Comparative Analysis of Mobile Application Architectures

CSCI6917: Guided Research Methods
Fidan Hasanguliyeva
08.08.2023

Project Objective

- 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.

- Conducting a comparative analysis of different mobile application architectures becomes essential for developers and businesses to make informed decisions and design efficient, and scalable mobile apps.

Heilmeier Questions

- **What are you trying to do?**
  I tried to conduct a comparative analysis of three popular mobile application architectures: Model-View-Controller (MVC), Model-View-Presenter (MVP), and Model-View-ViewModel (MVVM).

- **How is it done today?**
  The analysis focuses on understanding the strengths and weaknesses of each architecture, startup time, memory usage, and other performance-related aspects.

- What is new in your approach and why do you think it will be successful?
  Rather than focusing on a single aspect, I aimed to create a holistic view of each architecture's impact on mobile application development.

- **Who cares?**
  Developers and Technical Teams, Architects and Designers, Project Managers,

- **What are the risks?**
  Limited Scope, Ignoring Developer Preferences, Biased Analysis, Inaccurate Information

- **What is new in your approach and why do you think it will be successful?**
  Rather than focusing on a single aspect, I aimed to create a holistic view of each architecture's impact on mobile application development.

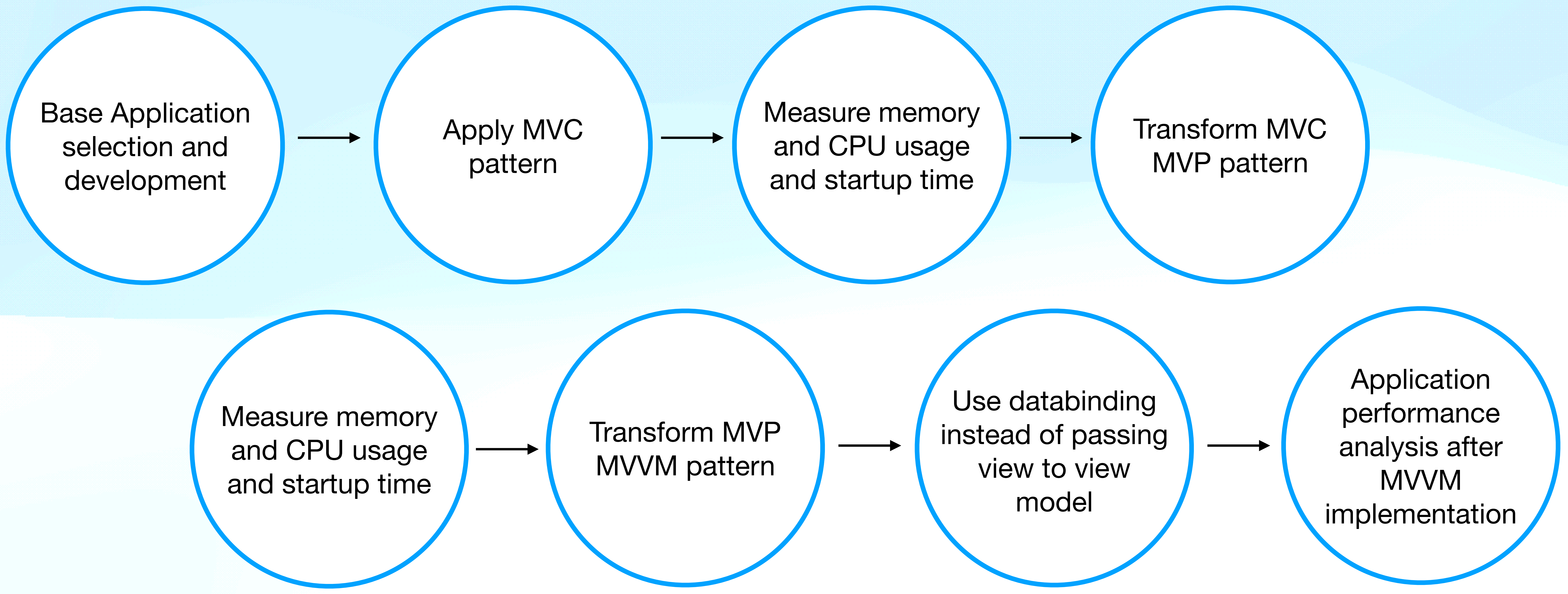
- **How much did it cost?**
  No out of pocket cost

- **How long did it take?**
  About 2 months

- **What are the mid-term and final "exams" to check for success?**
  Midterm: development of sample app in MVC and MVP, analysis of these applications; **Final:** development of sample app in MVVM and comparing 3 architectures;

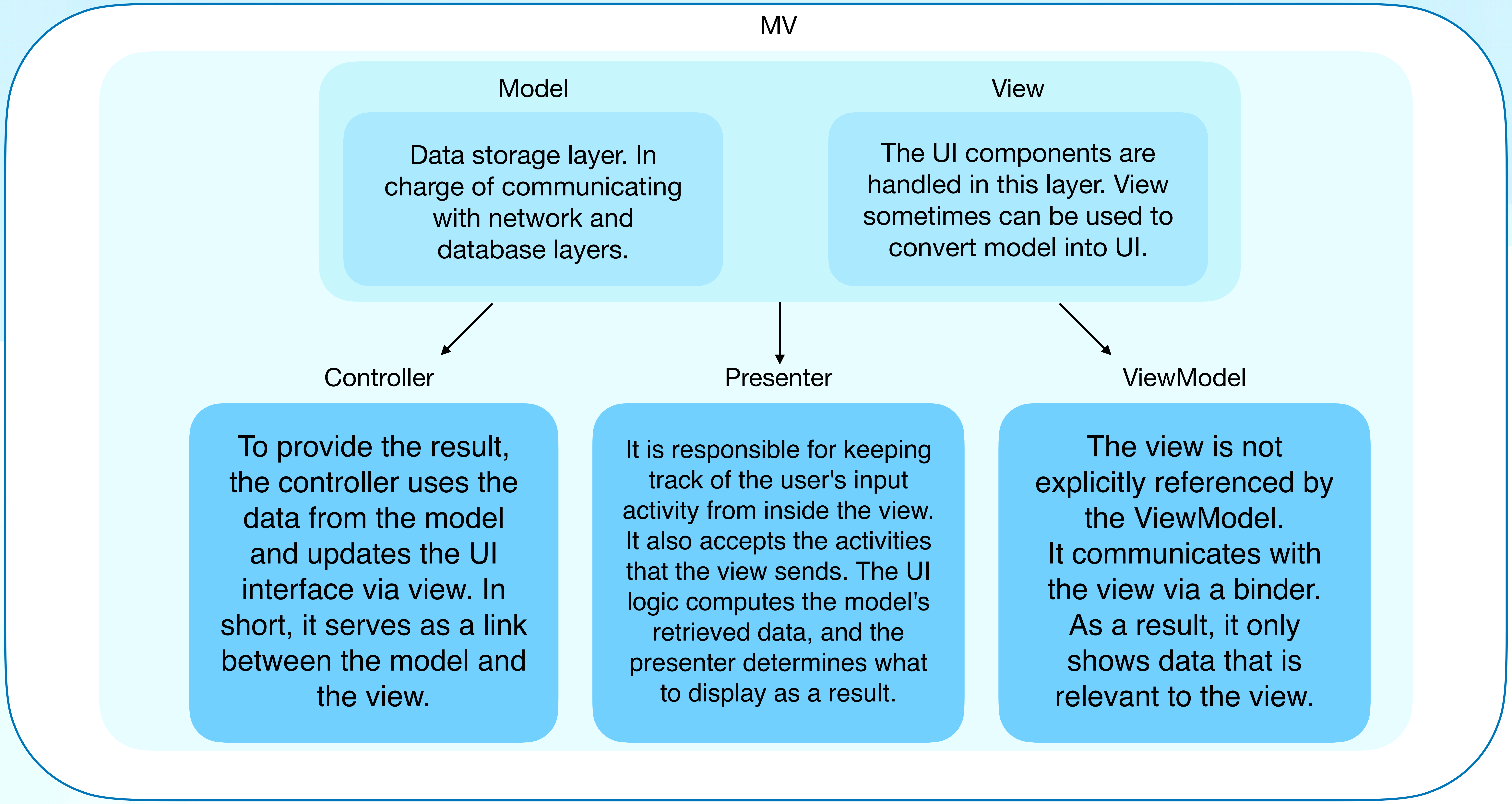
Technical Approach

Key steps



Base Application selection and development
Apply MVC pattern
Measure memory and CPU usage and startup time
Transform MVC MVP pattern
Measure memory and CPU usage and startup time
Transform MVP MVVM pattern
Use databinding instead of passing view to view model
Application performance analysis after MVVM implementation

