МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Факультет інформатики

Кафедра інформатики

ТЕМА

АРХІТЕКТУРА РОЗРОБКИ МОБІЛЬНИХ ЗАСТОСУНКІВ

КУРСОВА РОБОТА

Виконав: студент III курсу

Спеціальність “Комп’ютерні науки”

Ладошка Артур Сергійович

Науковий керівник:

Салата К. В.

КИЇВ 2022

Тема: архітектура розробки мобільних застосунків

Календарний план виконання роботи:

№	Назва етапу	Дата	Примітка
1	Отримання теми курсової роботи	10.10.2021	
2	Пошук тематичної літератури	11.10.2021	
3	Ознайомлення з літературою	20.10.2021	
4	Вивчення аналогів	01.12.2021	
5	Аналіз архітектур розробки	01.01.2022	
6	Аналіз патернів проектування	16.01.2022	
7	Порівняння мобільних платформ	25.01.2022	
8	Написання програми	25.04.2022	
9	Відлагодження додатку	04.05.2022	
10	Написання текстової частини	15.05.2022	
11	Перегляд змісту роботи керівником	19.05.2022	
12	Внесення змін до курсової роботи	25.05.2022	
13	Створення презентації	26.05.2022	
14	Захист роботи	06.06.2022	

Зміст

Вступ

У зв’язку зі зростанням популярності мобільних додатків виник великий попит на розробку застосунків для пристроїв базованих на платформах IOS та Android. Відповідно до статистики використання мобільних додатків за 2021 рік, людство активно використовує 6,3 мільярда користувачів смартфонів та 1,14 мільярда користувачів планшетів у всьому світі зі зростаючою динамікою протягом років. Людина перевіряє свій смартфон кожні 5,5 хвилини, що становить 262 рази на день. 88% часу, проведеного користувачами в телефонах, припадає саме на мобільні додатки.

Середній власник смартфона використовує 10 додатків на день і 30 додатків щомісяця. Минулого 2021 року було завантажено понад 39 мільярдів додатків. Це приблизно на 7% більше, ніж торік. 98% доходу від додатків у всьому світі надходить від безкоштовних програм, тому частіше розробники дотримують стратегії монетизації безкоштовної, або умовно безкоштовної, тобто саме завантаження з маркету є безкоштовним, проте частина функціоналу є недоступною без внутрішньої оплати, в другому випадку ліцензія на використання програми дається лише на обмежений період часу, частіше тиждень або місяць.

Користувач отримує додатки, які використовує для власних потреб з магазину Google Play (платформа Android), який містить 2,87 міліонна програм, та App Store(платформа IOS), що надає доступ до 1,96 мільйона програм. Кількість завантажень додатків з App Store за 2020 становить 8,2 мільярдів, з Google Play - 28,3 мільярдів, а приріст порівняно з 2019 роком – 31% для Android та 2,5% для IOS. З цього робимо висновок, що аудиторія Android набагато ширша за спільноту IOS, однак це зовсім не означає, що при розробці мобільних додатків та вибору архітектури розробки застосунків для мобільних платформ, слід обирати якусь одну з них, а варто орієнтуватись на кросплатформенний підхід для охоплення більшої кількості клієнтів.

Потреба у створенні сучасних мобільних застосунків означає для розробників мобільних платформ дізнатись більше про архітектури розробки мобільних додатків, провести їх аналіз та порівняння. Знання про архітектури розробки значно розширить можливості розробника програмувати для мобільної платформи, що набагато прискорить процес написання мобільного додатку.

Частина 1: дослідження та порівняння архітектур розробки мобільних застосунків

Архітектура розробки мобільного застосунку – це підходи, правила та шаблони розробки мобільного додатку. Архітектури розробки розділяють за платформами.

1.1. Архітекура розробки мобільного застосунку для Android мовами Java та Kotlin:

Операційна система Android – це багатокористувацька система Linux, в якій кожен додаток є іншим користувачем.

За замовчуванням система призначає кожній програмі унікальний ідентифікатор користувача Linux (ідентифікатор використовується тільки системою і невідомий програмі). Система встановлює дозволи для всіх файлів у програмі, щоб лише ідентифікатор користувача, призначений цій програмі, мав доступ до них. Інструменти Android SDK дозволяють компілювати код разом з набором файлів та даних у файл з розширенням “.apk” абж у пакет Android App Bundle . Apk-файл викороистовують під час інсталяції програми на мобільному пристрої. Файл AAB є фінальним форматом для розповсюдження додатку Android, включає в себе повний вміст проекту програми Android та метадані які не потрібні під час звичайного виконання, його не можна встановити, оскільки він відкладає створення і підписання APK файлу на пізніший етап.

