COMPUTER SCIENCE AND DATA ANALYTICS

INFORMATION RETRIEVAL

Midterm Report

Title: Comparative Analysis of Mobile Application Architectures

Student: Fidan Hasanguliyeva

Instructors: Prof. Dr. Stephen Kaisler, Assoc. Prof Dr. Jamaladdin Hasanov

Problem Description

In the rapidly evolving landscape of mobile application development, selecting the right architectural approach is critical to ensuring the success and sustainability of a mobile app. Various architectural patterns and frameworks have emerged over the years, each with its own set of advantages and limitations. Therefore, conducting a comparative analysis of different mobile application architectures becomes essential for developers and businesses to make informed decisions and design robust, efficient, and scalable mobile apps.

The goal of this project is to conduct a comprehensive comparative analysis of various mobile application architectures, considering factors such as performance, scalability, maintainability, security, development complexity, and user experience. The choice of a suitable architectural paradigm is crucial for the success of mobile application development. Different architectural paradigms, such as Model-View-Controller (MVC), Model-View-ViewModel (MVVM), and Clean Architecture, provide guidelines and patterns for organizing code, separating concerns, and creating scalable and maintainable applications. However, the implementation of these architectures can vary depending on factors such as the platform, programming language, and frameworks used. The project's scope will encompass a thorough investigation and comparison of these architectural patterns based on a set of predefined metrics. The metrics may include application startup time, memory consumption, responsiveness, code maintainability, ease of development, reusability, and adaptability to changing requirements. The analysis will be conducted through the development and evaluation of prototype applications using each architectural pattern. Real-world use cases will be simulated, and extensive performance testing will be carried out to assess the strengths and weaknesses of each approach.

Strategy Definition

The comparison in this project will be both qualitative and quantitative.

- Qualitative Comparison: This aspect of the analysis will involve assessing the different architectural patterns based on subjective factors, such as development complexity, maintainability, flexibility, ease of understanding, and developer experience. Evaluating these aspects may involve conducting code reviews, and considering best practices in software engineering.
- Quantitative Comparison: On the other hand, the quantitative comparison will involve measuring and analyzing objective metrics and performance indicators. These metrics may include application startup time, memory usage, response times, and other performance-related aspects. To quantify these metrics, the project will conduct rigorous testing, profiling of the prototype applications developed using each architectural pattern.

By combining both qualitative and quantitative approaches, the project aims to provide a comprehensive and balanced analysis of the mobile application architectures. This will allow for a more holistic understanding of the strengths and weaknesses of each approach, enabling developers and businesses to make well-informed decisions when selecting the most suitable architecture for their specific use cases and project requirements.

Based on the assessment results, this research will propose a hybrid approach that combines the most beneficial aspects of each architectural paradigm. The hybrid approach will be designed to address the limitations of individual paradigms and provide a more efficient and maintainable codebase for mobile applications. This will involve customizing the architectural components and adapting them to fit the specific requirements of the project.

