франка», 2016. − 83 с.
14. Ларсон, М. Чиста архітектура: розробка програмного забезпечення, незалежного від змін / М. Ларсон; – Київ: Експрес, 2016. – 320 с.
15. Основи програмування на Java. URL: https://promoter.net.ua/articles/osnovi-programuvannya-na-java.html (дата звернення 13.05.2024 р). – Назва з екрана.
16. Рейтинг мов програмування 2023. URL: https://dou.ua/lenta/articles/language-rating-2023 (дата звернення 10.05.2024 р). – Назва з екрана.
17. Розробка мобільних додатків. URL: https://webcase.com.ua/uk/razrabotka-mobilnyh-prilozhenij (дата звернення 12.05.2024 р). – Назва з екрана.
18. Розробка мобільних додатків від А до Я: повний гайд. URL: https://dan-it.com.ua/uk/blog/rozrobka-mobilnih-dodatkiv-vid-a-do-japovnij-gajd/ (дата звернення 11.05.2024 р). – Назва з екрана.
19. Типи мобільних додатків. URL: https://smileukraine.com/ua/mobile-apps/mobile-apps-types (дата звернення 17.05.2024 р). – Назва з екрана.

ДОДАТКИ

Додаток А

Рівень Presentation

```
@HiltAndroidApp
class App : Application() {
    @Inject
    lateinit var getListOfEarthquakesUseCase: GetListOfEarthquakesUseCase
    companion object{
        lateinit var newGetListOfEarthquakesUseCase: GetListOfEarthquakesUseCase
    }
    override fun onCreate() {
        super.onCreate()
        Hawk.init(this).build()
        OneSignal.initWithContext(this, ONESIGNAL_APP_ID)
        newGetListOfEarthquakesUseCase = getListOfEarthquakesUseCase
        CoroutineScope(Dispatchers.IO).launch {
            OneSignal.Notifications.requestPermission(true)
        }
        startService(Intent(this, NotificationBackgroundService::class.java))
    }
}
interface Clickable {
    fun clickableItemView(holder: EarthquakeAdapter.EarthquakeViewHolder, position: Int)
}
interface Settable {
    fun setBackgroundColor(mag: Float, holder:
EarthquakeAdapter.EarthquakeViewHolder)
```

```
}
@Module
@InstallIn(SingletonComponent::class)
class DataModule {
    @Provides
    @Singleton
    fun provideCloudDataSource(): CloudDataSource = CloudDataSource.Base()
    @Provides
    @Singleton
    fun provideToEarthquakeMapper(): ToEarthquakeDomainMapper =
ToEarthquakeDomainMapper.Base()
    @Provides
    @Singleton
    fun provideRepository(
        cloudDataSource: CloudDataSource,
        toEarthquakeDomainMapper: ToEarthquakeDomainMapper
    ): Repository = BaseRepository(
        cloudDataSource = cloudDataSource,
        toEarthquakeDomainMapper = toEarthquakeDomainMapper
    )
}
@Module
@InstallIn(SingletonComponent::class)
class DomainModule {
    @Provides
    @Singleton
    fun provideGetListOfEarthquakeUseCase(repository: Repository):
GetListOfEarthquakesUseCase =
```

```
GetListOfEarthquakesUseCase.Base(repository = repository)
}
@EntryPoint
@InstallIn(SingletonComponent::class)
interface InitializerEntryPoint {
    fun inject(initializer: WorkManagerInitializer)
    companion object {
        fun resolve(context: Context): InitializerEntryPoint {
            val appContext = context.applicationContext ?: throw IllegalStateException()
            return EntryPointAccessors.fromApplication(
                appContext,
                InitializerEntryPoint::class.java
            )
        }
    }
}
class DependencyGraphInitializer : Initializer<Unit> {
    override fun create(context: Context) {
        InitializerEntryPoint.resolve(context)
        return
    }
    override fun dependencies(): List<Class<out Initializer<*>>> {
        return emptyList()
    }
}
class NotificationBackgroundService: android.app.Service() {
    override fun onBind(intent: Intent?) = null
    override fun onCreate() {
```

```
        super.onCreate()
        setTimeWorkRequest()
    }
@SuppressLint("SuspiciousIndentation")
    private fun setTimeWorkRequest() {
val workManager = WorkManager.getInstance(this)
        val constraints = Constraints.Builder()
            .setRequiredNetworkType(NetworkType.CONNECTED)
            .build()
        val uploadRequest =
            PeriodicWorkRequestBuilder<NotificationWorker>(15, TimeUnit.MINUTES)
                .addTag("EarthquakeReports")
                .setConstraints(constraints)
                .build()
        workManager.cancelAllWorkByTag("EarthquakeReports")
        workManager.enqueue(uploadRequest)
    }
}
@HiltWorker
class NotificationWorker @AssistedInject constructor(
    @Assisted private val context: Context,
    @Assisted params: WorkerParameters
) : CoroutineWorker(context, params) {
    private val earthquakeStorage by lazy {
        EarthquakeStorage(context)
    }
    override suspend fun getForegroundInfo(): ForegroundInfo {
        return super.getForegroundInfo()
```

```
}
    override suspend fun doWork(): Result {
        return withContext(Dispatchers.IO) {
            val myProcess = RunningAppProcessInfo()
            ActivityManager.getMyMemoryState(myProcess)
            val isInBackground = myProcess.importance !=
RunningAppProcessInfo.IMPORTANCE_FOREGROUND
            try {
                if (isInBackground) {