Існує 4 види складових програми: Activities, Services, Broadcast receivers, Content providers. Activities є точкою входу для взаємодії з користувачем, фактично означає один екран з UI. Один екран є імплементацією класу Activity. Воно сприяє ключовим взаємодіям між системою та програмою: Відстеження того, що в даний момент цікавить користувача, аби повідомити

систему про продовження виконання процесу в якому розміщена Activity. Забезпечення реалізації потоків користувачів.

Service – це загальноприйнята точка входу призначена для того аби тримати програму у фоновому режимі. На відміну від Activity, не надає інтерфейс користувача, за допомогою іншого компонента дозволяє запускати службу або підключитися до служби для подальшої взаємодії з нею. Поділяються на запущені служби, мета яких забезпечити функціонування служб до їх завершення, для прикладу синхронізація повідомлень у меседжерах, та зав'язані служби, які запускаються виключно у випадку виклику служби іншою системою або програмою. Яскравим прикладом використання служб є прослуховувачі сповіщень, меседжери, музичні аудіопрогравачі, програми запису звуку, обробки мультимедіа та антивіруси.

Broadcast receiver це компонент, мета якого дозволяти системі доставлення подій програмі без урахування звичайного потоку звичайних користувачів, реагуючи на повідомлення системи про трансляцію. Головний клас – BroadcastReceiver, і кожен бродкаст доставляє 1 клас підзавдання Intent. Приклад: передача програмам інформації про наявність файлу чи увімкнення таймеру на пристрої.

Content provider постачає інформацію та розпоряджається даними з бази даних, Інтернету чи файлової системи. Саме він відповідає за читання/запис даних. Це точка входу в програму для публікації іменованих елементів даних, ідентифікованих схемою URI. Як приклад можна провести анонімне занесення даних в буфер обміну, тобто скористатися ним може виключно ця програма, розміщуючи URI об'єкта в буфер обміну, а ContentProvider буде заблокованим.

Активними компонентами з чотирьох складових є три: Activities, Services, Broadcast receivers. Intent це клас для приведення в дію цих активних компонент, асинхронне повідомлення про взаємодію інших компонентів. Передає запит на дію, наприклад "Оберіть фото з галереї", або обрання

способу відкриття файлу. ContentResolver працює по-іншому, він очікує запиту від однойменного класу й обробляє всі транзакції й викликає методи його об’єкту.

Функціонал маніфесту програми включає: забезпечення інформування системи про існування компонентів програми, опис дозволів програми, апаратні й п системні функції, мінімальну версію Android API, посилання на зовнішні бібліотеки, за умови, що файл маніфесту повинен знаходитись у кореневій папці проекту.

Взаємодія з медіаресурсами базується на використанні технології XML. Інструментами Android SDK утворено цілочисельний ідентифікатор конкретного ресурсу для використання у вигляді посилання. Кваліфікатор - це рядок, який включає назву каталогів ресурсів, з метою визначення конфігурації пристрою, цільового для використання цих ресурсів.

Структура розробки мобільного додатка на базі Android включає в себе 3 шари, які ідуть згори донизу шар користувацького інтерфейсу (UI Layer), доменів (Domain Layer), та шар даних (Data Layer). На рівні інтерфейсу користувача відображаються екран з даними програми, при зміні яких інтерфейс повинен оновлюватись, цей шар складається з двох частин – елементів інтерфейсу та State Holder, функція якого збереження даних, обробка логіки та надання даних до інтерфейсу. Доменний рівень включає в себе бізнес-логіку, яка повторно використовується кількома моделями перегляду , він є необов’язковим шаром, і частіше використовується лише в складних програмах. Він включає в себе класи які є варіантами використання, інтерактораами, які відповідають за певну функціональність. Рівень даних містить сховища (Repositories), що вкоючають джерела даних (Data sources), що диференціюються залежно від типу та виконують функції надання доступу даних до програми, вирішення конфліктів між джерелами, опис бізнес-логіки, відокремлення даних від решти програми. Джерела поділяються на внутрішні – локальна база даних, та зовнішні – передусім Internet.

