Данілов Д. В., бакалавр

Київський національний університет технологій та дизайну

ДОСЛІДЖЕННЯ ОСНОВНИХ ШАБЛОНІВ MODEL VIEW CONTROLLER ТА MODEL VIEW VIEWMODEL ДЛЯ ПРОЕКТУВАННЯ ANDROID ДОДАТКІВ

Анотація. *В роботі досліджуються основні шаблони розробки інтерфейсів користувача для Android додатків Model View Controller (MVC) і Model View ViewModel (MVVM).*

Ключові слова: *Android; MVVM; MVC.*

Danilov D. V.

Kyiv National University of Technologies and Design

EXPLORING THE BASIC PATTERNS MODEL VIEW CONTROLLER AND MODEL VIEW VIEWMODEL FOR DESIGNING ANDROID APPLICATIONS

Absract. *The work examines the main patterns of user interface development for Android applications Model View Controller (MVC) and Model View ViewModel (MVVM).*

Keywords: *Android; MVVM; MVC.*

Вступ. на сьогоднішній день ми вже не уявляємо свого життя без смартфонів, до того ж згідно з статистичними даними Google Play App, опублікованими Statista, у 2021 році користувачі Google Play у всьому світі завантажили 111,3 мільярда мобільних додатків проти 76 мільярдів у 2018 році. Це означає що розробка додатків для телефонів стає з кожним роком все більш затребуваною. Існує багато різних підходів для проектування інтерфейсів користувача, кожен зі своїми можливостями та труднощами. Щоб визначити шаблон, який дає нам найкращі можливостя та переваги при розробці різних мобільних додатків необхідно дослідити ці шаблони в даному випадку шаблон Model View Controller (MVC) та Model View ViewModel (MVVM). Програмісти використовують дані підходи для того щоб зробити свій код більш зрозумілим для інших, а також для того щоб в майбутньому його було легко підтримувати та масштабувати. Основна мета шаблонів проектування:

- Надання повторно використовуваних каталогів компонентів для проектування програмних систем.
- Стандартизувати загальну термінологію серед розробників.
- Стандартизуйте спосіб виконання проекту.
- Полегшити навчання майбутніх розробників шляхом інтеграції наявних знань.

Пан Шуккур Шамлія в своїй статті «Розуміння основ шаблону проектування MVVM» стверджує, що «два розробника не можуть працювати одночасно над одним і тим же тому, що, зміни внесені одним розробником можуть порушити код іншого. Таким чином, мати все в одному місці – погана ідея з погляду стійкості, розширюваності та тестованості» [1], тому він пропонує використовувати шаблон проектування MVVM.

Постановка завдання: дослідити основні шаблони проектування користувацьких інтерфейсів Model View Controller (MVC) та Model View ViewModel (MVVM).

Результати досліджень. Основною задачею яка стоїть перед розробником програмного забезпечення, є створення простого для розуміння іншими програмістами коду, який досить просто підтримувати та маштабувати. Отже звичайний підхід коли програміст зберігає всю логіку певного користувацького інтерфейсу в одному місті призводить тільки до хаосу адже не зрозуміло як з цим працювати та підтримувати такий проект. Для того щоб такого не сталося треба використовувати шаблони для проектування користувацьких інтерфейсів наприклад Model View Controller (MVC) або

Model View ViewModel (MVVM) основною метою яких є розділити логіку та забезпечити читабельність коду.

Патерни проектування були актуальні у розробці програмного забезпечення з початку 1990-х років. Шаблони поділяються на:

- шаблони створення: вони абстрагують процес створення екземплярів об'єктів, також вони допомагають створювати незалежні системи створення, складання та подання об'єктів; у цій категорії у нас є: абстрактна фабрика, фабричный метод, прототип та сінглтон;
- структурні шаблони пов'язані з композицією класів або об'єктів, мають справу з тим, як класи та об'єкти використовуються для складання більших структур;
- шаблони поведінки характеризують спосіб взаємодії класів та об'єктів та як розподіляються між ними відповідальності.