Technical Approach



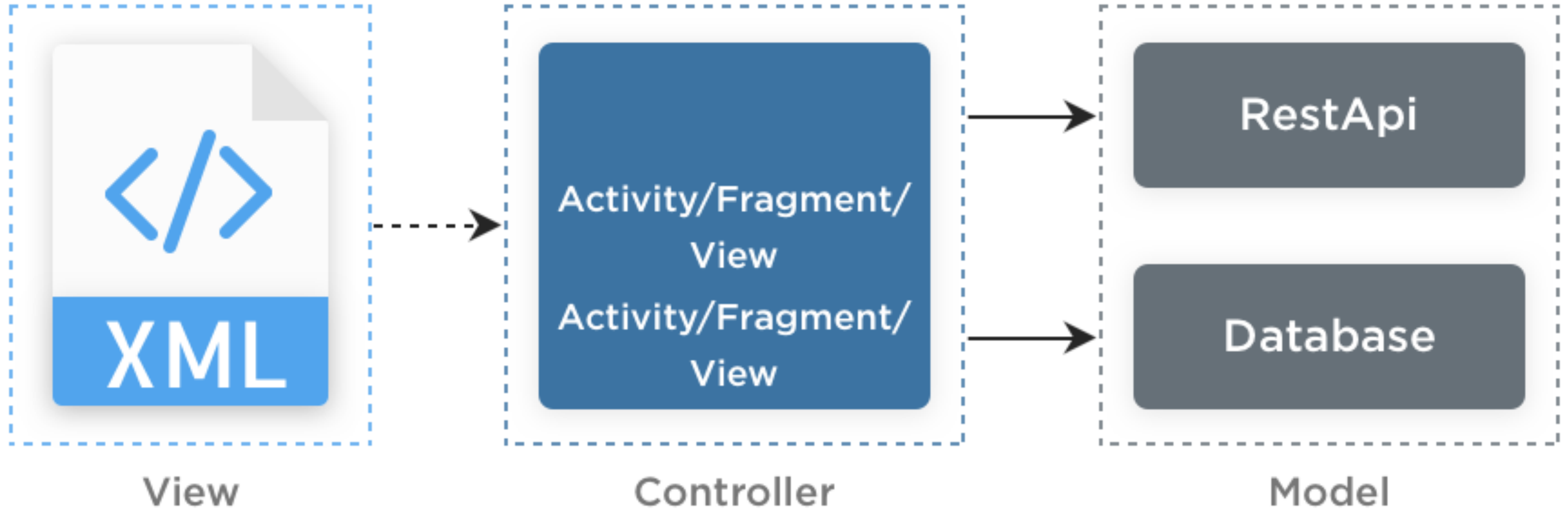
Data storage layer. In charge of communicating with network and database layers.

The UI components are handled in this layer. View sometimes can be used to convert model into UI.

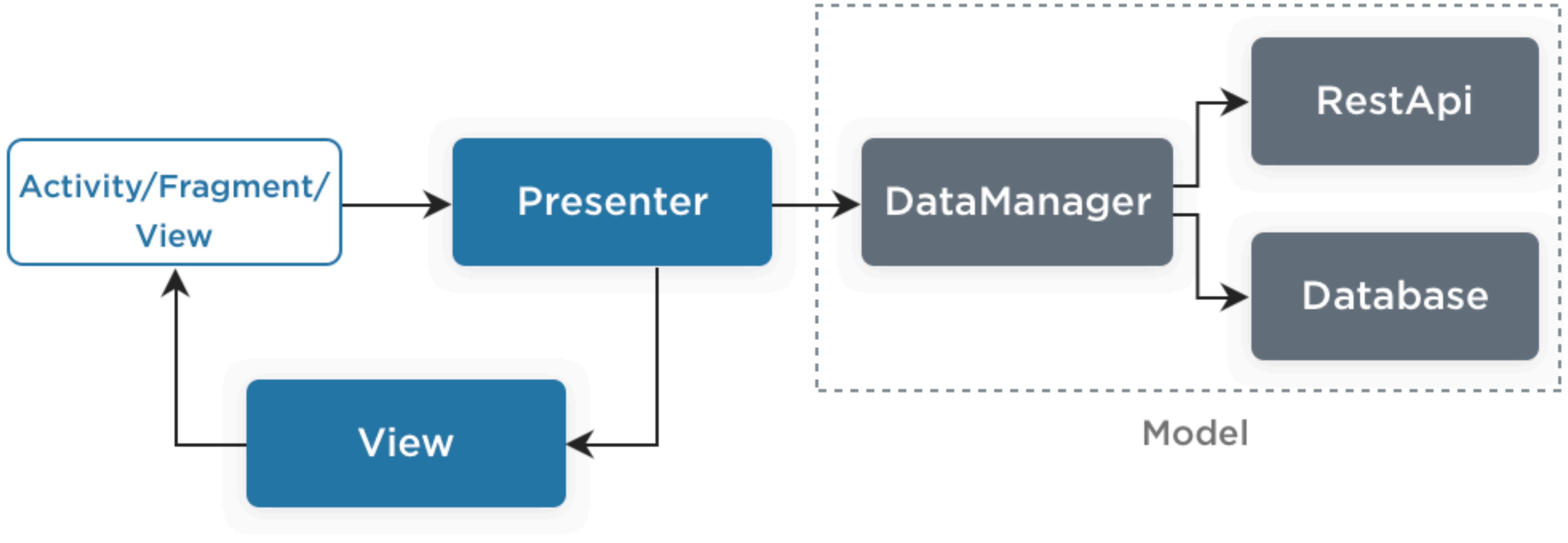
To provide the result, the controller uses the data from the model and updates the UI interface via view. In short, it serves as a link between the model and the view.

It is responsible for keeping track of the user's input activity from inside the view. It also accepts the activities that the view sends. The UI logic computes the model's retrieved data, and the presenter determines what to display as a result.