1.2. Архітектура розробки кросплатформеного мобільного застосунку для Android та IOS за допомогою фреймворку Xamarin

Платформа Xamarin передбачає створення структури програмних файлів таким чином, аби цей код використовувався різними платформами. Варто зазначити, що бібліотека орієнтована на 3 платформи Android, IOS, та Windows 10. Розглянемо перші дві, передбачаючі використання мобільних пристроїв. Архітектура складається з 6 шарів, а саме: UI layer – користувацький інтерфейс програми, екрани та компоненти управління. Application Layer – рівень, програмний код у якому є плаформо-залежним, часто включає контролери відображення, що не можуть бути спільними для обох платформ. Service Access Layer – це рівень застосунку, який використовується для виклику хмарних служб та проводить інкапсуляцію поведінки мережі, у застосунках не пов’язаних з Мережею не використовується. Business Layer визначає бізнес-логіку. Data Access Layer є шаром забезпечуючим чотири елементарні операції при роботі з базами даних – створення, читання, оновлення та видалення, однак не надає інформацію про реалізації об’єкту. І останній шар, Data Layer – це база даних.

Яким же чином відбувається взаємодія кросплатформенної основи та конкретної реалізації для ОС Android чи IOS? Розглянемо на прикладі Android. Спочатку вони компілюються у проміжну мову IL, яка у випадку виконання застосунку компілює Just-in-time до нативної збірки. Головним для Android перетворення є середовище виконання Mono та віртуальна машина Android Runtime ART, забезпечуючи прив’язку до просторів імен Java Android. Mono викликає ці простори через керовані обгортки виклику (MCW) й забезпечує обгортку виклику Android до ART, що надає змогу обом середовищам викликати один й той самий код один в одному. З IOS ситуація схожа, однак замість компіляції через оболонку використаються iOS-зкомпільовані з C# у власний код збірки ARM у мову Objective-C.

1.3. Архітекура розробки кросплатформеного мобільного застосунку для Android та IOS за допомогою фреймворку React Native

У середовищі виконання React Native є три ключові потоки: потік JavaScript, власний основний потік і фоновий потік для обробки Shadow Node. У поточній архітектурі зв’язок між цими потоками відбувається через бібліотеку, яка називається «Bridge» . Він надає абстракцію для декларативного інтерфейсу користувача. Віртуальне представлення інтерфейсу користувача зберігається в пам’яті та синхронізується із зовнішніми бібліотеками інтерфейсу користувача. Цей процес називається “примиренням”.

React Native надає власний рівень абстракції інтерфейсу користувача на платформах iOS та Android. Ядро React Native і нативні компоненти викликають уявлення, що можливо написати інтерфейс програми для смартфона за допомогою JavaScript. Нативні компоненти охоплюють комплексні вбудовані елементи інтерфейсу користувача, але необхідно реалізувати багато компонентів, щоб імітувати складніший приклад, наприклад навігацію по вкладках.

React Native тісно інтегрується з Jest і забезпечує чіткі методології тестування разом із популярними бібліотеками, такими як React Navigation і Redux. Одна реалізація знижує витрати на розробку та обслуговування. Це покращує продуктивність створення React Shadow Trees і обчислення макета, оскільки накладні витрати на інтеграцію Yoga з рендерером зведені до мінімуму на Android (фреймворк JNI). Також, обсяг пам’яті кожного React Shadow Node менший ніж C++, а саме , якби він був виділений з Kotlin або Swift .

Завдяки покращеній сумісності між представленнями хоста та переглядами React, засіб візуалізації може вимірювати та відображати поверхні React синхронно. У застарілій архітектурі макет React Native був асинхронним, що призвело до проблеми «перескоку» макета під час вбудовування відображеного React Native подання в подання хоста.

Завдяки підтримці багатопріоритетних і синхронних подій засіб візуалізації може визначити пріоритети певних взаємодій користувача, щоб гарантувати, що вони обробляються вчасно. Інтеграція з React Suspense, яка забезпечує більш інтуїтивно зрозумілий дизайн отримання даних у програмах React. Увімкнення паралельних функцій React на React Native. Простіше реалізувати рендеринг на стороні сервера для React Native.

Завдяки новій кросплатформній реалізації системи візуалізації кожна платформа отримує вигоду від покращення продуктивності, що, можливо, було спричинено обмеженнями однієї платформи. Наприклад, вирівнювання вигляду спочатку було ефективним рішенням для Android, а тепер воно за замовчуванням надається як на Android, так і на iOS.

