УДК 004.45

ВИКОРИСТАННЯ ДИЗАЙН-ПАТЕРНІВ MVVM ТА MVP ПРИ РОЗРОБЦІ МОБІЛЬНЫХ ЗАСТОСУНКІВ МОВОЮ KOTLIN

Козін Іван

Науковий керівник: канд. ф.-м. наук, доцент Паращук С.Д.

Центральноукраїнський державний педагогічний університет імені Володимира Винниченка, м. Кропивницький, Україна

Стаття присвячена огляду двох найпопулярніших архітектурних патернів, що використовуються при розробці мобільних застосунків під операційну систему Android мовою Kotlin. З цією метою були створені декілька застосунків за допомогою архітектури MVP та MVVM та виконане їх порівняння. Досліджена базова імплементація та реалізація архітектурних паттернів MVP та MVVM. Стаття буде корисна розробникам, які обирають підходящу для них архітектуру свого мобільного застосунку.

Ключові слова: Android, Kotlin, архітектурний патерн, MVP, MVVM.

USE OF MVVM AND MVP DESIGN PATTERNS IN THE DEVELOPMENT OF MOBILE APPLICATIONS IN THE KOTLIN LANGUAGE

I.Kozin

Scientific supervisor: Candidate of Physics and Mathematics Docent Paraschuk S.D.

Volodymyr Vynnychenko Central Ukrainian State Pedagogical University, Kropyvnytsky, Ukraine

The article is devoted to an overview of the two most popular architectural patterns used in the development of mobile applications for the Android operating system in Kotlin. To this end, several applications were created using the MVP and MVVM architectures and compared. The basic implementation and realization of architectural patterns MVP and MVVM are investigated. The article will be useful for developers who choose the appropriate architecture for their mobile application.

Keywords: Android, Kotlin, architectural pattern, MVP, MVVM.

Постановка проблеми

Наразі мобільні пристрої (смартфони, планшети) стають найбільш перспективним каналом комунікації та засобом оптимізації бізнес-процесів. Згідно з останніми дослідженнями IDC, у другому кварталі 2021 року обсяг поставок світових смартфонів досягав 313 мільйонів одиниць [1]. Ринкова

частка Android останні два роки тримається у межах 70-75% [2]. Іншими словами, приблизно сім з десяти пристроїв використовують саме цю ОС.

Багато власників бізнесу усвідомили, що присутність їхнього продукту на мобільних платформах є обов'язковою складовою результативної маркетингової стратегії.

Через велику кількість тем та інформації у процесі навчання основам розробки можуть виникати багато труднощів. Для систематизації та узагальнення знань існують спеціальні навчальні мобільні застосунки, які допомагають вивчити особливості розробки на цій платформі та перевірити знання на практиці за допомогою тестів або вправ.

Щоб успішно стартувати на ринку мобільних застосунки, потрібно прийняти кілька важливих рішення. Одним з них буде вибір правильного технології створення застосунки. Для аналізу ефективності вибору технології для розробки Android застосунків варто досліджувати наступні критерії:

Час розробки. Якщо взяти фахівців одного рівня в кожній з технологій і дати однакове технічне завдання, скільки часу буде потрібно, щоб розв'язати цю проблему за допомогою кожної з архітектур.

Наявність фахівців. Наскільки швидко можна знайти розробників, які здатні створити продукт на високому якісному рівні, а також фахівців, які зможуть його в подальшому супроводжувати, використовуючи кожну з архітектур.

Зручність розробки та налагодження. Наскільки розвинені інструменти розробки і налагодження в рамках даної архітектури.

Швидкість роботи. Наскільки швидким буде інтерфейс програми. Чи будуть помітні затримки в переходах між екранами і станами застосунка.

Постановка завдання

Мета статті полягає у висвітленні ефективності розробки та дослідженні можливостей глибини застосування архітектур MVP та MVVM мовою Kotlin для створення мобільних застосунків.

Виклад основного матеріалу дослідження

Для дослідження ефективності технологій розробки мобільних застосунків у середовищі Android Studio було створено два застосунка мовою Kotlin з імплементацією архітектур MVP та MVVM. Для створення застосунка з MVVM (рис.1) було взято декілька навчальних матеріалів з офіційного ресурсу від Google[6].