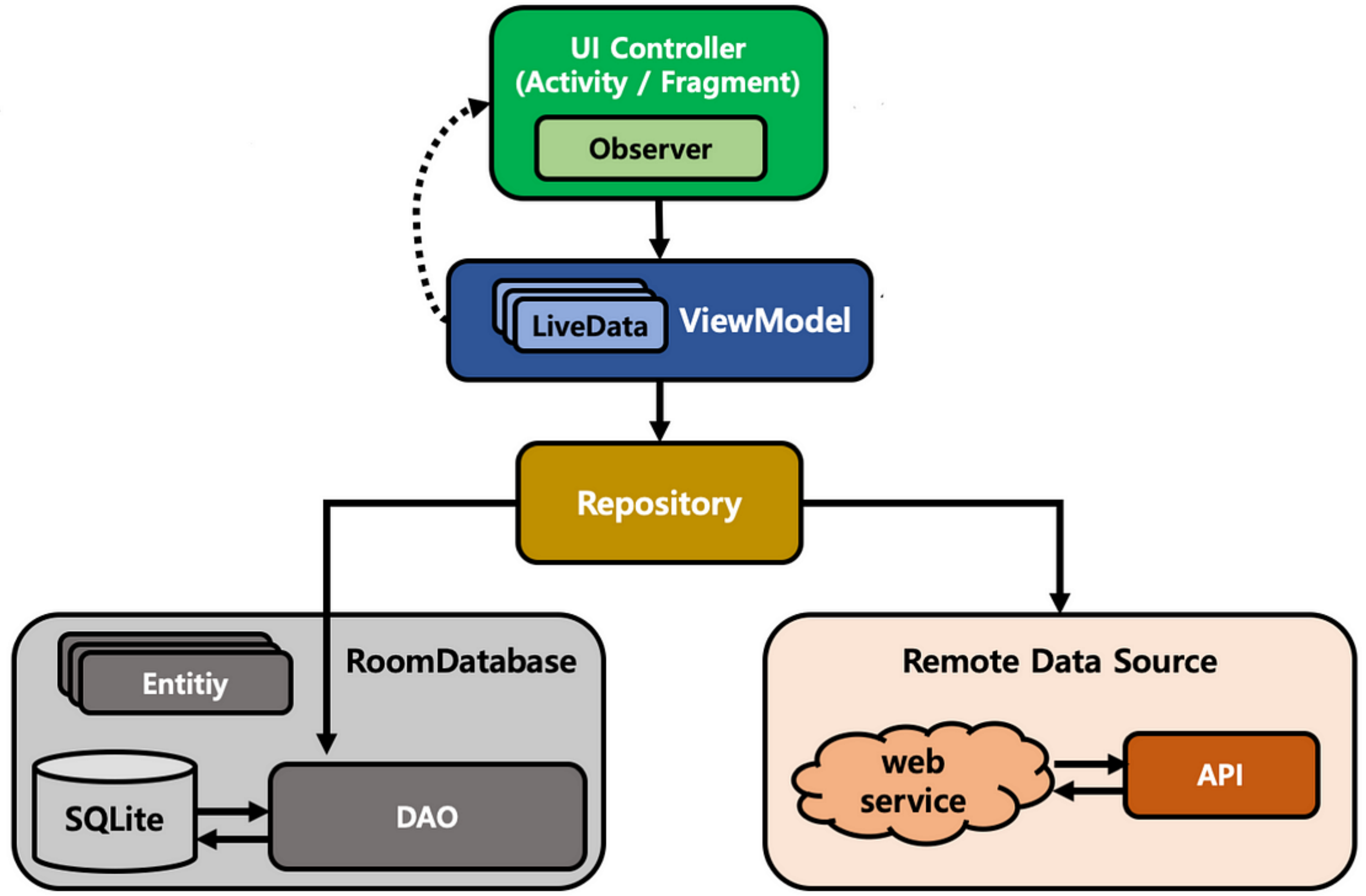
The view is not explicitly referenced by the ViewModel. It communicates with the view via a binder. As a result, it only shows data that is relevant to the view.