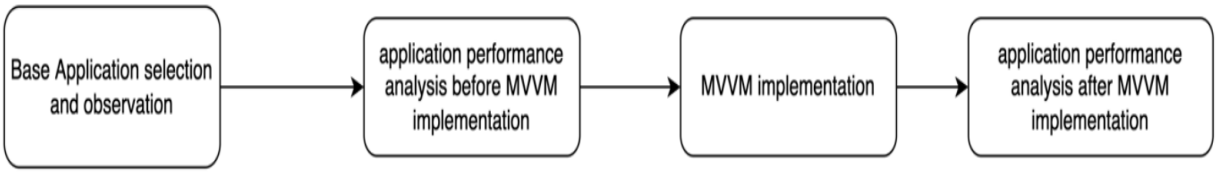


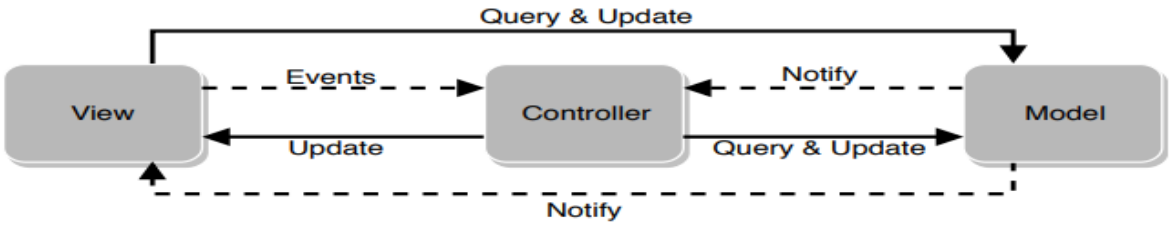
Figure 1. Research Stages

Progress Overview:

The development of maintainable and high-quality interactive software requires a clear separation between the underlying data, its presentation and manipulation. The MVC-based architectural design patterns are thus widely used in such type of software. There are however different perspectives leading to different MVCbased patterns.

1. Model View Controller : Proposed in 1979, MVC aims at separating the business logic from the presentation layer [31] [8]. It defines three main layers.

- Model: basically a set of entities that represent knowledge, information and a set of rules that handle updates and access to these information. To ensure loose coupling with the other layers, the Model should not know about the other components but can send them notifications about the state of its objects
- View, which is in charge of providing a visual representation of the Model. The View shows data that it retrieves from the Model by sending queries, and can change its state through messages.
- Controller, which is the main entry point of the application, manipulates the View and translates the interaction of the user to the model. Furthermore, the Controller communicates directly with the Model and must be able to change its state . Figure shows the three components of the original MVC architecture and describes the connections between them.



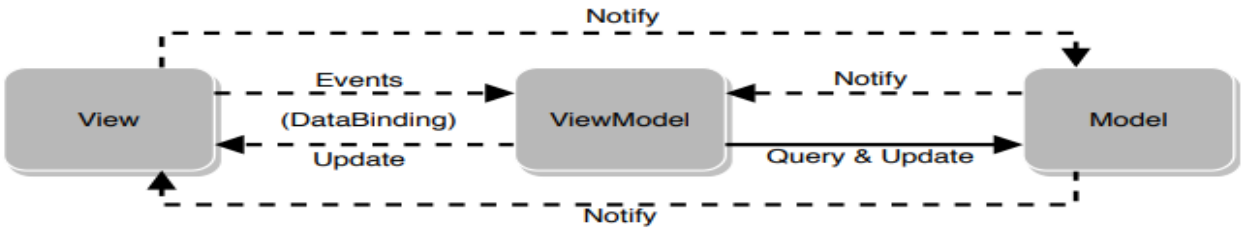
(a) Classical MVC

2. Model View Presenter. MVP is a variant of the MVC architecture, introduced in the early 90s. It brings more decoupling between the domain layer and the user interface. Compared to MVC, the Presenter in MVP is more isolated - than the Controller in MVC

- from the underlying system and should know nothing about the system. The Model has essentially the same responsibilities whereas the View handles actions that manipulate components of the system or the user interface and only delegates to the Presenter actions that manipulate the model entities. If the Presenter needs to make visual changes to the user interface or make interactions with the underlying operating system, it should call a method of the View that implements this need.

