Comparative Analysis of Mobile Application Architectures
Fidan Hasanguliyeva

This report delves into the Model-View-ViewModel (MVVM) design pattern, its implementation, and its impact on application performance. A simple application was developed to compare MVVM's startup time, memory usage, and CPU utilization. This evaluation aims to provide insights into MVVM's suitability for building efficient and maintainable software. Having emerged after MVC and MVP, the MVVM pattern is like an upgraded version of these two. The view and the model act the same as in both previous cases, but the view model becomes the intermediary. MVVM aims to keep the UI code simple, making it easy to manage.

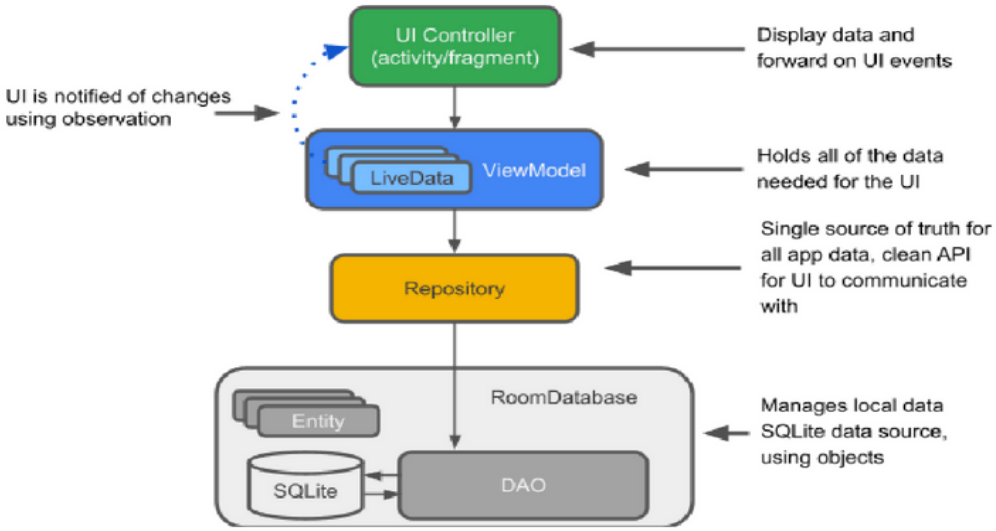
1. **View**
   The view component includes both the activity class and the fragment that is responsible for the display. The View will observe the data in the ViewModel component and make updates when the data changes.

2. **ViewModel**
   Viewmodel is a component that represents how a view interacts with the data in the Model, so that the view can represent and process the data contained in the Model class. In this case ViewModel acts as a component that stores data in the form of livedata so that the view component can observe changes that occur in existing data. The viewmodel component also communicates with the data repository.

3. **Model**
   The model components will be directly related to the data to be used in the application. The repository will be responsible for the data needed by the application. Data will be obtained in two ways, namely by using restAPI remotely with the help of a retrofit library and with a Room database in the form of local data.

   The MVVM pattern supports bidirectional databinding between the View and the View Model, and there is many to one relationship between the View and the View Model. In android development, Android MVVM Helper library is a set of basic classes for convenient working withMVVM. This list includes classes for workingwith Activity (Binding Activity and ActivityViewModel), and with Fragment (BindingFragment and Fragment ViewModel), whichalready have binding logic, and the necessary methods for callbacks are also defined. The benefit of the MVVM over MVP is that the Presentation layer is completely view independent which makes it simpler

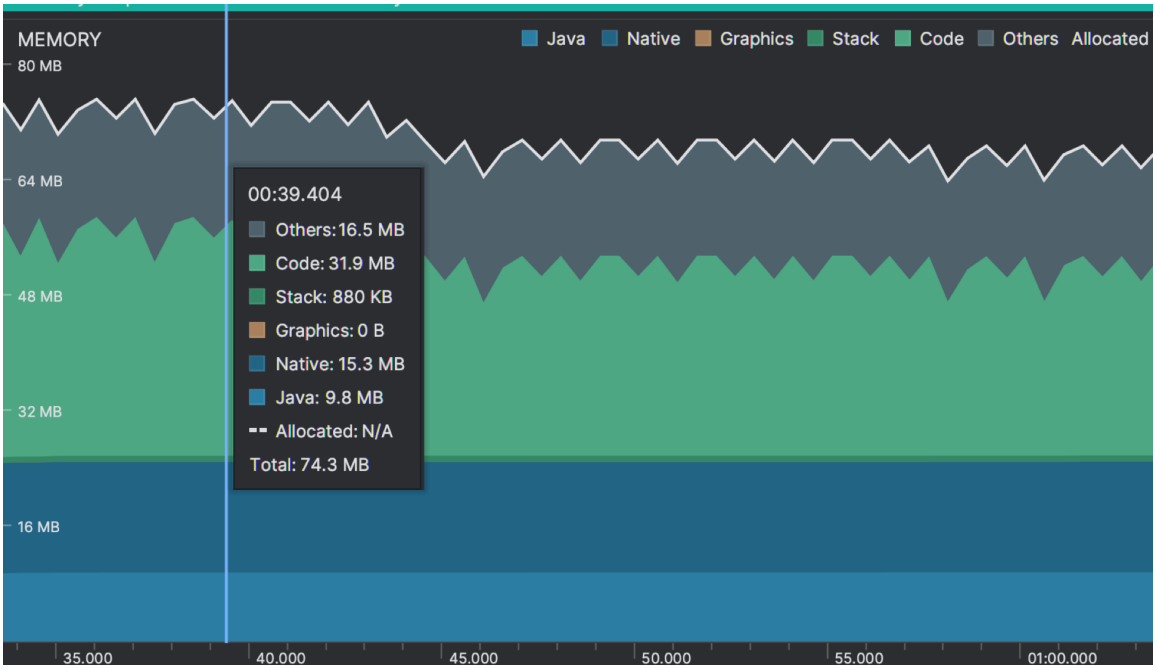
   The assessment was conducted using the Android Profiler tool within Android Studio. This tool facilitated the measurement of CPU usage, with a focus on displaying the percentage of total available CPU time consumed. Additionally, for memory usage measurement, the Android Profiler provided insights into the memory consumption of various memory categories, indicating the amount of memory utilized by each.