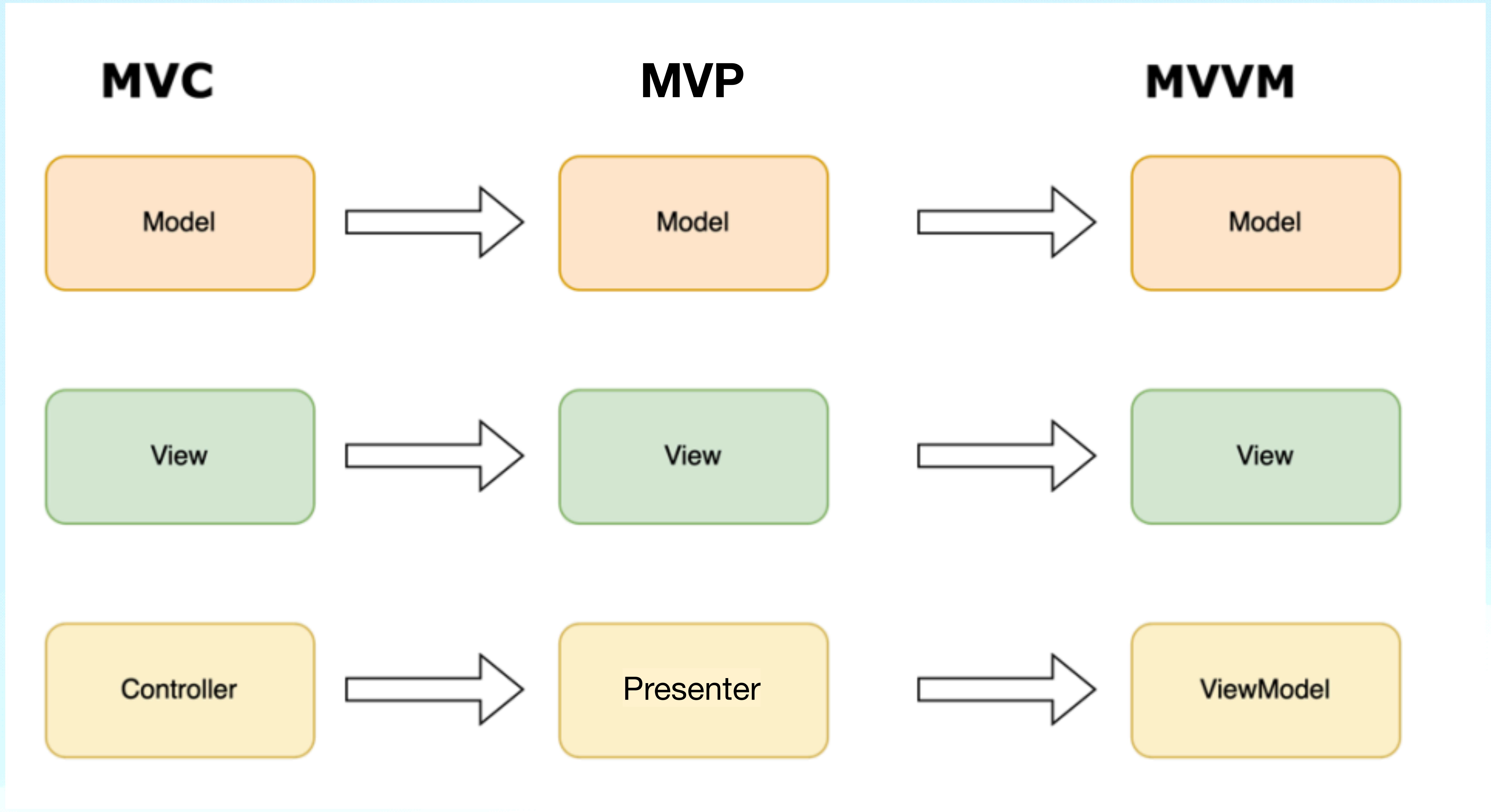
MVC in Android



MVP in Android

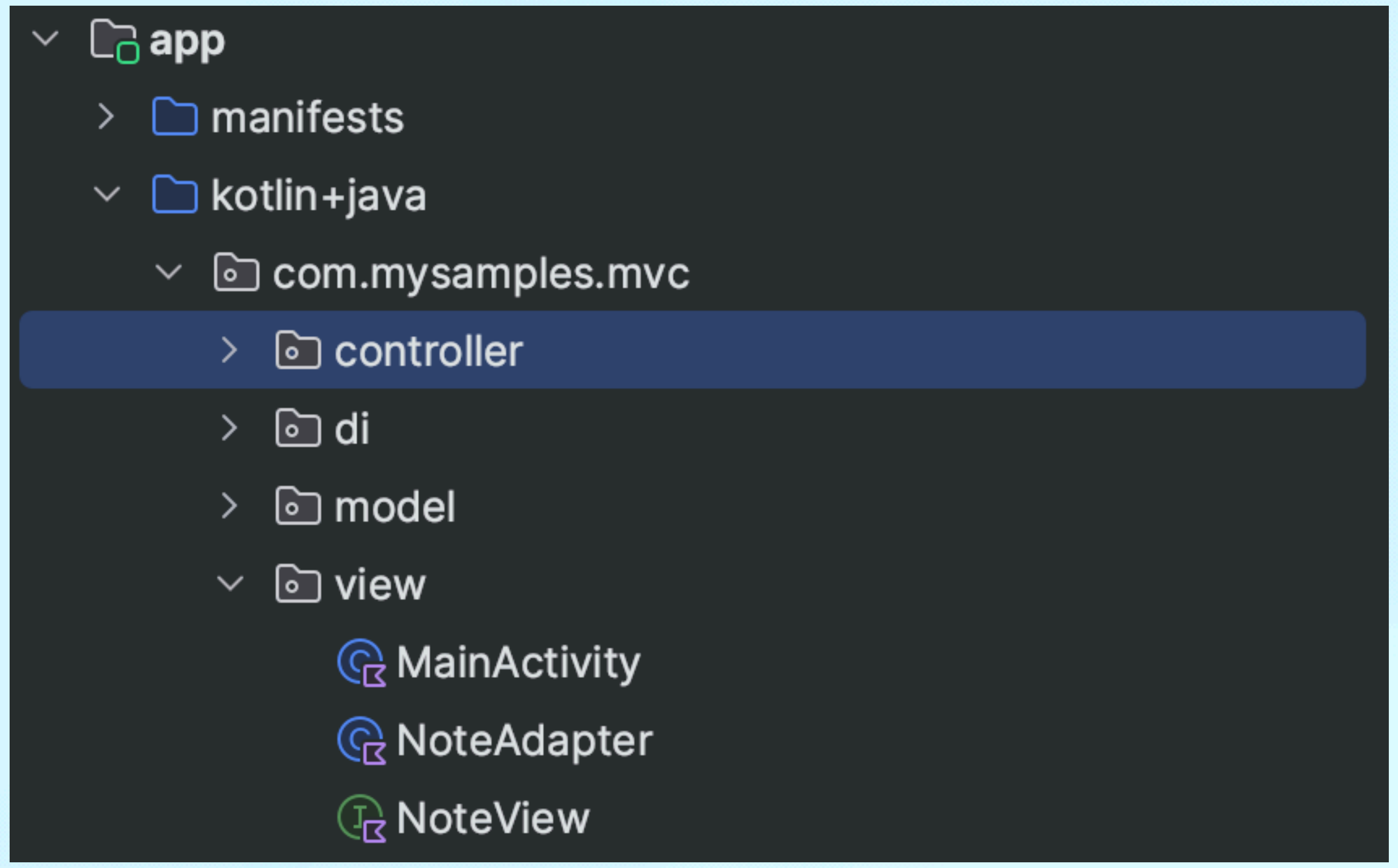


MVVM in Android

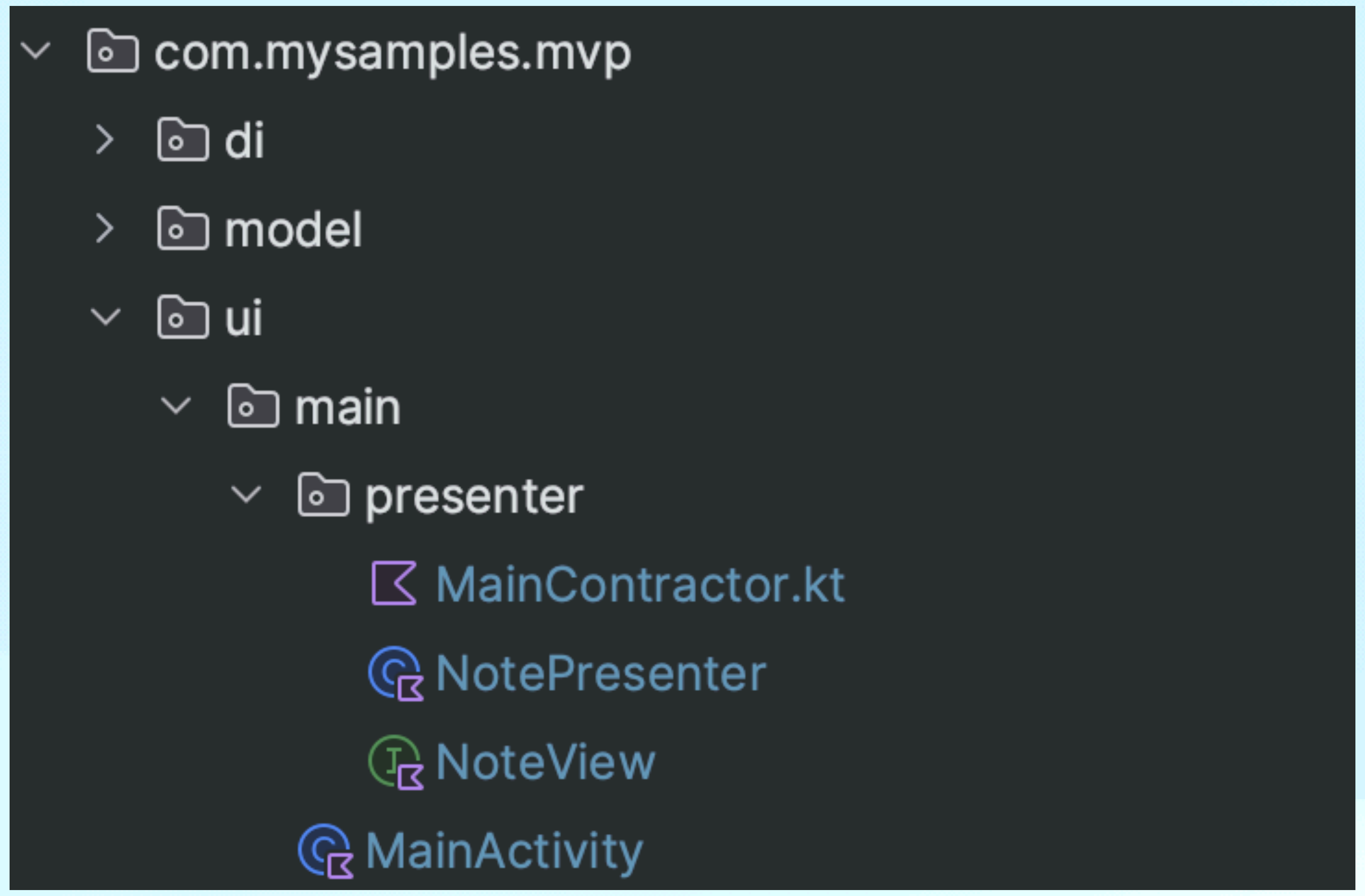


A single controller can be shared by multiple views. As a result, the Controller and View have a one-to-many relationship.	One presenter is assigned to each view. As a result, there is a one-to-one relationship between the view and the presenter.	View and ViewModel have a one-to-many relationships.
Determines which view needs to be updated.	The related view of the presenter will be updated.	The User Interface is modified by the ViewModel.