React API розроблено так, щоб бути декларативним і повторно використовуватися за допомогою композицій. Це забезпечує чудову модель для інтуїтивного розвитку. Однак у реалізації ці якості API призводять до створення глибоких дерев елементів React, де велика більшість вузлів React Element впливає лише на макет View і нічого не відображає на екрані. Ми називаємо ці типи вузлів вузлами «лише макет».

Кожен із вузлів дерева елементів React має співвідношення 1:1 з виглядом на екрані, тому відтворення глибокого дерева елементів React, яке складається з великої кількості вузлів «Тільки для макета», призводить до поганої продуктивності під час надання.

1.4. Патерни проектування, які застосовуються в мобільній розробці

1) MVC – Model, View, Controller.

Model: Цей компонент зберігає дані програми. Він не знає про інтерфейс. Модель відповідає за обробку логіки домену (реальних бізнес-правил) і зв'язку з базою даних і мережевими рівнями.

View: це шар інтерфейсу користувача (Інтерфейс користувача), який містить компоненти, видимі на екрані. Крім того, він забезпечує візуалізацію даних, що зберігаються в моделі, і пропонує взаємодію з користувачем.

Controller: цей компонент встановлює зв'язок між представленням і моделлю. Він містить основну логіку програми та отримує інформацію про поведінку користувача та оновлює модель відповідно до потреб.

Коли використовується: для звичайних програм, для розробки клієнт-серверної архітектури.

Переваги: одноманітна глобальна архітектура застосунку, легкість засвоєння архітектури новому члену команди розробників, тестування та відлагодження модулів

Недоліки: неоптимізованість, оскільки модулі взаємодіють між собою лише шляхом передачі данних, необхідність мати більше обмежень для даних, кожен модуль повинен включати три складові.

2) MVP

Model: шар для зберігання даних. Він відповідає за обробку логіки домену (реальних бізнес-правил) і зв'язок з базою даних і мережевими рівнями.

View: шар інтерфейсу користувача (Інтерфейс користувача). Він забезпечує візуалізацію даних і відстежує дії користувача, щоб сповістити доповідача.

Presenter: отримує дані з моделі та застосовує логіку інтерфейсу користувача, щоб вирішити, що відображати. Він керує станом View та виконує дії відповідно до сповіщень, які користувач вводить із View.

Коли використовується: при обмеженій вибірці компонентів інтерфейсу користувача та простій навігації в додатку.

Переваги: повторне використання коду, підхід, орієнтований на тестування – ізолювання компонентів, адаптивний дизайн, відокремлення логіки перегляду від

Недоліки: складність, поріг входження у проект, не варто обирати для простих програм.

3) MVVM: Model, View, ViewModel.

Model: тут зберігаються дані програми. Він не може безпосередньо спілкуватися з View. Як правило, рекомендується надавати дані ViewModel через Observables.

View: він представляє інтерфейс користувача програми, позбавлений будь-якої логіки програми. Він спостерігає за ViewModel.

ViewModel: діє як сполучна ланка між моделлю та представленням. Він відповідає за перетворення даних з моделі. Він надає потоки даних до View. Він також використовує хуки або зворотні виклики для оновлення View та запитує дані з моделі.

Коли використовується: при розробці додатків, з подальшою перспективою їх розширення.

Переваги: полегшує паралельну розробку інтерфейсу користувача та будівельних блоків, які його забезпечують. абстрагує View і таким чином зменшує кількість бізнес-логіки, ViewModel простіше використовувати для модульного тестування без проблем автоматизації UI та взаємодії.

Недоліки: прив’язки важче налагодити, ніж імперативний код, у великих програмах може бути складніше розробити ViewModel наперед, щоб отримати необхідну кількість узагальнень.

4) VIPER – View, Interactor, Presenter, Entity, Router.

Перегляд: рівень інтерфейсу, який включає файли UIKit, називається View (включаючи UIViewController). У більш відокремленій архітектурі очевидно, що підкласи UIViewController повинні належати до рівня перегляду. У VIPER все майже так само, як і в MVVM: перегляди відповідають за показ того, про що їх просить доповідач, і передають відгуки користувачів назад доповідачеві.

Interactor: Ця частина включає бізнес-логіку, викладену у випадках використання програми. Інтерактор відповідає за отримання даних з рівня моделі (через мережу або локальну базу даних), і він повністю не залежить від інтерфейсу користувача. Оскільки адміністратори мережі та бази даних не входять до VIPER, вони обробляються як окремі залежності.