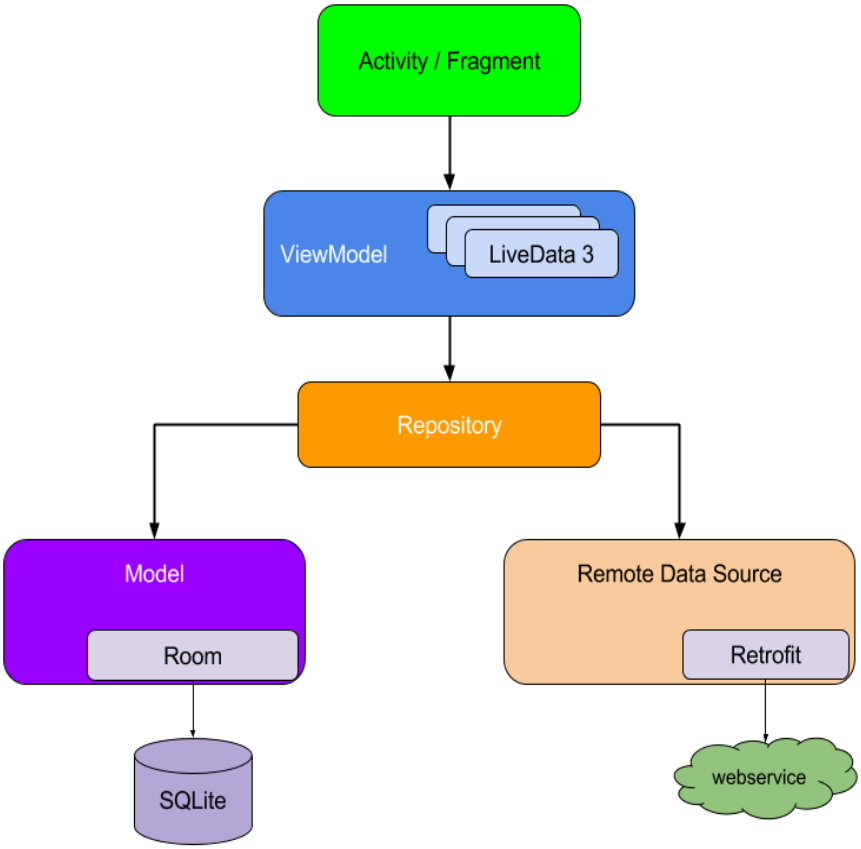
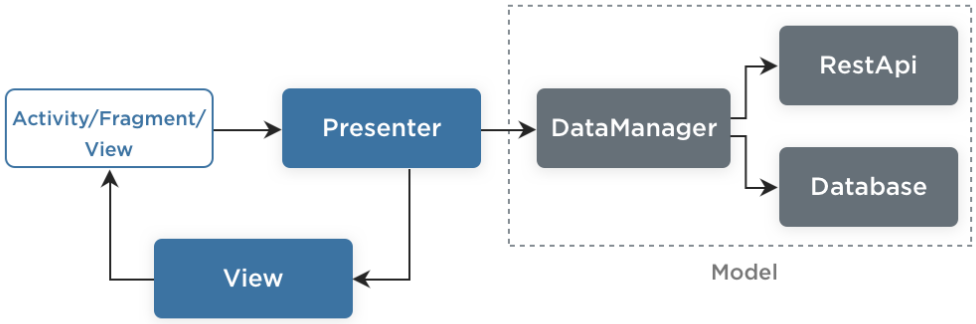


Рисунок 1. Приклад архітектури MVVM

Прикладом застосунка з архітектурним патерном MVP (рис. 2) стане комерційний продукт, мобільний застосунок MPG Rewards, який можна інсталювати з Google Play Market[7].



MVP in Android

Рисунок 2. Приклад архітектури MVP

Для застосунків сформоване технічне завдання: застосунок повинен бути клієнт-серверним, тобто мати можливість робити запити на сервер за допомогою фреймворку Retrofit та обробляти відповідь серверу за допомогою фреймворку GSON.