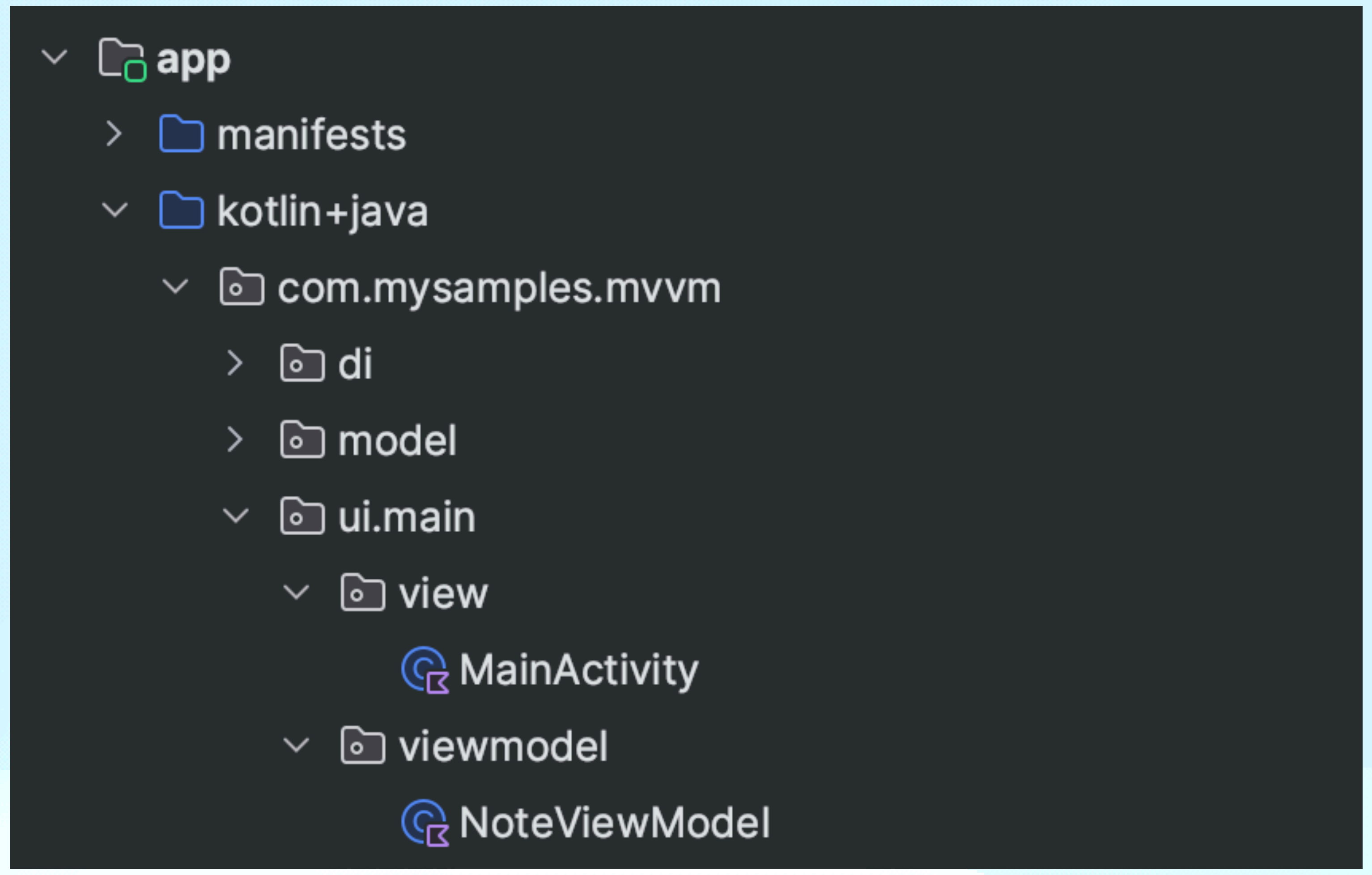
MVC -> MVP -> MVVM



```
app
  manifests
  kotlin+java
    com.mysamples.mvc
      controller
      di
      model
      view
        MainActivity
        NoteAdapter
        NoteView
```



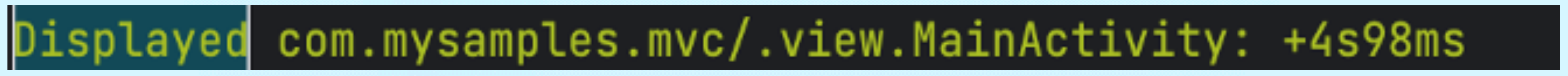
```
com.mysamples.mvp
  di
  model
  ui
    main
      presenter
        MainContractor.kt
        NotePresenter
        NoteView
      MainActivity
```



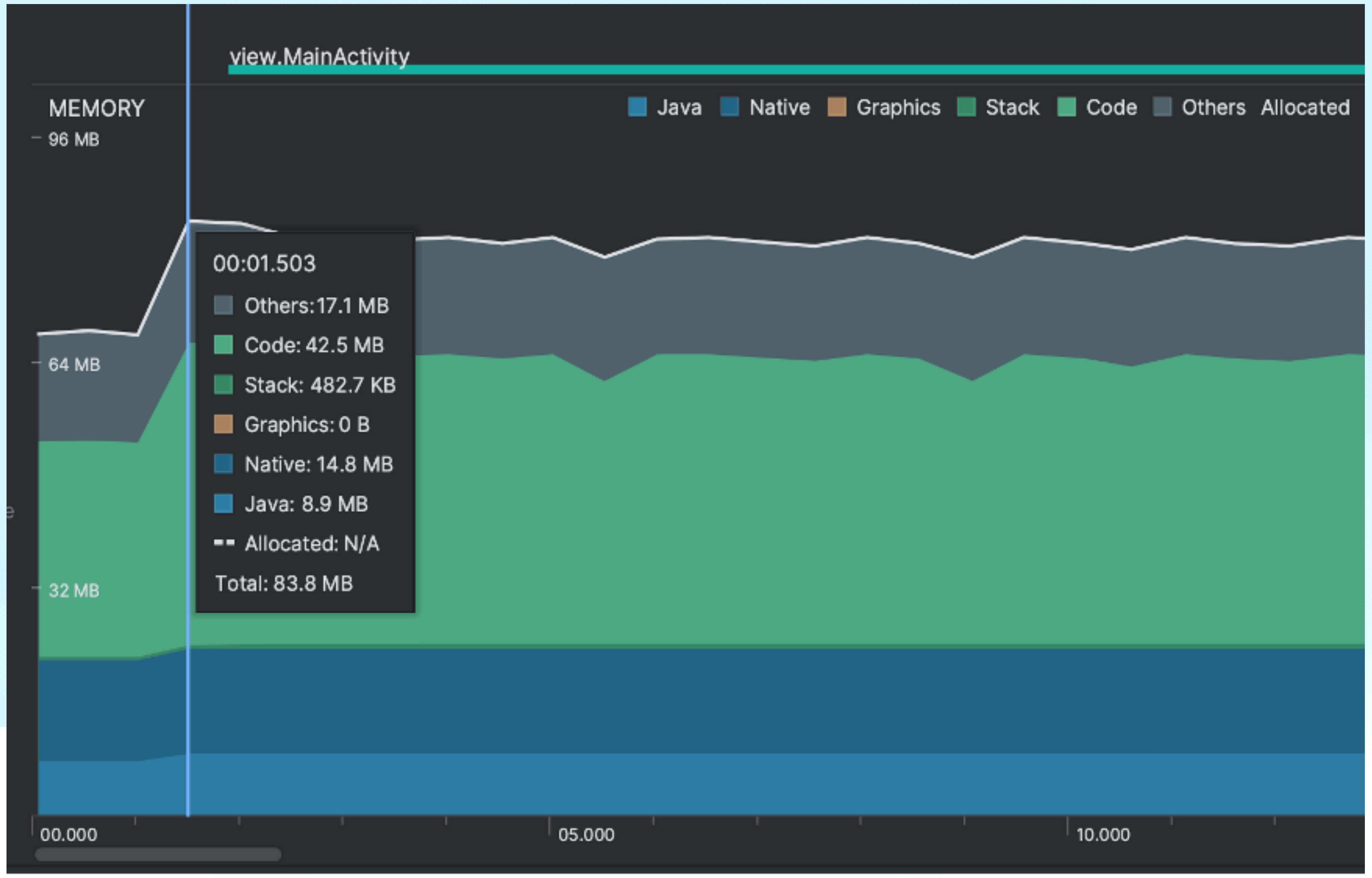
```
app
  manifests
  kotlin+java
    com.mysamples.mvvm
      di
      model
      ui.main
        view
          MainActivity
        viewmodel
          NoteViewModel
```