Доповідач: цей компонент містить логіку перегляду для форматування даних, які будуть відображатися. Це частина роботи, яку ViewModelController у нашому прикладі виконує в MVVM. Доповідач бере дані з інтерактора, генерує випадок моделі подання та передає їх у View. Він також реагує на відгуки користувачів, запитуючи додаткову інформацію або повертаючи її інтерактору.

Суб'єкт: в інших архітектурах він поділяє деякі зобов'язання рівня моделі. Інтерактор і менеджери даних обробляють сутності як базові структури даних без бізнес-логіки.

Маршрутизатор: логіка навігації програми. Здається, це не великий шар, але якщо вам доведеться повторно використовувати одні й ті самі подання iPhone у додатку для iPad, єдина різниця полягає в тому, як вони відображаються. Це захищає ваші інші рівні, а маршрутизатор відповідає за потік навігації в кожному випадку.

Переваги: Найбільш адаптована архітектура для великих команд, Код розділений для повторного використання та перевірки, Зменшується кількість конфліктів прості й чітко визначені інтерфейси користувача, спрощення складних завдань.

Недоліки: зв'язок між компонентами дуже міцний, сторонні пакети SDK не підтримуються, можливі проблеми з продуктивністю.

Частина 2 Побудова Мобільного застосунку та опис архітектури його розробки

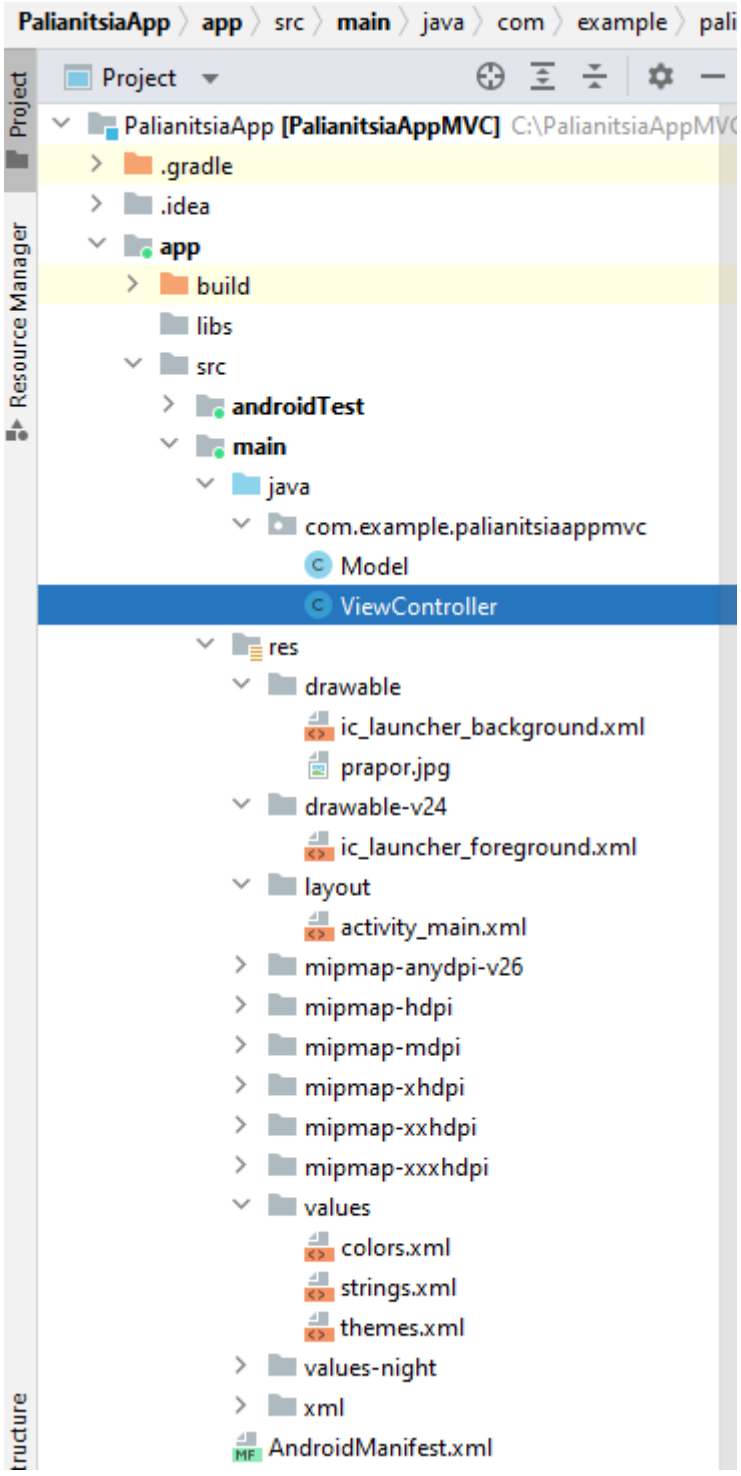
Мета: створити застосунок, що реалізує загальну трирівневу архітектуру розробки мобільних застосунків, конкретизовану патерном MVC.