                    val notificationsIsEnable = earthquakeStorage.getNotifications()
                    if (notificationsIsEnable)
                        App.newGetListOfEarthquakesUseCase.execute(true)
                    Log.d("DoWork", "success")
                } else throw java.lang.Exception("foreground worker")
                Result.success()
            } catch (e: Exception) {
                Log.d("DoWork", "retry ${e.message}")
                Result.retry()
            }
        }
    }
}
class WorkManagerInitializer : Initializer<WorkManager>, Configuration.Provider {
    @Inject lateinit var hiltWorkerFactory: HiltWorkerFactory
    override fun create(context: Context): WorkManager {
        InitializerEntryPoint.resolve(context).inject(this)
        WorkManager.initialize(context, workManagerConfiguration)
        return WorkManager.getInstance(context)
```

```
 }
    override fun dependencies(): List<Class<out Initializer<*>>> {
        return listOf(DependencyGraphInitializer::class.java)
 }
    override fun getWorkManagerConfiguration() = Configuration.Builder()
        .setMinimumLoggingLevel(Log.INFO)
.setWorkerFactory(hiltWorkerFactory)
        .build()
}
class EarthquakeAdapter(
    private val earthquakeList: List<EarthquakeDomain>,
    private val earthquakeStorage: EarthquakeStorage,
    private val webViewBuilder: WebViewBuilder
) :
    RecyclerView.Adapter<EarthquakeAdapter.EarthquakeViewHolder>(), Settable,
Clickable {
    class EarthquakeViewHolder(val binding: EarthquakeItemBinding) :
        RecyclerView.ViewHolder(binding.root)
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
EarthquakeViewHolder {
        val itemBinding =
            EarthquakeItemBinding.inflate(LayoutInflater.from(parent.context), parent, false)
        return EarthquakeViewHolder(itemBinding)
    }
    @SuppressLint("NewApi", "DefaultLocale")
    override fun onBindViewHolder(holder: EarthquakeViewHolder, position: Int) {
        val mag = earthquakeList[position].mag
        val time = earthquakeList[position].time
```

```
val sdf = java.text.SimpleDateFormat("yyyy-MM-dd, HH:mm:ss", Locale.ENGLISH)
        val date = Date(time!!)
        holder.binding.mag.text = String.format("%.1f", mag)
 holder.binding.place.text = earthquakeList[position].place?.replace("?", "a")
        UISetter.setFontStyle(earthquakeStorage, holder.binding.place)
        UISetter.setTextColor(earthquakeStorage, holder.binding.place)
        holder.binding.time.text = sdf.format(date)
 setBackgroundColor(mag = mag!!, holder = holder)
        clickableItemView(holder = holder, position = position)
    }
    override fun getItemCount(): Int {
        return earthquakeList.size
    }
    override fun setBackgroundColor(mag: Float, holder: EarthquakeViewHolder) {
        if (mag >= 0f && mag < 2f) {
            holder.binding.mag.setBackgroundResource(R.drawable.round_silver)
        } else if (mag > 2f && mag <= 3f) {
            holder.binding.mag.setBackgroundResource(R.drawable.round_yellow)
        } else if (mag > 3f && mag <= 5f) {
            holder.binding.mag.setBackgroundResource(R.drawable.round_magenta)
        } else if (mag > 5f) {
            holder.binding.mag.setBackgroundResource(R.drawable.round_red)
        }
    }
    override fun clickableItemView(holder: EarthquakeViewHolder, position: Int) {
        holder.itemView.setOnClickListener {
            webViewBuilder.createNewWebView(earthquakeList[position].url.toString())
            webViewBuilder.customBackPressedListener()
```

```
        }
    }
}
class EarthquakeInfoFragment : Fragment() {
    private lateinit var tvInfo: TextView
    private lateinit var tvHeader: TextView
    private lateinit var ivGoBack: ImageView
  private lateinit var btnMoreTips: Button
    private lateinit var earthquakeStorage: EarthquakeStorage
    private lateinit var webViewBuilder: WebViewBuilder
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_earthquake_info, container, false)
    }
    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        earthquakeStorage = EarthquakeStorage(requireContext())
        tvInfo = view.findViewById(R.id.text_info)
        tvHeader = view.findViewById(R.id.earthquakeTipsHeader)
        ivGoBack = view.findViewById(R.id.go_back)
        btnMoreTips = view.findViewById(R.id.moreTipsButton)
        UISetter.setTopColor(earthquakeStorage, tvInfo.parent as ViewGroup)
        UISetter.setFontStyle(earthquakeStorage, tvHeader, btnMoreTips, tvInfo)
        UISetter.setTextColor(earthquakeStorage, tvInfo, tvHeader)
        ivGoBack.setOnClickListener {
```

```
            findNavController().popBackStack()
        }
webViewBuilder = WebViewBuilder(this, tvHeader.parent.parent.parent as ViewGroup)
        btnMoreTips.setOnClickListener {
            webViewBuilder.createNewWebView("https://www.usgs.gov/faqs/what-should-i-
do-during-
earthquake#:~:text=Get%20under%20a%20desk%20or,things%20can%20fall%20on%20y
ou).")