- To get optimal results, application performance measurements was carried out with simulator of Android Studio.
- The measurement was done using a tool available in android studio called android profiler which will be able to measure CPU usage.
- On the measurement of memory usage android profiler will show the amount of memory used by each memory category.

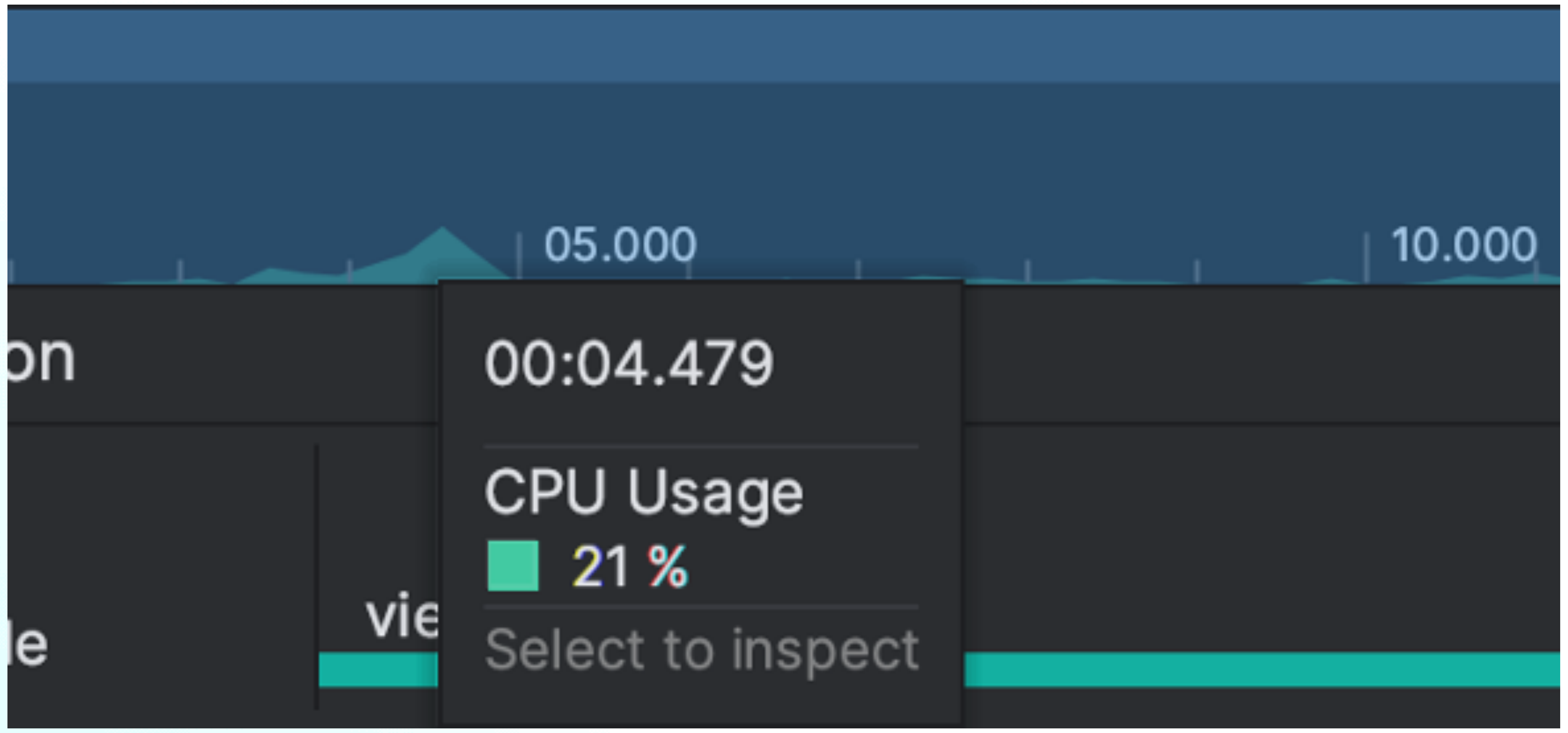
Result



```
Displayed com.mysamples.mvc/.view.MainActivity: +4s98ms
```

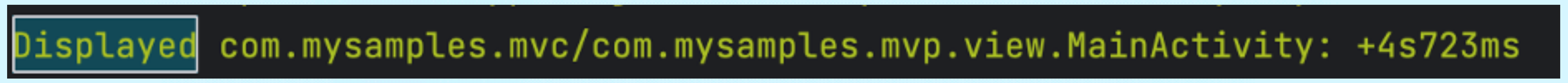


```
00:01.503
Others:17.1 MB
Code: 42.5 MB
Stack: 482.7 KB
Graphics: 0 B
Native: 14.8 MB
Java: 8.9 MB
Allocated: N/A
Total: 83.8 MB
```

