UDC 004.4

Бабюк Н. П.
Наконечний В. В.

MVI Architecture in Android

Вінницький національний технічний університет

Анотація

Стаття містить загальну інформацію про сучасний архітектурний стиль MVI та деталі його реалізації в Android розробці.

Ключові слова: MVI, архітектура, архітектурний стиль, Android розробка.

Abstract

The article contains general information about the modern MVI architecture pattern and its implementation details in Android development.

Keywords: MVI, architecture, architecture pattern, Android development.

Introduction

Architecture patterns play a crucial role in structuring and organizing code for Android applications. Their main purpose is to divide the code into certain layers or to separate concerns. There are several existing patterns with their advantages and disadvantages, for instance, MVC [1]. The MVC pattern is one of the oldest Android app architectures. It aims to separate concerns and provide a clear structure for code. MVC stands for Model View Controller where Model is responsible for data storage, domain logic, and communication with databases and networks. View handles the UI layer, visualizing data from the Model. The Controller contains core logic, responding to user behavior and updating the Model. But this pattern has disadvantages: code layers still depend on each other and it lacks a mechanism for handling UI logic explicitly. So later on more architectural patterns started to appear. The most widely-used pattern is MVVM (Model View Viewmodel) and it was the most modern one until MVI (Model View Intent) was founded. MVI provides a more rigid structure, emphasizing unidirectional data flow, immutability, and explicit Intent handling. While MVVM remains popular, MVI offers additional guarantees for maintainable and predictable Android app development.

About MVI

MVI [2], short for Model-View-Intent, is an architectural pattern that emphasizes unidirectional data flow and a reactive approach to building user interfaces. It draws inspiration from concepts like Redux in web development and combines them with the principles of reactive programming. As Android app development matured, developers encountered challenges related to managing complex UI states, handling user interactions, and maintaining codebases as applications scaled. Traditional architectures like Model-View-Controller (MVC) and Model-View-Presenter (MVP) struggled to address these challenges effectively. MVI emerged as an alternative that offered a more structured and predictable approach to handling state and UI updates. In MVI, the architecture is divided into three main components:

- Model: Represents the state of the application. It encapsulates all the data required for rendering the user interface.
- View: Represents the user interface. It observes the state changes emitted by the Model and renders the UI accordingly.
- Intent: Represents the user's intention or action. It captures user interactions such as button clicks, text input, etc., and converts them into immutable objects.

MVI follows a strict unidirectional data flow, ensuring that data flows in a single direction: from the Model to the View. This simplifies the flow of information, making it easier to reason about the application's behavior and maintain consistency. The state in MVI is immutable, meaning once created, it

cannot be modified. Any state change results in the creation of a new state object. This immutability ensures predictability and helps in managing the application state more effectively. MVI leverages reactive programming principles, typically using libraries like RxJava or Kotlin Coroutines with Flow. Observables are used to represent streams of data, allowing components to react to state changes reactively.

MVI has several advantages over other patterns:

- MVI encourages a clear separation between the Model, View, and Intent, making it easier to understand and maintain codebases.
- Since business logic is contained within the Model, it becomes easier to write unit tests for individual components. Testing becomes more predictable due to the unidirectional data flow.
- With an immutable state and strict data flow, managing the application state becomes more predictable, reducing the chances of bugs related to state inconsistency.
- MVI facilitates reactive UI updates, where the View reacts to changes in the Model's state. This leads to a smoother and more responsive user experience.

Implementation in Android

In other architectures such as MVP or MVVM, the definition of the Model typically refers to the data layer and the domain layer. This Model acts as a bridge between the application and remote data sources. In contrast, within the MVI pattern, the Model represents data but is structured as an immutable state. This implies that state modifications occur solely within the app's business logic, ensuring that alterations to the state are confined to a single location. Consequently, the business logic becomes the only source of truth responsible for generating the immutable Model [3].

The View reacts to state changes emitted by the ViewModel and updates the UI. In the ViewModel, process intents and update the application state accordingly. Emit the new state to be observed by the View. Capture user interactions (intents) in the View layer and pass them to the ViewModel.

Conclusion

The Model-View-Intent (MVI) architecture pattern offers a structured and efficient approach to building Android applications. By enforcing a unidirectional data flow and emphasizing reactive programming principles, MVI promotes maintainability, testability, and a better user experience. While it may require a learning curve for developers unfamiliar with reactive programming concepts, the benefits it brings to the table make it a compelling choice for modern Android app development.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

- https://www.geeksforgeeks.org/android-architecture-patterns/
- https://medium.com/swlh/mvi-architecture-with-android-fcde123e3c4a
- MVI Architecture with Android. The application lifespan is tied to its… | by Rim Gazzah | The Startup | Medium

Наконечний Влас Володимирович – студент групи 3ПІ-22б, Факультет інформаційних технологій та комп'ютерної інженерії, Вінницький національний технічний університет, м. Вінниця, Україна, e-mail: vlas.nak.05@gmail.com

Бабюк Наталя Петрівна – кандидат технічних наук, доцент кафедри програмного забезпечення, Вінницький національний технічний університет, м. Вінниця, e-mail: babiuk@vntu.edu.ua.

Nakonechnyi Vlas – student of group 3PI-22b, Faculty of Information Technology and Computer Engineering, Vinnytsia National Technical University, Vinnytsia, Ukraine, e-mail: vlas.nak.05@gmail.com

Babiuk Natalia – Ph. D., associate Professor at the Department of the Software Engineering, Vinnytsia National Technical University, Vinnytsia, e-mail: babiuk@vntu.edu.ua