            webViewBuilder.customBackPressedListener()
        }
        requireActivity().onBackPressedDispatcher.addCallback(object:
OnBackPressedCallback(true){
            override fun handleOnBackPressed() {
                findNavController().popBackStack()
            }
        })
    }
}
@AndroidEntryPoint
class EarthquakeMainFragment : Fragment() {
    private val viewModel: MainViewModel by viewModels()
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View = inflater.inflate(R.layout.fragment_earthquake_main, container, false)
    private lateinit var linearLayout: LinearLayout
    private lateinit var appTextView: TextView
    private lateinit var errorTextView: TextView
```

```
private lateinit var earthquakeStorage: EarthquakeStorage
    private lateinit var webViewBuilder: WebViewBuilder
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        val recView = view.findViewById<RecyclerView>(R.id.rec_view)
        appTextView = view.findViewById(R.id.tv_app_name)
        linearLayout = view.findViewById(R.id.linearLayout)
        errorTextView = view.findViewById(R.id.text_error)
  recView.layoutManager = LinearLayoutManager(requireContext())
        val swipe = view.findViewById<SwipeRefreshLayout>(R.id.swipe)
        earthquakeStorage = EarthquakeStorage(requireActivity())
        val switchToSettings = view.findViewById<ImageView>(R.id.arrow_go_setting)
        val switchToInfo = view.findViewById<ImageView>(R.id.arrow_go_info)
        webViewBuilder = WebViewBuilder(this, linearLayout.parent as ConstraintLayout)
        Hawk.init(requireContext()).build()
        switchToSettings.setOnClickListener {
            findNavController().navigate(R.id.action_mainInfoFragment_to_settingsFragment)
        }
        switchToInfo.setOnClickListener {
findNavController().navigate(R.id.action_mainInfoFragment_to_earthquakeInfoFragment
2)
        }
        setRecycler(recView)
        swipe.setOnRefreshListener {
            setRecycler(recView)
            swipe.isRefreshing = false
        }
        requireActivity().onBackPressedDispatcher.addCallback(object:
OnBackPressedCallback(true){
```

```
override fun handleOnBackPressed() {
            requireActivity().finish()
}
        })
    }
    override fun onStart() {
        super.onStart()
  setTextColor(earthquakeStorage, errorTextView, appTextView)
        setFontStyle(earthquakeStorage, errorTextView, appTextView)
        setTopColor(earthquakeStorage, linearLayout)
    }
    private fun setRecycler(recyclerView: RecyclerView) {
        viewModel.getEarthquakeList { mutableList ->
            if (mutableList.isNotEmpty()) {
                val adapter = EarthquakeAdapter(mutableList, earthquakeStorage,
webViewBuilder)
                recyclerView.adapter = adapter
                errorTextView.visibility = View.GONE
                return@getEarthquakeList
            }
            errorTextView.visibility = View.VISIBLE
        }
    }
}
class EarthquakeStorage(context: Context) {
    private val key = "SHARED_PREFS_KEY"
    private val sharedPreferences = context.getSharedPreferences(key, 0)
    private val keyFontStyle = "font_style"
```

```
  private val keyTopFieldColor = "top_field_color"
    private val keyTextColor = "text_color"
private val keyPermission = "permission"
    private val keyNotifications = "notifications"
    fun getPermission() = sharedPreferences.getBoolean(keyPermission, false)
    fun savePermission(answer: Boolean) =
        sharedPreferences.edit().putBoolean(keyPermission, answer).apply()
    fun getTopFieldColor() = sharedPreferences.getInt(keyTopFieldColor, Color.WHITE)
  fun getFontStyle() = sharedPreferences.getInt(keyFontStyle, R.font.adventuro)
    fun getTextColor() = sharedPreferences.getInt(keyTextColor, Color.BLACK)
    fun saveTopFieldColor(value: Int) {
        sharedPreferences.edit().putInt(keyTopFieldColor, value).apply()
    }
    fun saveFontStyle(value: Int) {
        sharedPreferences.edit().putInt(keyFontStyle, value).apply()
    }
    fun saveTextColor(value: Int) {
        sharedPreferences.edit().putInt(keyTextColor, value).apply()
    }
    fun saveNotifications(boolean: Boolean) {
        sharedPreferences.edit().putBoolean(keyNotifications, boolean).apply()
    }
    fun getNotifications(): Boolean {