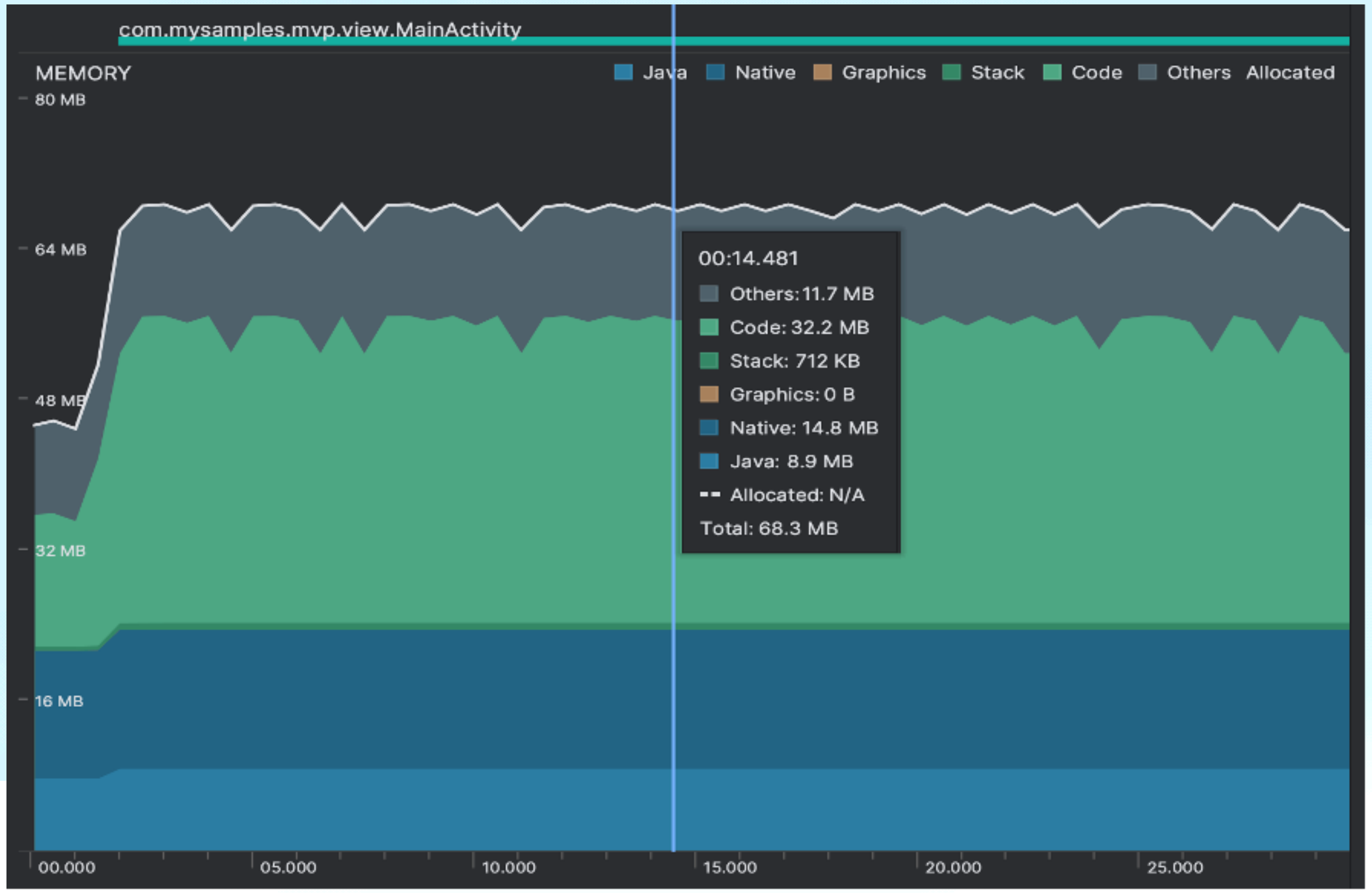


```
00:04.479
CPU Usage
21 %
Select to inspect
```

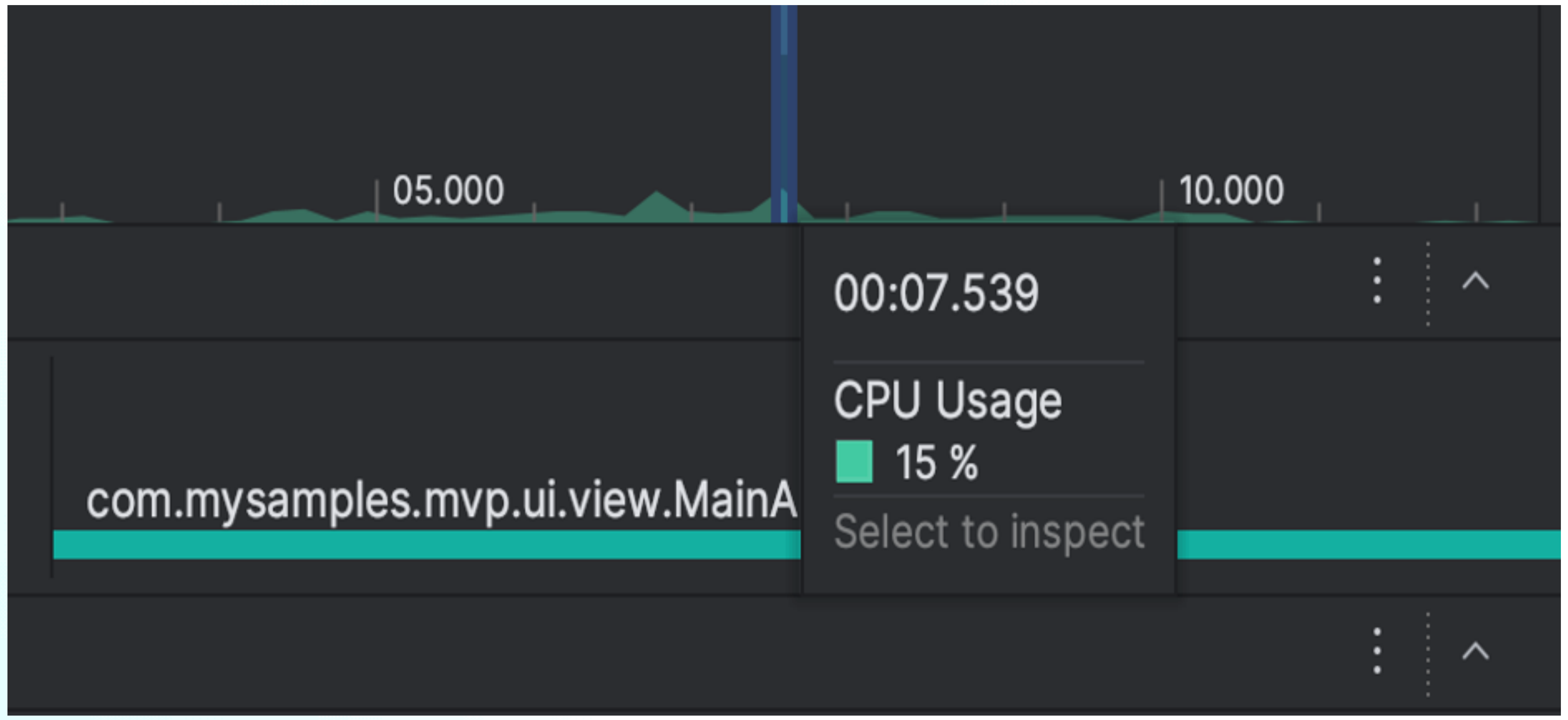
MVC



```
Displayed com.mysamples.mvc/com.mysamples.mvp.view.MainActivity: +4s723ms
```