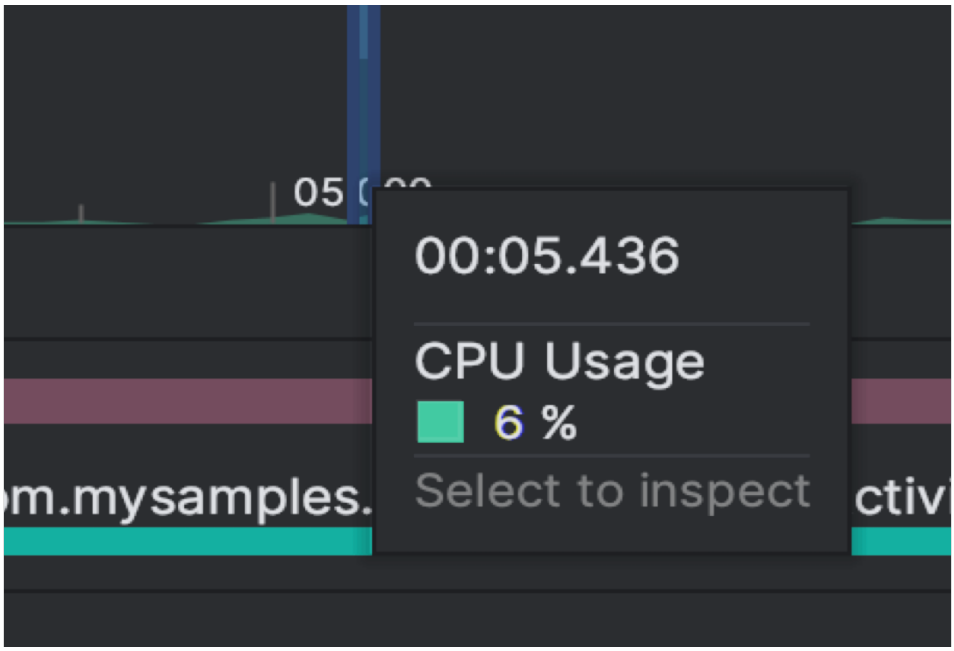
App startup time

```
Displayed com.mysamples.mvc/com.mysamples.mvvm.ui.view.MainActivity: +2s43ms
```

Memory usage:



CPU Usage



The results of the evaluation indicate that the MVVM (Model-View-ViewModel) pattern exhibits similar memory usage compared to both the MVC (Model-View-Controller) and MVP (Model-View-Presenter) patterns. However, MVVM demonstrates lower CPU usage and faster startup times than the other two patterns. This outcome can be explained by the fundamental principles and mechanics of the MVVM pattern.

Memory Usage: The MVVM pattern, like MVC and MVP, maintains a clear separation of concerns, ensuring that each component has a specific responsibility. In terms of memory usage, all three patterns generally allocate memory for the model (data), view (UI elements), and the intermediate layer that handles communication between the model and view.

However, the comparable memory usage observed between the patterns can be attributed to the fact that the memory overhead introduced by the ViewModel in MVVM is balanced by the efficient management of data states. The ViewModel is designed to store and manage the presentation logic and data state of the view, which can help reduce unnecessary data duplication and therefore maintain a relatively consistent memory footprint.

CPU Usage: MVVM's advantage in terms of lower CPU usage is primarily due to its data-binding mechanism and the unidirectional flow of data. In MVVM, the ViewModel holds the application's data state and communicates changes to the view via data-binding. This

reduces the need for frequent and direct interactions between the view and the model, resulting in fewer updates and less CPU-intensive operations during rendering and user interactions.
In contrast, both MVC and MVP often involve more direct communication between the view and the controller/presenter. This can lead to more frequent updates, increased interactions, and subsequently higher CPU usage as the application's complexity grows.

Startup Time: The MVVM pattern's advantage in startup time is also related to its unidirectional data flow and the efficient separation of concerns. During the application's initialization, the ViewModel can prepare the data state that the view requires. This allows for a streamlined initialization process, as the view can be constructed with the necessary data already available.

In contrast, MVC and MVP patterns might involve more complex interactions between the model, view, and controller/presenter during initialization, potentially leading to slightly slower startup times.