Кінцевий результат ViewModel та Repository зі застосунку з MVVM архітектурою:

```
class UserProfileViewModel @Inject constructor(
    savedStateHandle: SavedStateHandle,
    userRepository: UserRepository
) : ViewModel() {
    val userId : String = savedStateHandle["uid"] ?:
            throw IllegalArgumentException("missing user id")
    val user : LiveData<User> = userRepository.getUser(userId)
}

//Інформуємо Dagger, що цей клас повинен бути створений тільки один раз.
@Singleton
class UserRepository @Inject constructor(
    private val webservice: Webservice,
    // Простий кеш в пам'яті. Деталі опущені для стислості.
    private val userCache: UserCache
) {
    fun getUser(userId: String): LiveData<User> {
        val cached = userCache.get(userId)
        if (cached != null) {
            return cached
        }
        val data = MutableLiveData<User>()
        userCache.put(userId, data)
        // Ця реалізація все ще неоптимальна, але краще, ніж раніше.
        webservice.getUser(userId).enqueue(object : Callback<User> {
```

```
            override fun onResponse(call: Call<User>, response:
Response<User>){
                data.value = response.body()
            }
            // Випадок помилки опущений для стислості.
            override fun onFailure(call: Call<User>, t: Throwable) {
                TODO()
            }
        })
        return data
    }
}
```

Приклад базової реалізації архітектури MVP:

```
/**
 * Base class to work with any activities
 */
abstract class BaseActivity<T : BaseActivityPresenter> :
AppCompatActivity(),
    Progressable {
    abstract val presenter: T

    override fun attachBaseContext(newBase: Context) {
        super.attachBaseContext(newBase)
        presenter.updateLocale()
    }

    /**
     * handling onActivity result
     */
    override fun onActivityResult(requestCode: Int, resultCode: Int,
data: Intent?) {
        super.onActivityResult(requestCode, resultCode, data)
        presenter.onActivityResultHandler(requestCode, resultCode, data)
    }

/**
 * handling onRequestPermissionsResult
 */
override fun onRequestPermissionsResult(
    requestCode: Int,
    permissions: Array<out String>,
    grantResults: IntArray
) {
    super.onRequestPermissionsResult(requestCode, permissions,
grantResults)
    presenter.onPermissionResultHandler(requestCode, permissions,
grantResults)
```

```
}

abstract                    class                    BaseActivityPresenter(
    val           repository:           Repository<out           DataCash>
)                                                                          {
    abstract           val           activity:           BaseActivity<*>
    var onActivityResultHandler: (Int, Int, Intent?) -> Unit = { _, _, _
->                                                                         }
    var onPermissionResultHandler: (Int, Array<out String>, IntArray) ->
Unit        =        {        _,        _,        _        ->        }

    // class names of activities, should be initialized in proper
activity
    //         to         perform         showActivity()         actions
    var       loginActivityClassName:       Class<*>?       =       null
    var       mainActivityClassName:       Class<*>?       =       null

    fun                          showProgress()                        {
        activity.showProgress()
    }

    fun                          hideProgress()                        {
        activity.hideProgress()
    }

    /**
     *      showing      main      activity      clearing      backstack
     *   property   [mainActivityClassName]   should   be   inititalized
     */
    fun           showMainActivity(bundle:           Bundle?)           {
        showActivity(activity,  mainActivityClassName!!,  true,  bundle)
    }
}
```

Нижче наведено порівняння між архітектурами MVP та MVVM на основі таких функцій, як:

Логіка. У моделі MVVM клас коду ViewModel є застосунком, а його View - інтерфейсом між користувачем і програмою для цілей взаємодії. Тоді як у випадку MVP View є застосунком, а Presenter обробляє потік програми.

Введення даних. Процес у MVP ініціюється з View, а не з презентатора, але у випадку MVVM він починається з View, а не з ViewModel.

Ремонтопридатність. Легко додавати нові функції у ViewModel, якщо розробник програми має певний досвід роботи з конкретною бібліотекою.

Показники коду. MVP створює більше коду та класів, ніж архітектура ViewModel.