Вибір архітектури та платформи вівся серед Xamarin та Java Android SDK. Хоч і Xamarin є кросплатформеним фреймворком, однак його властивість досягається лише при найменшому наборі унікального для платформи коду.

Шаблон проектування обирався серед MVC, MVVM, MVP, VIPER. 2 останні втрачають свою актуальність через орієнтацію їх на більші, або специфічніші програмні проекти. Недостатком MVVM, з іншого боку, є дозвіл створювати специфічні для перегляду підмножини моделі, які можуть містити як логіку, так і стан, і таким чином уникнути доступу до всієї моделі представленням. MVVM забезпечує переваги безпеки та продуктивності для n-рівневих додатків, він має недолік, пов’язаний з підвищеною складністю, і, таким чином, MVC обрано тут, де дані переваги не потрібні.

Використовуємо IDE Android Studio, на базі Android SDK та мови програмування Java. Розроблено проект магазину зброї “Palianitsia” через патерн MVC.

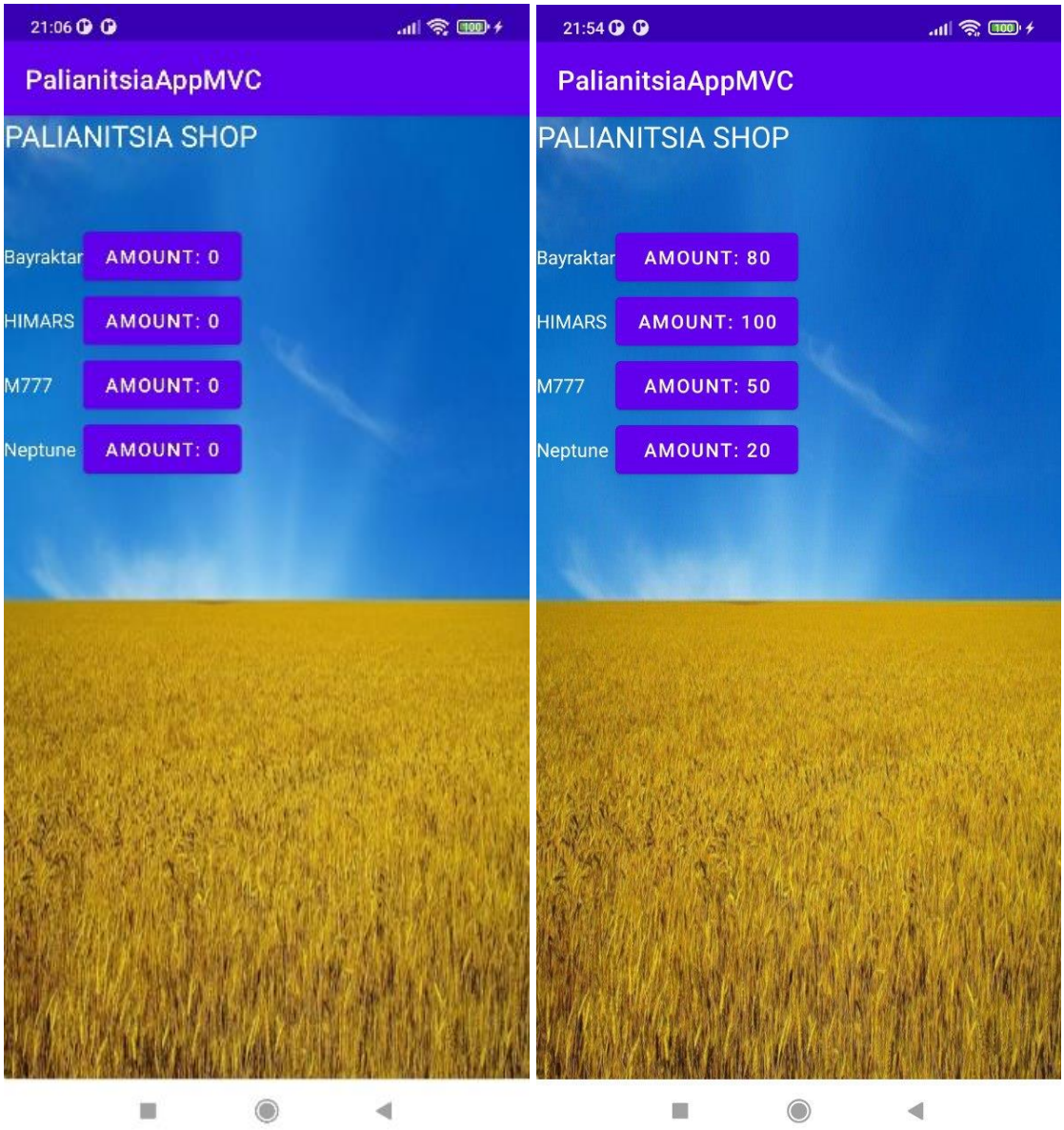
Структура проекту:



Програма у дії:

При натисненні на кнопку, ідентифікатори яких weapon1-weapon4, спрацьовує метод-обробник подій onClick(), викликається метод setJavelinIndex(), який підраховує значення для конкретної кнопки, обраної за ідентифікатором підраховує нове значення для Моделі, використавши

setChanged(), щоб сповістити суперклас Observer про те, що відбулася зміна. Після чого метод апдейт реагує на подію зміну значень тексту кнопок у Моделі, та оновлює їх у користувацькому інтерфейсі.



Android – маніфест застосунку: AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.example.palianitsiaappmvc">

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="PalianitsiaAppMVC"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.PalianitsiaAppMVC"
        tools:targetApi="31">
        <activity
            android:name=".ViewController"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

2.1 Реалізація Model

Клас Model відповідає за представлення зброї, включає приватний член-список javelins, конструктор, що створє порожній список та присвоює кожній клітинці 0, геттер та сеттер, який додає до значення кількості однієї зброї 10.

```java
package com.example.palianitsiaappmvc;

import java.util.ArrayList;
import java.util.List;
import java.util.Observable;

public class Model extends Observable {
    private List<Integer> javelins;
    public Model(){
        javelins = new ArrayList<Integer>( initialCapacity: 4);
        for(int i=0; i<4; i++){
            javelins.add(0);
        }
    }
    public int getJavelinIndex(final int idJavelin) {
        return javelins.get(idJavelin);
    }
    public void setJavelinIndex(final int idJavelin) {
        javelins.set(idJavelin, javelins.get(idJavelin)+10);
        setChanged();
        notifyObservers();
    }
}
```

2.2 Реалізація View

Зовнішнє представлення та інтерфейс. Складається з файлів activity_main.xml, який описує компоненти головної Активіті (Android SDK), медіа файли. У файлі ViewController Java займає частину, яка є у методі onCreate, вказуючи програмі, який саме екран треба рендерити.

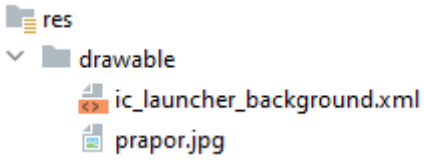
```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.and
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"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

    <ImageView
        android:id="@+id/imageView"
        android:layout_height="match_parent"
        android:layout_width="match_parent"
        app:srcCompat="@drawable/prapor"
        android:scaleType="fitXY"/>

    <TextView
        android:layout_width="190dp"
        android:layout_height="68dp"
        android:text="@string/programTitle"
        android:textAlignment="center"
        android:textColor="#FFFFFF"
        android:textSize="60px"
        tools:layout_editor_absoluteX="110dp"
        tools:layout_editor_absoluteY="64dp" />

    <TableLayout
        android:layout_width="409dp"
        android:layout_height="354dp"
        android:gravity="center"
        tools:layout_editor_absoluteX="110dp"
        tools:layout_editor_absoluteY="188dp">

        <TableRow
            android:layout_width="match_parent"
            android:layout_height="match_parent">
```



```
    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/weapon1"
        android:textColor="#FFFFFF" />

    <Button
        android:id="@+id/weaponAmount1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/counterBut" />


</TableRow>

<TableRow
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/weapon2"
        android:textColor="#FFFFFF" />

    <Button
        android:id="@+id/weaponAmount2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/counterBut" />
</TableRow>

<TableRow
    android:layout_width="match_parent"
    android:layout_height="match_parent">
```

```
            <TextView
                android:id="@+id/textView3"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/weapon3"
                android:textColor="#FFFFFF" />

            <Button
                android:id="@+id/weaponAmount3"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/counterBut" />
        </TableRow>

        <TableRow
            android:layout_width="match_parent"
            android:layout_height="match_parent">

            <TextView
                android:id="@+id/textView4"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/weapon4"
                android:textColor="#FFFFFF" />

            <Button
                android:id="@+id/weaponAmount4"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/counterBut" />


        </TableRow>
    </TableLayout>

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

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

palianitsiaModel = new Model();
palianitsiaModel.addObserver( o: this);
weapon1=(Button) findViewById(R.id.weaponAmount1);
weapon2=(Button) findViewById(R.id.weaponAmount2);
weapon3=(Button) findViewById(R.id.weaponAmount3);
weapon4=(Button) findViewById(R.id.weaponAmount4);
weapon1.setOnClickListener(this);
weapon2.setOnClickListener(this);
weapon3.setOnClickListener(this);
weapon4.setOnClickListener(this);
```

```
<resources>
    <string name="app_name">PalianitsiaAppMVC</string>
    <string name="weapon1">Bayraktar</string>
    <string name="weapon2">HIMARS</string>
    <string name="weapon3">M777</string>
    <string name="weapon4">Neptune</string>
    <string name="counterBut">Amount: 0</string>
    <string name="programTitle">PALIANITSIA SHOP</string>
</resources>
```