        return sharedPreferences.getBoolean(keyNotifications, false)
    }
}
@AndroidEntryPoint
class MainActivity : AppCompatActivity() {
```

```
 private lateinit var binding: ActivityMainBinding
    override fun onCreate(savedInstanceState: Bundle?) {
super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)
        window.addFlags(1024)
    }
}
@HiltViewModel
class MainViewModel @Inject constructor(
    private val getListOfEarthquakesUseCase: GetListOfEarthquakesUseCase
) : ViewModel() {
   fun getEarthquakeList(
       block: (List<EarthquakeDomain>) -> Unit
    ) {
        viewModelScope.launch(Dispatchers.Main) {
            val result = getListOfEarthquakesUseCase.execute(false)
            block.invoke(result)
        }
    }
}
class SettingsFragment : Fragment() {
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
```

```
        return inflater.inflate(com.earthquakereport.R.layout.fragment_settings, container, false)
    }
    private lateinit var spinnerFontFamily: Spinner
    private lateinit var textViewFontFamily: TextView
    private lateinit var textViewFieldTopColor: TextView
    private lateinit var textViewColorText: TextView
    private lateinit var earthquakeStorage: EarthquakeStorage
    private lateinit var linearLayout: LinearLayout
    private lateinit var textSettingsView: TextView
    private lateinit var imageViewGoBack: ImageView
    private lateinit var tvNotification: TextView
    private lateinit var ivNotifications: ImageView
    private val fonts = mutableListOf(
        com.earthquakereport.R.font.adventuro,
        com.earthquakereport.R.font.beardsons,
        com.earthquakereport.R.font.milky_boba,
        com.earthquakereport.R.font.reach_story
    )
    private val POWERMANAGER_INTENTS = arrayOf(
        Intent().setComponent(
            ComponentName(
                "com.miui.securitycenter",
                "com.miui.permcenter.autostart.AutoStartManagementActivity"
            )
        ),
        Intent().setComponent(
            ComponentName(
```

```
  "com.letv.android.letvsafe",
        "com.letv.android.letvsafe.AutobootManageActivity"
    )
  ),
Intent().setComponent(
    ComponentName(
        "com.huawei.systemmanager",
        "com.huawei.systemmanager.startupmgr.ui.StartupNormalAppListActivity"
    )
),
Intent().setComponent(
    ComponentName(
        "com.huawei.systemmanager",
        "com.huawei.systemmanager.optimize.process.ProtectActivity"
    )
),
Intent().setComponent(
    ComponentName(
        "com.huawei.systemmanager",
        "com.huawei.systemmanager.appcontrol.activity.StartupAppControlActivity"
    )
),
Intent().setComponent(
    ComponentName(
        "com.coloros.safecenter",
        "com.coloros.safecenter.permission.startup.StartupAppListActivity"
    )
),
```

```
  Intent().setComponent(
    ComponentName(
        "com.coloros.safecenter",
"com.coloros.safecenter.startupapp.StartupAppListActivity"
    )
 ),
 Intent().setComponent(
    ComponentName(
        "com.oppo.safe",
        "com.oppo.safe.permission.startup.StartupAppListActivity"
    )
 ),
 Intent().setComponent(
    ComponentName(
        "com.iqoo.secure",
        "com.iqoo.secure.ui.phoneoptimize.AddWhiteListActivity"
    )
 ),
 Intent().setComponent(
    ComponentName(
        "com.iqoo.secure",
        "com.iqoo.secure.ui.phoneoptimize.BgStartUpManager"
    )
 ),
 Intent().setComponent(
    ComponentName(
        "com.vivo.permissionmanager",
        "com.vivo.permissionmanager.activity.BgStartUpManagerActivity"
```

```
  )
),
Intent().setComponent(
  ComponentName(
        "com.samsung.android.lool",
        "com.samsung.android.sm.battery.ui.BatteryActivity"
    )
),
Intent().setComponent(
    ComponentName(
        "com.samsung.android.lool",
        "com.samsung.android.sm.ui.battery.BatteryActivity"
    )
),
Intent().setComponent(
    ComponentName(
        "com.htc.pitroad",
        "com.htc.pitroad.landingpage.activity.LandingPageActivity"
    )
),
Intent().setComponent(
    ComponentName(
        "com.asus.mobilemanager",
        "com.asus.mobilemanager.MainActivity"
    )
),
Intent().setComponent(
    ComponentName(
```

```
 "com.transsion.phonemanager",
            "com.itel.autobootmanager.activity.AutoBootMgrActivity"
        )
       )
   )
   override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