Model View Controller (MVC) – це шаблон архітектури програмного забезпечення, який відокремлює дані програми та логіку уявлення від інтерфейсу користувача та модуля, що відповідає за обробку подій та обмін даними. Для цього MVC пропонує побудувати три компонента: моделі, уявлення та контролери, тобто з одного боку визначає компоненти подання інформації, з другого боку взаємодії з користувачем [5].

Цей шаблон проектування ґрунтується на ідеях повторного використання коду та поділу концепцій для полегшення розробки додатків та їх подальшого обслуговування.

Модель – це набір класів, які відповідають за подання інформації, з якою працює користувач. Ці класи можуть бути класами предметної області, DTO data transfer object (об'єкт передачі даних) або ViewModels (під ViewModel розуміється клас), в якому зберігаються дані, що представляють певне уявлення, а не ViewModel.

Подання делегуються для графічного представлення моделі та забезпечення операцій контролера, щоб користувач міг взаємодіяти з розробленою системою. У розробці мобільних додатків подання є Activity або Fragment з динамічним вмістом, над яким можна виконувати операції.

Контролер організовує взаємодію між уявленнями та моделлю. Він отримує запити користувачів, взаємодіє з моделлю, роблячи запити та модифікує їх, вирішує, яке подання буде відображатися як відповідь, і надає дані, необхідні для його рендерингу, або делегує відповідь іншій дії в іншому контролері.

Насправді, Android фреймворк на сьогоднішній день побудований так щоб надавати віджети здатні обробляти введені користувачем дані, це в свою чергу означає що роль контроллер зникла. Але все ще можливо застосувати шаблон MVC зображено на рис. 1 для розробки нативної програми під Android для цього потрібно:

- Activity та Fragment а також всі View повинні бути представленими в MVC.
- Контроллер та модель не повинні бути окремими класами, та наслідувати жодного класу з Android.
- Представлення та контролер залежать один від одного, але з метою тестування та дотримання принципу інверсії залежностей, представлення має реалізовувати інтерфейс, а контролер має посилатися на нього, а не на конкретний клас безпосередньо.

Model View ViewModel (MVVM) – це шаблон який є похідним від Model View Controller в ньому все розбито по певним шарам це зроблено щоб розподілити відповідальність по різним шарам зображено на рис. 2.

MVVM заснований на загальному механізмі прив'язки даних, який спрощує побудову шару подання окремо від решти шаблону шляхом винесення всього коду за межі шару подання.

ViewModel інкапсулює бізнес-логіку та дані. Бізнес-логіка визначається як будь-яка логіка програми, пов'язана з пошуком даних програми та керування ними, а також

забезпечує дотримання бізнес-правил, що гарантують узгодженість та достовірність даних. Щоб максимізувати можливості повторного використання, моделі не повинні містити жодних варіантів використання або певної поведінки користувача або логіки програми яка відповідає за отримання даних, або відображення даних користувачу для того що передати дані до Activity, Fragment або View використовують DVO (data visual object).