2.3 Реалізація Controller

Залежно від ідентифікатора натиснутої кнопки, звертається до Моделі та її методу setJavelinIndex(final int id), збільшуючи на 10 значення та зберігаючи результат до повторного натиснення або виходу.

```
private Button weapon1;
private Button weapon2;
private Button weapon3;
private Button weapon4;
private Model palianitsiaModel;
```

```
@Override
public void onClick(View v) {
    switch (v.getId()){
        case R.id.weaponAmount1:
            palianitsiaModel.setJavelinIndex(0);
            break;
        case R.id.weaponAmount2:
            palianitsiaModel.setJavelinIndex(1);
            break;
        case R.id.weaponAmount3:
            palianitsiaModel.setJavelinIndex(2);
            break;
        case R.id.weaponAmount4:
            palianitsiaModel.setJavelinIndex(3);
            break;
    }
}
```

```
@Override
public void update(Observable o, Object arg) {
    weapon1.setText("Amount: "+ palianitsiaModel.getJavelinIndex( idJavelin: 0));
    weapon2.setText("Amount: "+ palianitsiaModel.getJavelinIndex( idJavelin: 1));
    weapon3.setText("Amount: "+ palianitsiaModel.getJavelinIndex( idJavelin: 2));
    weapon4.setText("Amount: "+ palianitsiaModel.getJavelinIndex( idJavelin: 3));
}
```

Висновки

1. Проведено аналіз мобільних архітектур, усвідомлено визначну роль мобільних платформ у сучасних реаліях.

2. Для архітектури розробки сучасних мобільних додатків існує багато паттернів, з них виділяються чотири та мають свої переваги над іншими

3. На прикладі застосунку Palianitsia детально розглянуто патерн MVC

4. Виявлено недоліки розробки, які можна було уникнути використавши патерн MVVM, який також підходив для розробки цього застосунку.

Список використаної літератури

1) Native Mobile Development A Cross-Reference for iOS and Android Shaun Lewis and Mike Dunn [Електронне джерело]https://dokumen.pub/qdownload/native-application-development-a-cross-reference-for-ios-and-android-1492052809-9781492052807.html

2) Статистика використання мобільними пристроями (2020): https://techcrunch.com/2020/12/09/app-stores-to-see-130-billion-downloads-in-2020-and-record-consumer-spend-of-112-billion/

3) React Native : https://reactnative.dev/architecture/overview

4) Аналіз використання мобільних додатків (2021) https://www.businessofapps.com/marketplace/app-analytics/

5) Фреймворк Xamarin: https://docs.microsoft.com/en-us/xamarin/get-started/

6) Android SDK: https://developer.android.com/studio

7) Патерни проектування: https://www.developer.com/design/mvc-vs-mvp-vs-mvvm-design-patterns/

8) VIPER pattern: https://medium.com/@smalam119/viper-design-pattern-for-ios-application-development-7a9703902af6

9) React Native архітектура: https://dev.to/goodpic/understanding-react-native-architecture-22hh

10) “The comprehensive guide to application architecture in mobile development” : https://nix-united.com/blog/the-comprehensive-guide-to-mobile-application-architecture/