MVVM : A Software Design Architecture

Course Code: CSE-404
Course Title: SOFTWARE ENGINEERING LABORATORY

Submitted by

MD. MAHFUZ MOLLA(395)

Computer Science and Engineering

Jahangirnagar University

Dhaka, Bangladesh

September 18, 2024

Contents

List of Figures iii

List of Tables iii

1 Introduction to MVVM 1

2 Purpose of MVVM 1

3 Components of MVVM 2

3.1 Model 2

3.2 View 3

3.3 ViewModel 3

4 Data Binding in MVVM 4

5 Advantages of MVVM 4

5.1 Separation of Concerns 4

5.2 Modular Development 4

5.3 Testability 5

5.4 Scalability 5

5.5 Data Binding 5

6 Disadvantages of MVVM 5

6.1 Learning Curve 5

6.2 Boilerplate Code 5

6.3 Overhead 6

7 MVVM in Android Development 6

8 Example of MVVM Workflow in Android 6

9 Conclusion 7

List of Figures

1.1 MVVM Architecture. 1

3.1 MVVM Design Architecture Diagram. 2

7.1 MVVM in Android. 6

8.1 MVVM Workflow Diagram. 7

List of Tables

List of Algorithms

1. Introduction to MVVM

The Model-View-ViewModel (MVVM) is a software architectural pattern designed to facilitate the separation of the user interface (UI) development from the business logic or back-end logic of an application. Initially introduced by Microsoft for the development of applications in WPF (Windows Presentation Foundation) and Silverlight, it has since gained popularity across various platforms, including Android, iOS, and Xamarin.

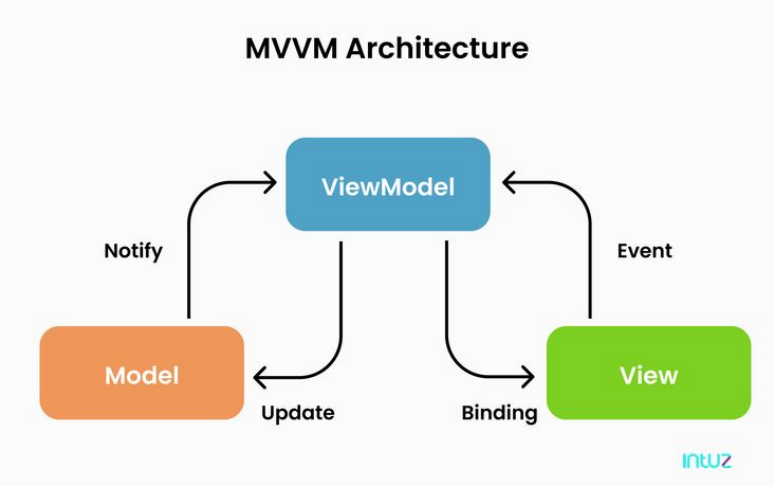

Figure 1.1: MVVM Architecture.

2. Purpose of MVVM

The primary goal of MVVM is to provide a clean separation of concerns by decoupling the view (UI) from the business logic. It promotes a modular structure, making the codebase easier to maintain, test, and scale. By leveraging data binding, it facilitates

real-time updates between the UI and underlying logic without manual intervention.

3. Components of MVVM

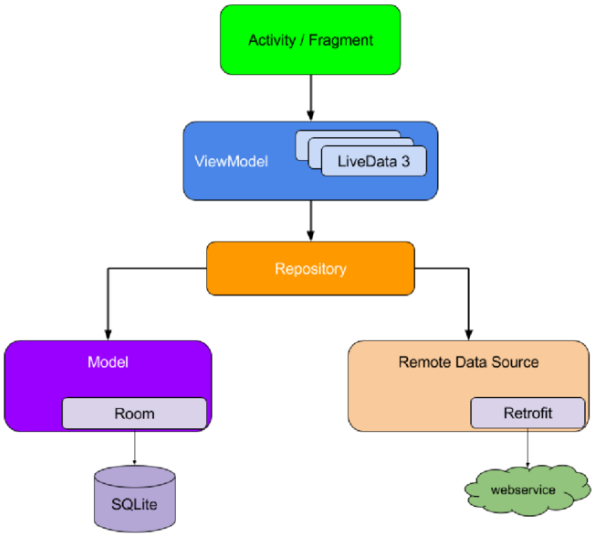

Figure 3.1: MVVM Design Architecture Diagram.

3.1 Model

The **Model** represents the application's data, logic, and rules. It encapsulates the core business logic, including data retrieval, validation, and manipulation. The Model is independent of the View and ViewModel and is responsible for communicating with databases, APIs, or other data sources.

Responsibilities of the Model:

- Manage the application's business logic.
- Define and handle data validation rules.
- Fetch and update data from external data sources (e.g., databases or APIs).

3.2 View

The **View** is the user interface layer of the application. It is responsible for presenting data to the user and sending user input back to the ViewModel. The View is tightly coupled with UI components such as layouts, buttons, text fields, and other visual elements.