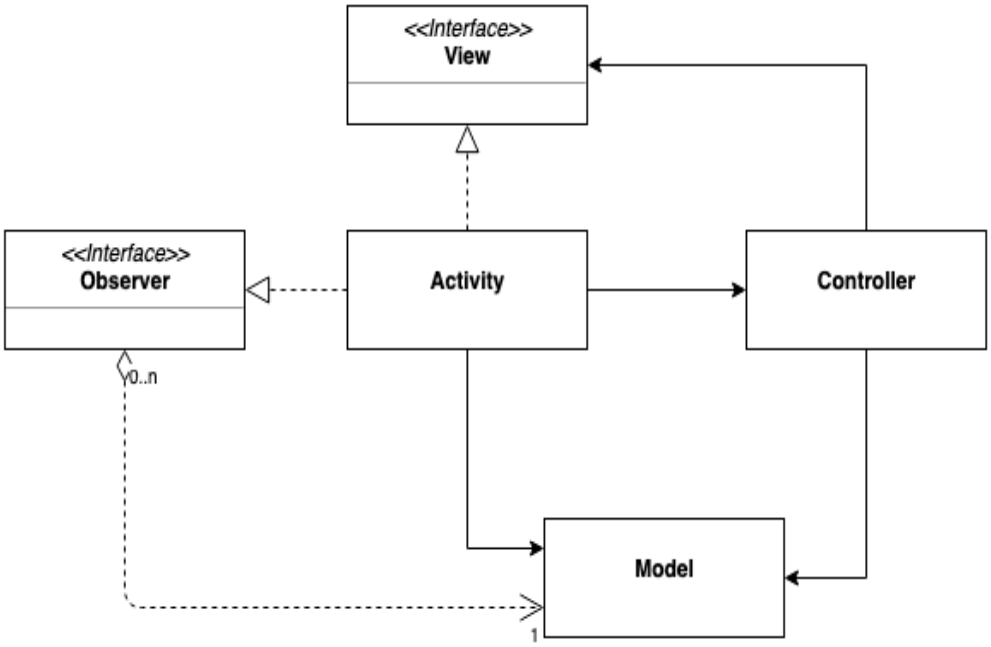

Рис. 1. **Діаграма класів реалізації MVC**

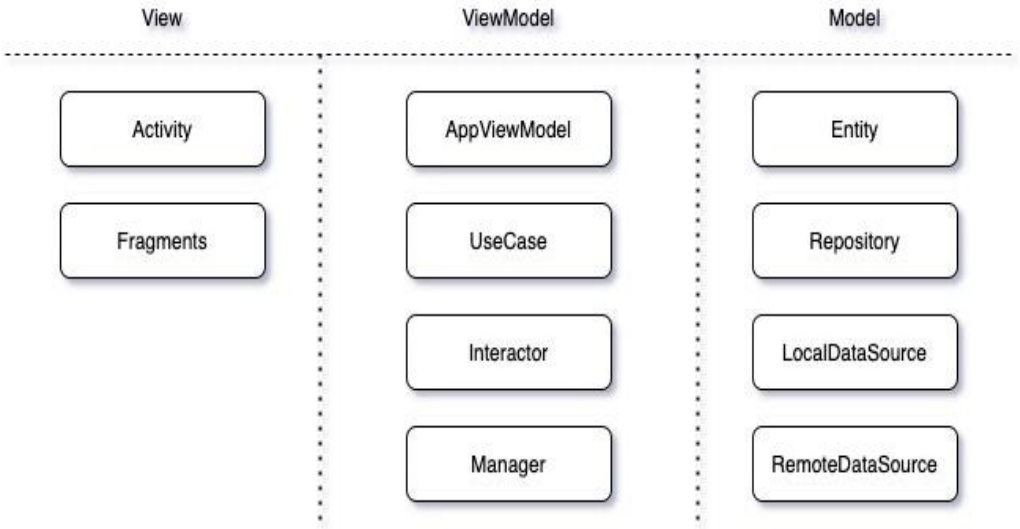

Рис. 2. **Шари MVVM та за що вони відповідальні в Android**

View інкапсулює інтерфейс користувача і логіку інтерфейсу користувача. Уявлення визначають конкретний інтерфейс користувача для частини програми. Зазвичай вони є елементами управління, які розроблені для правильної роботи при прив'язці до ViewModel.

Model інкапсулює логіку уявлення та стану. Логіка відображення визначається як логіка програми, яка має справу з варіантами використання програми (або користувальницький варіантами, завданнями користувача, робочим процесом і т. д.) і

визначає логічну поведінку та структуру застосування. Щоб максимізувати можливості повторного використання, ViewModel не повинна мати жодних посилань на певні класи користувача інтерфейс та елементи управління.

Зв'язок між всіма шарами MVVM в реалізації на Android дивитися рис. 3, є одностороннім тобто View знає про існування лиже ViewModel, а він в свою чергу знає тільки про Repository це означає, що всі шари максимально інкапсульовані.