      view.apply {
         earthquakeStorage = EarthquakeStorage(requireActivity())
         spinnerFontFamily = findViewById(com.earthquakereport.R.id.fontStyleSpinner)
         textViewFontFamily                                                    =
findViewById(com.earthquakereport.R.id.textView_font_family)
         textViewFieldTopColor                                                 =
findViewById(com.earthquakereport.R.id.colorTopTextView)
         textViewColorText = findViewById(com.earthquakereport.R.id.textView_Color)
         linearLayout = findViewById(com.earthquakereport.R.id.linearLayout)
         textSettingsView = findViewById(com.earthquakereport.R.id.text_settings)
         imageViewGoBack = findViewById(com.earthquakereport.R.id.go_back)
         tvNotification = findViewById(com.earthquakereport.R.id.tv_notifications)
         ivNotifications = findViewById(com.earthquakereport.R.id.iv_notifications)
      }
      spinnerFontFamily.visibility = View.GONE
      imageViewGoBack.setOnClickListener {
         findNavController().popBackStack()
      }
      (textViewColorText.parent.parent as LinearLayout).setOnClickListener {
         colorChooser(false)
      }
      (textViewFieldTopColor.parent.parent as LinearLayout).setOnClickListener {
```

```
            colorChooser(true)
        }
    (textViewFontFamily.parent.parent as LinearLayout).setOnClickListener {
            spinnerFontFamily.visibility =
                if (spinnerFontFamily.visibility == View.GONE) View.VISIBLE else
    View.GONE
            if (spinnerFontFamily.visibility == View.VISIBLE)
                spinnerFontFamily.performClick()
        }
        val adapter: ArrayAdapter<*> = ArrayAdapter.createFromResource(
            requireContext(), com.earthquakereport.R.array.fontNames,
            com.earthquakereport.R.layout.spinner_list
        )
        adapter.setDropDownViewResource(com.earthquakereport.R.layout.spinner_list)
        spinnerFontFamily.adapter = adapter
        spinnerFontFamily.onItemSelectedListener = object :
    AdapterView.OnItemSelectedListener {
            override fun onItemSelected(
                parent: AdapterView<*>?,
                view: View?,
                position: Int,
                id: Long
            ) {
                earthquakeStorage.saveFontStyle(fonts[position])
                setFontStyle(
                    earthquakeStorage,
                    textViewColorText,
                    textViewFontFamily,
```

```
        textViewFieldTopColor,
            textSettingsView,
        tvNotification
           )
           spinnerFontFamily.visibility = View.GONE
        }
        override fun onNothingSelected(parent: AdapterView<*>?) {
        }
    }
ivNotifications.setBackgroundResource(com.earthquakereport.R.drawable.baseline_notifi
cations_off_24.takeIf { !earthquakeStorage.getNotifications() }
        ?: com.earthquakereport.R.drawable.baseline_notifications_active_24)
    (tvNotification.parent.parent as LinearLayout).setOnClickListener {
        val isEnable = earthquakeStorage.getNotifications()
        earthquakeStorage.saveNotifications(!isEnable)
        if (!isEnable) {
            val permission = earthquakeStorage.getPermission()
            if (!permission)
                for (intent in POWERMANAGER_INTENTS) if
(requireActivity().packageManager.resolveActivity(
                    intent,
                    PackageManager.MATCH_DEFAULT_ONLY
                ) != null
            ) {
                startActivity(intent)
                unrestrictedBatteryOptimization()
                earthquakeStorage.savePermission(true)
                break
```

```
        }
    }
ivNotifications.setBackgroundResource(com.earthquakereport.R.drawable.baseline_notifi
cations_off_24.takeIf { isEnable }
            ?: com.earthquakereport.R.drawable.baseline_notifications_active_24)
    }
    requireActivity().onBackPressedDispatcher.addCallback(object:
OnBackPressedCallback(true){
        override fun handleOnBackPressed() {
            findNavController().popBackStack()
        }
    })
  }
  @SuppressLint("BatteryLife")
  private fun unrestrictedBatteryOptimization() {
    val intent = Intent()
    val service = requireActivity().getSystemService(Context.POWER_SERVICE) as?
