

The MVVM Pattern Architecture for Project ki-kinbo!

Lamia Binta Latif

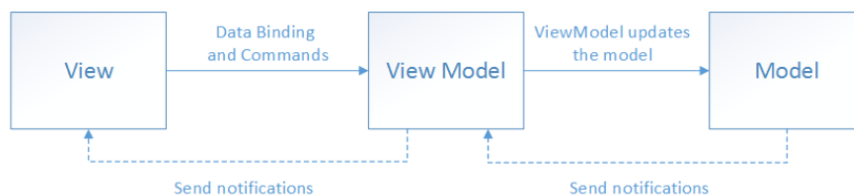
21 September 2024

1 Introduction

The MVVM pattern helps cleanly separate an application’s business and presentation logic from its user interface (UI). Maintaining a clean separation between application logic and the UI helps address numerous development issues and makes an application easier to test, maintain, and evolve. It can also significantly improve code reuse opportunities and allows developers and UI designers to collaborate more easily when developing their respective parts of an app.

2 The MVVM Pattern

There are three core components in the MVVM pattern: the **Model**, the **View**, and the **ViewModel**. Each serves a distinct purpose. The diagram below shows the relationships between the three components.



In addition to understanding the responsibilities of each component, it’s also important to understand how they interact. At a high level, the view “knows about” the view model, and the view model “knows about” the model, but the model is unaware of the view model, and the view model is unaware of the view. Therefore, the view model isolates the view from the model and allows the model to evolve independently of the view.

The benefits of using the MVVM pattern are as follows:

- If an existing model implementation encapsulates existing business logic, it can be difficult or risky to change it. In this scenario, the view model

acts as an adapter for the model classes and prevents you from making major changes to the model code.

- Developers can create unit tests for the view model and the model, without using the view. The unit tests for the view model can exercise exactly the same functionality as used by the view.
- The app UI can be redesigned without touching the view model and model code, provided that the view is implemented entirely in XAML or C#. Therefore, a new version of the view should work with the existing view model.
- Designers and developers can work independently and concurrently on their components during development. Designers can focus on the view, while developers can work on the view model and model components.

The key to using MVVM effectively lies in understanding how to factor app code into the correct classes and how the classes interact. The following sections discuss the responsibilities of each of the classes in the MVVM pattern.

3 View

The view is responsible for defining the structure, layout, and appearance of what the user sees on screen. Ideally, each view is defined in XAML, with a limited code-behind that does not contain business logic. However, in some cases, the code-behind might contain UI logic that implements visual behavior that is difficult to express in XAML, such as animations.

4 ViewModel

The view model implements properties and commands to which the view can data-bind and notifies the view of any state changes through change notification events. The properties and commands that the view model provides define the functionality to be offered by the UI, but the view determines how that functionality is to be displayed.

5 Model

Model classes are non-visual classes that encapsulate the app's data. Therefore, the model can be thought of as representing the app's domain model, which usually includes a data model along with business and validation logic. Examples of model objects include data transfer objects (DTOs), Plain Old CLR Objects (POCOs), and generated entity and proxy objects.

Model classes are typically used in conjunction with services or repositories that encapsulate data access and caching.

6 MVVM Frameworks

The MVVM pattern is well established in .NET, and the community has created many frameworks which help ease this development. Each framework provides a different set of features, but it is standard for them to provide a common view model with an implementation of the `INotifyPropertyChanged` interface. Additional features of MVVM frameworks include custom commands, navigation helpers, dependency injection/service locator components, and UI platform integration. While it is not necessary to use these frameworks, they can speed up and standardize your development. The eShop multi-platform app uses the .NET Community MVVM Toolkit. When choosing a framework, you should consider your application's needs and your team's strengths. Below are some common MVVM frameworks for .NET MAUI:

- .NET Community MVVM Toolkit
- ReactiveUI
- Prism Library

7 Benefits of MVVM in Android Development

1. **Separation of Concerns:** MVVM promotes a clear separation of concerns, where each component has a specific role and responsibility. This separation enhances code readability, maintainability, and reusability.
2. **Modularity:** MVVM facilitates modularity by breaking down the application into smaller, independent components. Each component can be developed, tested, and maintained separately, allowing for easier collaboration among team members.
3. **Testability:** MVVM simplifies unit testing and automation testing. The ViewModel, being independent of the View, can be tested thoroughly without any UI dependencies. This enables developers to write test cases for business logic and data manipulation, leading to more robust and bug-free applications.
4. **Data Binding and Observability:** MVVM utilizes data binding techniques to establish a connection between the View and the ViewModel. This allows automatic updating of the UI whenever the underlying data changes. The use of observability patterns, such as LiveData or RxJava, ensures that the View reflects the latest state of the data.

8 Implementing MVVM in Android

To illustrate the implementation of MVVM in Android, let's consider a simple task management application.

1. **Define the Model:** Create classes that represent tasks, repositories for data management, and APIs for data retrieval.
2. **Develop the ViewModel:** Create a ViewModel that fetches tasks from the repository and exposes them to the View. Implement methods for adding, deleting, or updating tasks.
3. **Create the View:** Design the user interface using XML layouts and bind the UI components with the ViewModel using data binding techniques. Observe the ViewModel's data to reflect changes in the UI automatically.
4. **Integrate Data Binding:** Utilize data binding expressions to bind data from the ViewModel to the UI elements. This reduces boilerplate code and ensures a consistent flow of data between the View and the ViewModel.
5. **Testability and Unit Testing:** Write unit tests for the ViewModel to validate the business logic and data manipulation. Since the ViewModel is decoupled from the View, testing becomes easier and more focused.

9 Conclusion

The Model-View-ViewModel (MVVM) architecture pattern has revolutionized Android app development by providing a clear separation of concerns, increased modularity, and improved testability. MVVM allows developers to create applications that are maintainable, scalable, and highly testable. Use MVVM in your Android projects to unlock the potential for developing robust and user-friendly applications.