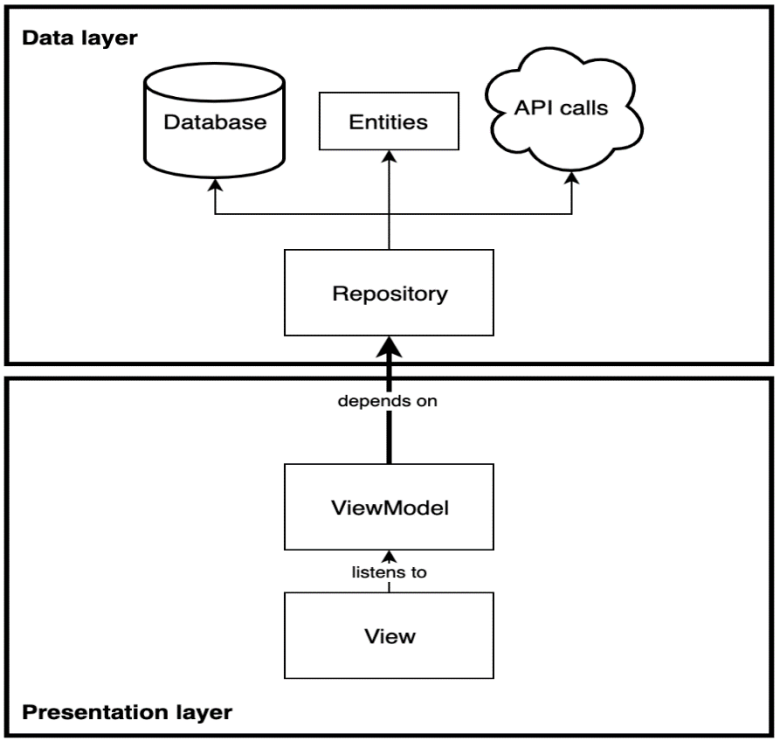

Рис. 3. **Модуляризація з MVVM в Android**

Висновки. Таким чином результати дослідження показують, що шаблон MVVM показує кращу продуктивність та покращує зручність використання та підвищує зрозумілість коду.

Список використаної літератури

1. The MVVM Pattern. *Microsoft.* URL: https://learn.microsoft.com/en-us/previous-versions/msp-n-p/hh848246(v=pandp.10)?redirectedfrom=MSDN (дата звернення: 15.11.2022).
2. Gossman, J. Tales from the Smart Client: Advantages and disadvantages of M-V-VM. *Microsoft.* URL: https://learn.microsoft.com/en-gb/archive/blogs/johngossman/advantages-and-disadvantages-of-m-v-vm (дата звернення: 15.11.2022).
3. Understanding the basics of MVVM design pattern. *Microsoft*. URL: https://learn.microsoft.com/en-gb/archive/blogs/msgulfcommunity/understanding-the-basics-of-mvvm-design-pattern (дата звернення: 15.11. 2022).
4. Fowler, M. Presentation Model. URL: https://martinfowler.com/eaaDev/PresentationModel.html (дата звернення: 15.11.2022).
5. Reenskaug, T., Coplien, J. O. The DCI Architecture: A New Vision of Object-Oriented Programming. URL: https://www.artima.com/articles/the-dci-architecture-a-new-vision-of-object-oriented-programming (дата звернення: 15.11.2022).
6. Model–view–viewmodel. *Wikipedia.* URL: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel (дата звернення: 15.11.2022).
7. Model–view–controller. *Wikipedia.* URL: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller (дата звернення: 15.11.2022).
8. Fowler, M. GUI Architectures. URL: https://martinfowler.com/eaaDev/uiArchs.html (дата звернення: 15.11.2022).
9. Popper, B. Google announces over 2 billion monthly active devices on Android. URL: https://www.theverge.com/2017/5/17/15654454/android-reaches-2-billion-monthly-active-users (дата звернення: 15.11.2022).