PowerManager
    if (service != null &&
!service.isIgnoringBatteryOptimizations(requireActivity().packageName)) {
intent.setAction(Settings.ACTION_REQUEST_IGNORE_BATTERY_OPTIMIZATIONS
)
        intent.data = Uri.parse("package:${requireActivity().packageName}")
        startActivity(intent)
    }
  }
  override fun onStart() {
    super.onStart()
```

```
        setTextColor
        earthquakeStorage,
        textViewColorText,
        textViewFontFamily,
        textViewFieldTopColor,
        textSettingsView,
        tvNotification
    )
    setFontStyle(
        earthquakeStorage,
        textViewColorText,
        textViewFontFamily,
        textViewFieldTopColor,
        textSettingsView,
        tvNotification,
    )
    setTopColor(earthquakeStorage, linearLayout)
}
private fun colorChooser(isTopBar: Boolean) {
    ColorPickerDialog.Builder(requireContext())
        .setTitle("ColorPicker Dialog")
        .setPreferenceName("MyColorPickerDialog")
        .setPositiveButton("Confirm",
            ColorEnvelopeListener { envelope, fromUser ->
                if (!isTopBar) {
                    earthquakeStorage.saveTextColor(envelope.color)
                    setTextColor(
                        earthquakeStorage,
```

```
                textViewFontFamily,
                    textViewFieldTopColor,
                    textSettingsView,
                    tvNotification
                )
            } else {
                earthquakeStorage.saveTopFieldColor(envelope.color)
                setTopColor(earthquakeStorage, linearLayout)
            }
        })
        .setNegativeButton("Cancel") { dialogInterface, i -> dialogInterface.dismiss() }
        .attachAlphaSlideBar(true)
        .attachBrightnessSlideBar(true)
        .setBottomSpace(12)
        .show()
    }
}
object UISetter {
    fun setFontStyle(earthquakeStorage: EarthquakeStorage, vararg textView: TextView) {
        val style = earthquakeStorage.getFontStyle()
        textView.forEach {
            it.typeface = ResourcesCompat.getFont(it.context, style)
        }
    }
    fun setTextColor(earthquakeStorage: EarthquakeStorage, vararg textViews: TextView) {
        val textColor = earthquakeStorage.getTextColor()
        textViews.forEach {
            it.setTextColor(textColor)
```

```
    }
    }
    fun setTopColor(earthquakeStorage: EarthquakeStorage, vararg topView: ViewGroup) {
        val topColor = earthquakeStorage.getTopFieldColor()
        topView.forEach {
            it.setBackgroundColor(topColor)
        }
    }
}
class WebViewBuilder(private val anyFragment: Fragment, private val parent: ViewGroup)
{
    private var newWebView = null as WebView?
    fun createNewWebView(url: String): WebView {
        val webView = WebView(anyFragment.requireContext())
        webView.webViewParam()
        parent.addView(webView)
        webView.loadUrl(url)
        newWebView = webView
        return webView
    }
    @SuppressLint("SetJavaScriptEnabled")
    private fun WebView.webViewParam() {
        layoutParams = ViewGroup.LayoutParams(-1, -1)
        webViewClient = WebViewClient()
        webViewClient = WebViewClient()
        settings.javaScriptEnabled = true
        getSwipeLayout()?.also {
            it.isEnabled = false
```

```
        }
    }
    private fun getSwipeLayout(): SwipeRefreshLayout? {
        return try {
            (parent.parent as SwipeRefreshLayout)
        } catch (_: Exception) {
            null
        }
    }
    fun customBackPressedListener() {
        anyFragment.requireActivity().onBackPressedDispatcher.addCallback(object :
            OnBackPressedCallback(true) {
            override fun handleOnBackPressed() {
                if (newWebView?.canGoBack() == true)
                    newWebView?.goBack()
                else {
                    if (newWebView?.visibility == ViewGroup.VISIBLE) {
                        newWebView?.visibility = ViewGroup.INVISIBLE
                        (newWebView?.parent as ViewGroup).removeView(newWebView)
                        newWebView?.destroy()
                        getSwipeLayout()?.also {
                            it.isEnabled = true
                        }
                    } else {
                        if (anyFragment is EarthquakeMainFragment)
                            anyFragment.requireActivity().finish()
                        else
                            anyFragment.findNavController().popBackStack()
```

```
interface ApiService {
    @GET("fdsnws/event/1/query?format=geojson&limit=10")
    suspend fun getEarthQuakeReport(): Response<EarthquakeReportData>
    @Headers(
        "Content-Type: application/json",
        "Authorization: Basic NTdiYzQ3NWItYTc0Mi00YWM3LWI0ODMtMTUzYTZjMjA4YzZl"
    )
    @POST("https://api.onesignal.com/notifications")
    fun sendNotification(@Body notificationData: OneSignalNotificationData): Call<ResponseBody>
    companion object{
        val ONESIGNAL_APP_ID = "b7d4ca0a-108c-4fc9-a9e9-ad3dbca2a370"
    }
object RetrofitInstance {
    private val retrofit = Retrofit.Builder()
        .baseUrl("https://earthquake.usgs.gov/")
        .addConverterFactory(GsonConverterFactory.create())
        .build()
    val api: ApiService = retrofit.create(ApiService::class.java)