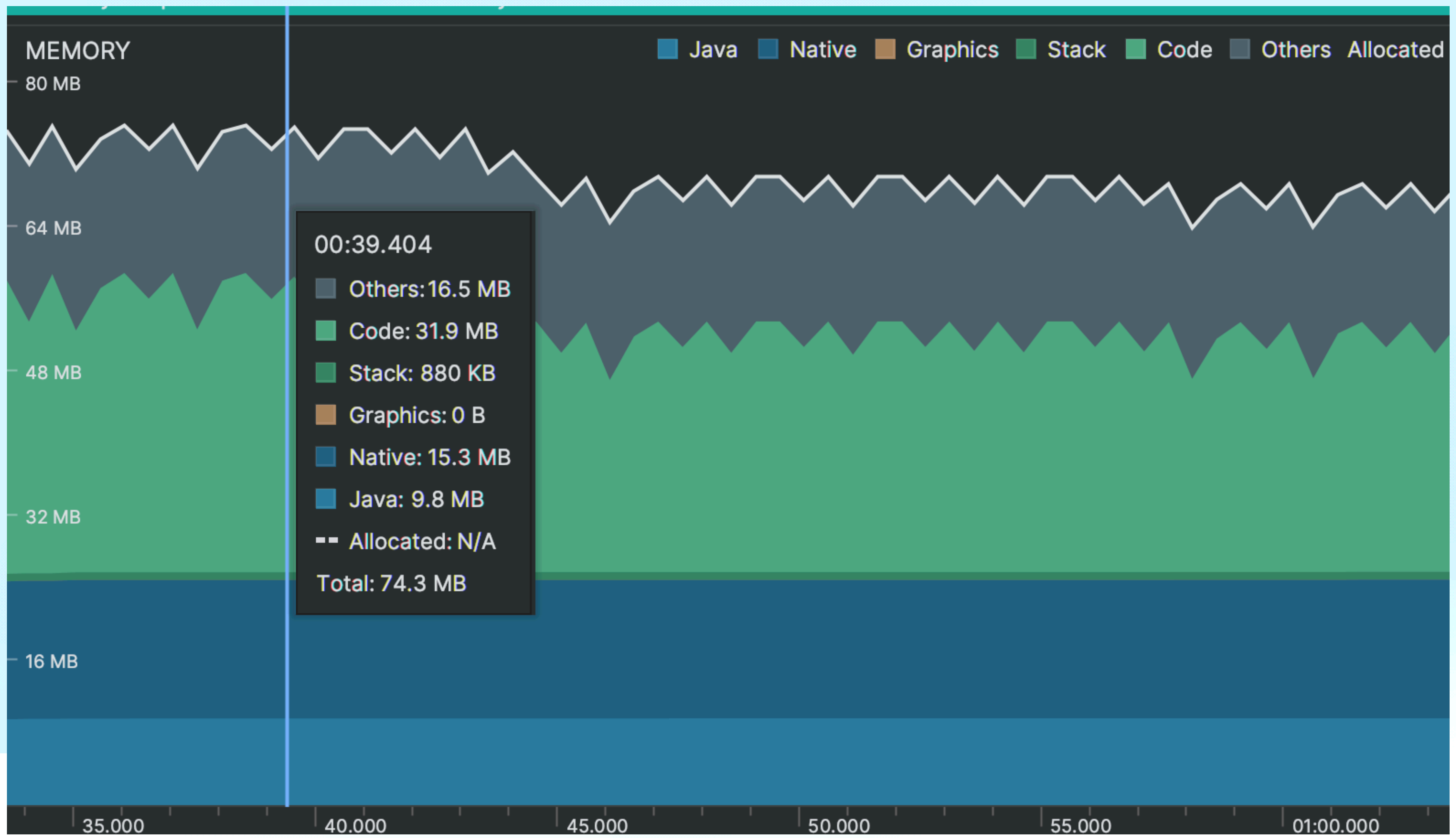
```
00:14.481
Others: 11.7 MB
Code: 32.2 MB
Stack: 712 KB
Graphics: 0 B
Native: 14.8 MB
Java: 8.9 MB
Allocated: N/A
Total: 68.3 MB
```



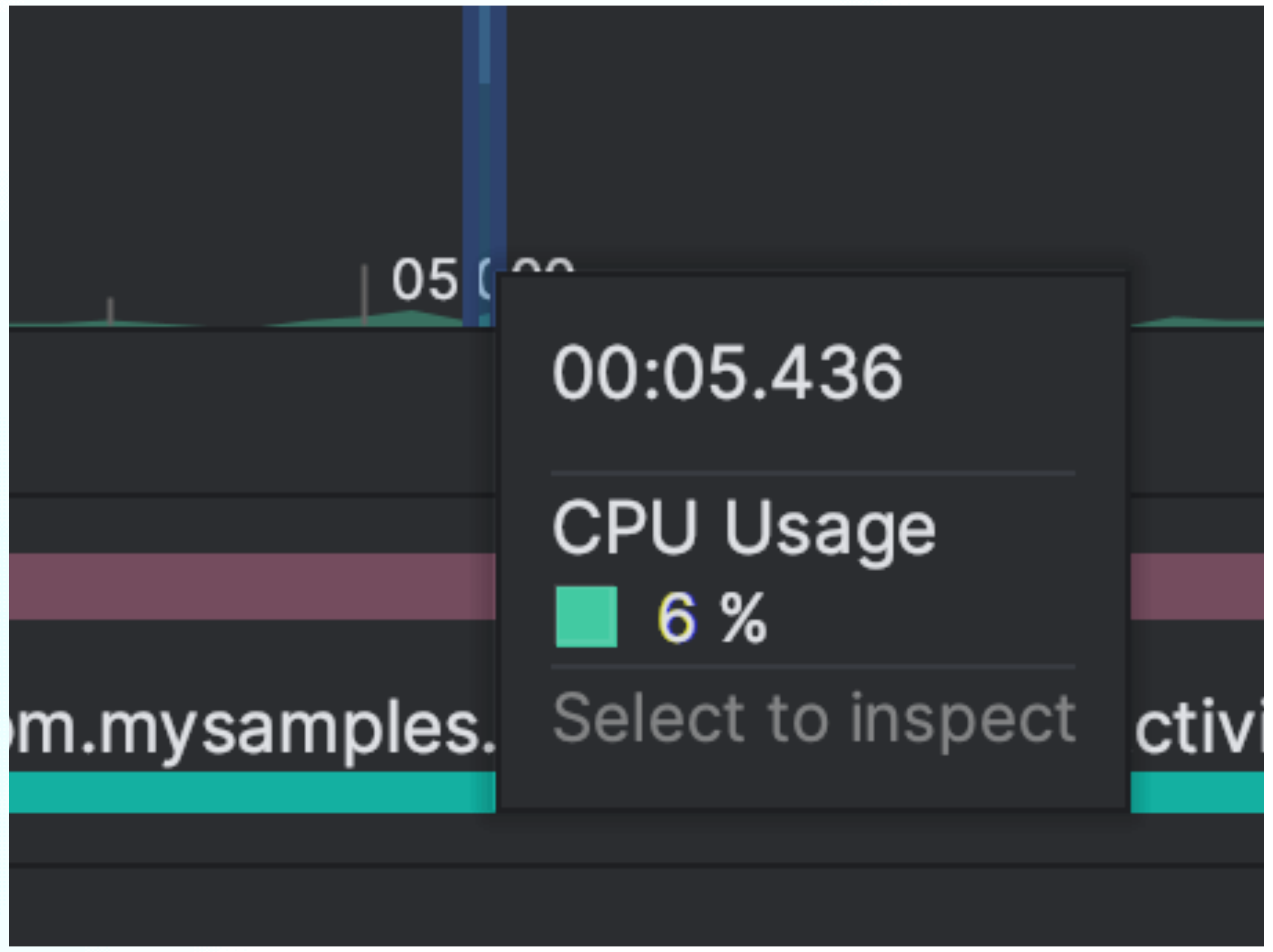
```
00:07.539
CPU Usage
15 %
Select to inspect
```

MVP

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



00:39.404
Others: 16.5 MB
Code: 31.9 MB
Stack: 880 KB
Graphics: 0 B
Native: 15.3 MB
Java: 9.8 MB
Allocated: N/A
Total: 74.3 MB



00:05.436
CPU Usage
6 %
Select to inspect

MVVM

Conclusion

- Architecture plays a great role in the development of application. MVVM architecture is viable for the development of the android. But still we cannot say that MVVM is the best architecture for android development in all the situations. Every project has different nature, so architecture must be chosen according to the nature of the project and needs.

- Google's primary recommendations support MVVM, making use of things like LiveData and ViewModels to address the two most common issues that Android apps face: lifecycle and rotation-change pitfalls. Proper separation of logic and behavior allows applications to be both flexible and easy to maintain.

Future Work

- The proposed future research might be the best practice to execute MVVM with MVI (Model-View-Intent) architecture in Android development. It has the potential to become the basis for the development of mobile architectural patterns in the future.
- This approach helps to separate data model and ui model to make maintaining easer than MVVM, since when developer needs to pass modified data to ui, he can transform to ui model in mapper class or view model itself.

Thank you for attention!