Також необхідно виділити наступні відмінності:

- Дані одержуються та маніпулюються зі стану даних, який присутній на рівні моделі. Ці дані передаються на рівень презентатора. У MVP немає взаємодії зі станом перегляду. MVVM має модель, розташовану в самій бізнес-логіці, де дані зберігаються у сховищі. Запити отримуються від моделі View, і дані розміщуються відповідно.

- Модель View в MVP має користувальницький інтерфейс, активність і фрагменти даних, а також взаємодіє з доповідачем. Модель View в MVVM взагалі не має бізнес-логіки і має лише користувацький інтерфейс.

- Дані з моделі подаються через Presenter, і він контролює поведінку в застосунку в архітектурі MVP. Він спрямовує шар View та керує взаємодією між моделлю та view. Дані також зберігаються в моделі. Рівень презентатора відсутній у MVVM, а спостережувані створюються в кожному компоненті інтерфейсу користувача.

- View у MVP не важливий або тісно пов’язаний із самим собою, щоб користувач міг легко ігнорувати шар перегляду. Шар View важливий, оскільки він створює міст між View та ViewModel в MVVM, так що неможливо не використовувати шар перегляду в MVVM.

- Шари View та Presenter у MVP можна використовувати багаторазово, тому цю архітектуру легко підтримувати. Крім того, його можна підтримувати за допомогою читабельних кодів, написаних мовою розмітки або іншою мовою кодування. У MVVM немає взаємодії між представленням і моделлю, і тому код керується одиницями. Це допомагає робити модульне тестування в MVVM.

Висновки

У статті проаналізовано два архітектурних патерни при розробці мобільних застосунків мовою Kotlin. Описано базові інтерфейси двох архітектур. Порівняльний аналіз був надан вище. З цієї статті можна зробити висновок, що архітектура MVVM є більш привабливою для великої кількості

розробників, тому що Google активно пропагує саме цю архітектуру через її переваги у розробці довготривалих та великих застосунків (більше 20 екранів), проте важче підтримувати та робити зміни, якщо розробник не має великою кількості досвіду роботи із цією архітектурою.

MVP є більш привабливим для застосунків, кількість екранів яких є менше двадцяти та за умови невеликої кількості досвіду розробника.

Якщо проаналізувати кількість викладених проектів у мережі Github з тегом MVVM та тегом MVP (рис. 3), то можна побачити, що саме перша ахітектура має більший попит серед розробників мобільних застосунків мовою Kotlin, яку Google офіційно визнав основною мовою для розробки застосунків під Android.

Рисунок 3 Кількість викладених у вільний доступ проектів з архітектурами MVVM та MVP відповідно

Проаналізувавши архітектури MVP та MVVM неможливо дати остаточну відповідь, яка з архітектур є найкращою. Кожна з архітектур залежить від багатьох умов, від навичок розробника до вимог клієнта. MVVM має допомогти нам позбутися деяких недоліків, які несе в собі MVP. З іншого боку, з'являться нові недоліки щодо MVVM, і ми повинні бути готові їх виправити, оскільки переваги добре розробленої архітектури завжди перевершують недоліки.

Список літератури

1. IDC: The Global Smartphone Market Grew 13.2% in the Q2 2021 Despite Supply Concerns and Vendor Shakeups [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://www.idc.com/getdoc.jsp?containerId=prUS48120021.

2. Загальна частка ринку мобільних ОС у продажах кінцевим споживачам з 1 кварталу 2012 року до 3-го кварталу 2021 року [Електронний ресурс] – Режим доступу до

ресурсу: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/.

3. Android Studio Release Notes [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://developer.android.com/studio/releases/index.html.

4. Building Projects with Native Code [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: http://facebook.github.io/react-native/docs/getting-started.html.

5. К.В. Харченко Методи та засоби розробки програмних додатків для операційної системи андроїд.// Збірник наукових праць. Серія: Нові рішення в сучасних технологіях. – Х.: НТУ «ХПІ». – 2014. - № 17. – С. 68-72.

6. Training Android Courses from Google Android Developers Notes [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://developer.android.com/courses.

7. MPG Rewards from Google Play Store [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=mt.think.micheleperesso&gl=UA.

8. Martin Fowler — GUI Architectures. Часть 2 [Електронний ресурс]. — 2009 — Режим доступу: https://habr.com/post/53536/.