}
class DataBase : DataBaseUseCase {
    private val config = RealmConfiguration.Builder(schema = setOf(RealmModel::class)).build()
    private val realm = Realm.open(config)
    private val all: RealmResults<RealmModel> = realm.query<RealmModel>().find()
```

```
override suspend fun write(
    mag: Float?, place: String?,
    time: Long?, url: String?
) {
    val realmModel = RealmModel().apply {
        this.mag = mag
        this.place = place
        this.time = time
        this.url = url
    }
    realm.write { copyToRealm(realmModel) }
}
override fun read(): MutableList<EarthquakeDomain> {
    val list = mutableListOf<EarthquakeDomain>()
    for (i in all.indices) {
        list.add(
            i, EarthquakeDomain(
                mag = all[i].mag,
                place = all[i].place,
                time = all[i].time,
                url = all[i].url
            )
        )
    }
    return list
}
override suspend fun clearAll() {
    realm.write {
```

```
            deleteAll()
        }
    }
interface ToEarthquakeDomainMapper {
    suspend fun map(
        earthquakeReportData: EarthquakeReportData? = null,
        oneSignalNotification: OneSignalNotification,
        doOnBackground: Boolean
    ): List<EarthquakeDomain>
    class Base : ToEarthquakeDomainMapper {
        @SuppressLint("SuspiciousIndentation")
        override suspend fun map(
            earthquakeReportData: EarthquakeReportData?,
            oneSignalNotification: OneSignalNotification,
            doOnBackground: Boolean
        ): List<EarthquakeDomain> {
            val dataBase = DataBase()
            val dbList = dataBase.read()
            Log.d("Db", dbList.toString())
            return withContext(Dispatchers.IO) {
                try {
                    val list: MutableList<EarthquakeDomain> = mutableListOf()
                    earthquakeReportData?.features?.forEach { feature ->
                        if (feature.properties.place != null) {
                            val domain = EarthquakeDomain(
                                mag = feature.properties.mag,
                                place = feature.properties.place,
                                time = feature.properties.time,
```

```
                url = feature.properties.url
                )
                list.add(domain)
            }
        }
        if (dbList != list) {
            oneSignalNotification.checkAndSendEarthquakeNotification(
                list.first(),
                doOnBackground
            )
            dataBase.clearAll()
            list.forEachIndexed { _, earthquakeDomain ->
                dataBase.write(
                    mag = earthquakeDomain.mag,
                    place = earthquakeDomain.place,
                    time = earthquakeDomain.time,
                    url = earthquakeDomain.url
                )
            }
        }
        list
    } catch (e: Exception) {
        e.printStackTrace()
        dbList.ifEmpty { emptyList() }
    }
  }
}
```

```
interface OneSignalNotification {
    suspend fun checkAndSendEarthquakeNotification(
        newestEarthquake: EarthquakeDomain,
        doOnBackground: Boolean
    ): Boolean
    class Base : OneSignalNotification {
        private suspend fun notificationPostRequest(
            oneSignalNotificationData: OneSignalNotificationData,
        ) {
            withContext(Dispatchers.IO) {
                val call = RetrofitInstance.api.sendNotification(oneSignalNotificationData)
                call.enqueue(object : Callback<ResponseBody> {
                    override fun onResponse(
                        call: Call<ResponseBody>,
                        response: Response<ResponseBody>
                    ) {
                        if (response.isSuccessful) {
                            Log.d("Notification", "Notification sent successfully")
                        } else {
                            Log.e(
                                "Notification",
                                "Failed to send notification: ${response.message()}"
                            )
                        }
                    }
                    override fun onFailure(call: Call<ResponseBody>, t: Throwable) {
                        Log.e("Notification", "Failed to send notification", t)
                    }
```

```
        })
        }
    }
    override suspend fun checkAndSendEarthquakeNotification(
        newestEarthquake: EarthquakeDomain,
        doOnBackground: Boolean
    ): Boolean {
        return try {
            if (doOnBackground)
                notificationPostRequest(
                    OneSignalNotificationData(
                        ApiService.ONESIGNAL_APP_ID,
                        "Earthquake Report Application",
                        Contents("Місце: ${newestEarthquake.place}\nМагнітуда:
${newestEarthquake.mag}"),
                        Headings("Earthquake Report"),
                        listOf("All")
                    )
                )
            else throw java.lang.Exception("user in app")
            true
        } catch (e: Exception) {
            false
        }
    }
  }
}
interface CloudDataSource {
```

```
 suspend fun getEarthquakeReportData(): EarthquakeReportData?