In MVVM, the View is ”dumb,” meaning it has no direct knowledge of the business logic and interacts solely with the ViewModel through **data binding**. This separation allows developers and designers to work on the UI layer independently from the logic layer.

Responsibilities of the View:

- Display the data provided by the ViewModel.
- React to user input, such as clicks or gestures.
- Forward user actions and events to the ViewModel for processing.

3.3 ViewModel

The **ViewModel** acts as an intermediary between the View and the Model. It manages the data that is displayed by the View, processes user input from the View, and updates the Model as necessary. The ViewModel does not have a direct reference to the View, which allows for loose coupling between these layers.

Two-way data binding is typically used between the View and ViewModel. This means that when data in the ViewModel changes, the View automatically updates, and when the user interacts with the View (e.g., by entering text or selecting an option), those changes are automatically reflected in the ViewModel.

Responsibilities of the ViewModel:

- Expose data from the Model to the View in a format that the View can display.
- Handle user interactions passed from the View.
- Implement the logic to update the Model based on user input.

- Maintain state and manage business logic for the UI.

4. Data Binding in MVVM

One of the key concepts in MVVM is data binding. Data binding allows for automatic synchronization of data between the View and the ViewModel. For example, if the ViewModel updates a property, the change is automatically reflected in the View without additional code. Similarly, user input in the View is automatically propagated to the ViewModel.

In many frameworks, two-way data binding is available. This means:

- Changes in the Model are reflected in the View automatically.
- Changes in the View (such as user input) are automatically sent back to the ViewModel, where they can be processed.

5. Advantages of MVVM

5.1 Separation of Concerns

The UI is separated from the business logic, making the code easier to understand and maintain.

5.2 Modular Development

Since each component has a distinct role, developers can work on the View, ViewModel, and Model separately, improving collaboration and reducing dependencies.

5.3 Testability

The ViewModel can be easily tested without any dependency on the UI (View), enabling developers to write unit tests for the application's logic.

5.4 Scalability

As applications grow in complexity, the separation of concerns provided by MVVM ensures that the architecture remains organized, and new features can be added with minimal changes.

5.5 Data Binding

The automatic synchronization of data between the View and ViewModel reduces the need for boilerplate code, making development faster and reducing the chances of bugs.

6. Disadvantages of MVVM

6.1 Learning Curve

For developers new to MVVM, the separation of concerns and the concept of data binding may take time to learn and implement correctly.

6.2 Boilerplate Code

In some cases, MVVM can introduce additional boilerplate code, particularly when setting up data binding or handling complex ViewModel-View interactions.

6.3 Overhead

For smaller applications, the overhead introduced by the MVVM pattern may not always be necessary. It is often better suited for larger applications with complex UIs and multiple user interactions.

7. MVVM in Android Development

In Android, MVVM has become a popular architecture pattern, especially with the introduction of tools like LiveData, ViewModel, and DataBinding libraries provided by Google. Android's implementation of MVVM follows similar principles:

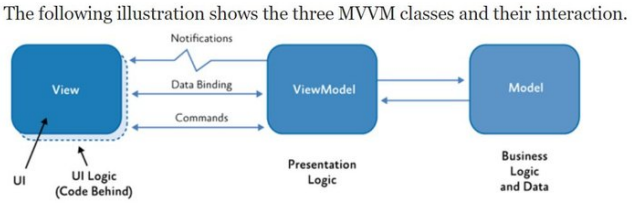

Figure 7.1: MVVM in Android.

- **Model**: Handles the data layer (e.g., using Room for local databases).
- **View**: Typically implemented using Activities or Fragments.
- **ViewModel**: Responsible for providing data to the View using LiveData or MutableLiveData and handling business logic.

8. Example of MVVM Workflow in Android

- **Model**: A repository class fetches data from a remote server or local database (Room).

- **ViewModel**: The ViewModel requests the data from the repository and processes it (if necessary). It exposes the data using LiveData.
- **View**: The Activity or Fragment observes changes in the LiveData object and automatically updates the UI when the data changes.

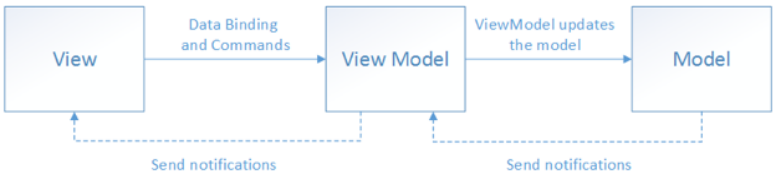

Figure 8.1: MVVM Workflow Diagram.

9. Conclusion

The Model-View-ViewModel (MVVM) architecture is an effective design pattern that enhances application development by clearly separating concerns among data management, user interface, and application logic. The Model handles data and business rules, the View manages the presentation and user interactions, and the ViewModel bridges the gap, ensuring a cohesive flow of data and logic. This separation promotes modularity, testability, and maintainability, making MVVM a valuable approach for developing scalable and robust applications. By adhering to MVVM principles, developers can achieve cleaner code and improved application performance.