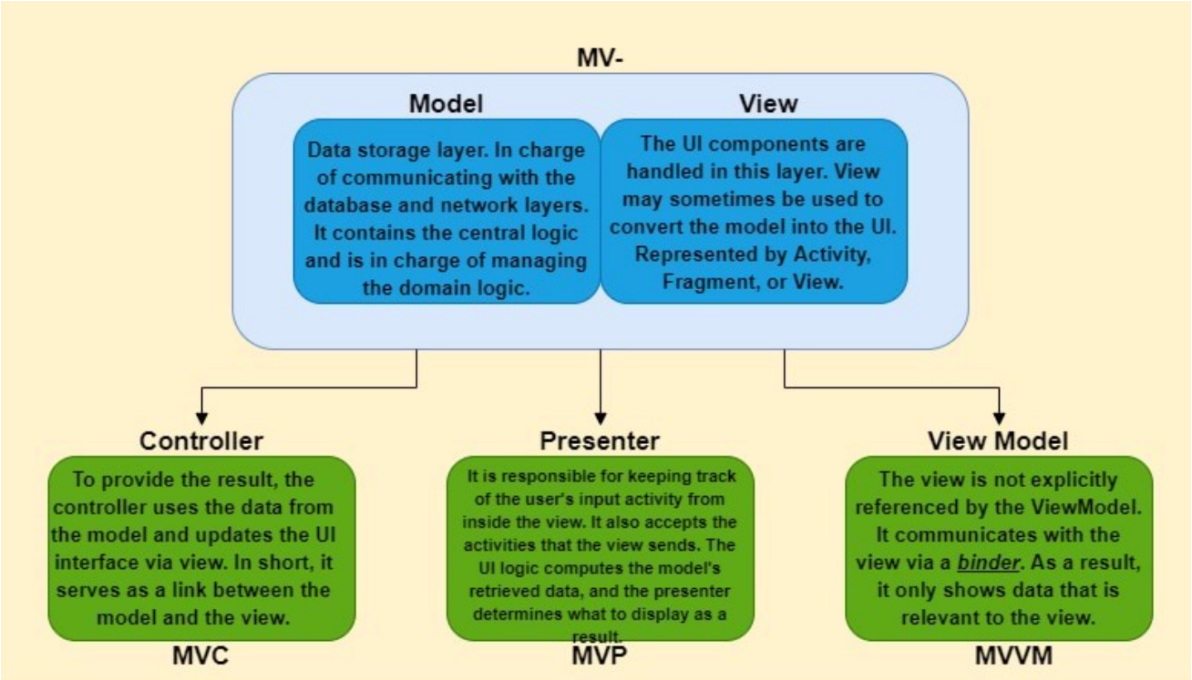
(b) MVP

3. **Model View ViewModel**: MVVM is an evolution of the MVP pattern. As shown in Figure 1 (c), it aims to realise more decoupling between the user interface and the middle layer represented by the ViewModel. The communication between these two is done exclusively through two-way data-binding notifications. The View, generally written in declarative languages, usually XML variants, represents the user interface. It is as passive as possible; declaring only widgets and user interface objects, and delegating all the logic to its backing ViewModel, from which it retrieves any kind of updates. The ViewModel is simply a Model of the View. It defines and implements the user input events, transforms the Model data and makes it ready to be displayed by the View. Importantly, the ViewModel should be completely ignorant about its corresponding View. No reference to a View should be declared nor used within the ViewModel. The ViewModel should be ready at anytime to be used with any other View entity without breaking code. The role of Model entities remains unchanged compared to the two other patterns.



(c) MVVM

Since the above architectural designs are based around the idea of collaborating with their own structure to achieve a positive outcome, each component has its own set of responsibilities. Model and View are shared by all of the patterns mentioned above. The third aspect is what distinguishes them from one another. As a result, here is a simplified mind map.



Pattern	Dependency on Android API	Unit Testability	Follow Modular and single responsibility principle
MVC	High	Difficult	No
MVP	Low	Good	Yes
MVVM	Low or No dependency	Best	Yes

Conclusion

I've read papers and articles to understand how mobile app architectures work. It's important to choose the right architecture to make sure apps work well and are easy to maintain. I've also figured out the main things to look at when comparing these

architectures like how the code is organized, how easy it is to test, and how well it can handle lots of users. By looking at these factors, I'll be able to compare the different architectures and see which ones are the best for different situations. I plan to take my research a step further by developing a simple mobile app that showcases the differences between various design patterns in a visual and practical manner. This approach will allow me to demonstrate the impact of different architectural paradigms on the app's structure, behavior, and user experience. Each version of the app will be implemented using a different design pattern, allowing users to switch between the patterns and observe the resulting changes in the app's behavior and structure.