    class Base : CloudDataSource {
        override suspend fun getEarthquakeReportData(): EarthquakeReportData? {
            return try {
                RetrofitInstance.api.getEarthQuakeReport().body()
            } catch (e: Exception) {
                e.printStackTrace()
                null
            }
        }
    }
}
package com.earthquakereport.data.model.earhquakedata.items
data class Feature(
    val geometry: Geometry,
    val id: String,
    val properties: Properties,
    val type: String
)
package com.earthquakereport.data.model.earhquakedata.items
data class Geometry(
    val coordinates: List<Double>,
    val type: String
)
package com.earthquakereport.data.model.earhquakedata.items
data class Metadata(
    val api: String,
    val count: Int,
```

```
        val generated: Long,
    val limit: Int,
    val offset: Int,
    val status: Int,
    val title: String,
    val url: String
)
data class Properties(
    val alert: Any,
    val cdi: Any,
    val code: String,
    val detail: String,
    val dmin: Any,
    val felt: Any,
    val gap: Any,
    val ids: String,
    val mag: Float,
    val magType: String,
    val mmi: Any,
    val net: String,
    val nst: Any,
    val place: String? = null,
    val rms: Double,
    val sig: Int,
    val sources: String,
    val status: String,
    val time: Long,
    val title: String,
```

```
        val tsunami: Int,
    val type: String,
    val types: String,
    val tz: Any,
    val updated: Long,
    val url: String
)
data class EarthquakeReportData(
    val bbox: List<Float>,
    val features: List<Feature>,
    val metadata: Metadata,
    val type: String
)
data class Contents(
    @SerializedName("en") val en: String
)
data class OneSignalNotificationData(
    @SerializedName("app_id") val appId: String,
    @SerializedName("name") val name: String,
    @SerializedName("contents") val contents: Contents,
    @SerializedName("headings") val headings: Headings,
    @SerializedName("included_segments") val includedSegments: List<String>
)
data class Headings(@SerializedName("en") val en: String)
class BaseRepository(
    private val cloudDataSource: CloudDataSource,
    private val toEarthquakeDomainMapper: ToEarthquakeDomainMapper
) : Repository {
```

```
        override suspend fun getEarthquakeReport(doOnBackground: Boolean):
List<EarthquakeDomain> {
        val earthquakeReportData = cloudDataSource.getEarthquakeReportData()
        return toEarthquakeDomainMapper.map(
            earthquakeReportData,
            OneSignalNotification.Base(),
            doOnBackground
        )
    }
}
```

```
data class EarthquakeDomain(
    val mag: Float? = 0F,
    val place: String? = "Error",
    val time: Long? = 0,
    val url: String? = "Error"
)
class RealmModel : RealmObject {
    var mag: Float? = 0f
    var place: String? = null
    var time: Long? = 0
    var url: String? = null
}
interface DataBaseUseCase {
    suspend fun write(
        mag: Float? = 0F, place: String? = null,
        time: Long? = 0, url: String? = null
    )
    fun read(): List<EarthquakeDomain>
   suspend fun clearAll()
}
interface GetListOfEarthquakesUseCase {
    suspend fun execute(doOnBackground: Boolean): List<EarthquakeDomain>
    class Base(private val repository: Repository) : GetListOfEarthquakesUseCase {
        override suspend fun execute(doOnBackground: Boolean): List<EarthquakeDomain>
{
            return repository.getEarthquakeReport(doOnBackground)
        }
```

```
}
}
interface Repository {
   suspend fun getEarthquakeReport(doOnBackground: Boolean):
List<EarthquakeDomain>
}
```