**Comparative Analysis of Mobile Application Architectures**

**Fidan Hasanguliyeva**

**Current Progress**

I have read about MVC and MVP patterns in different articles, and papers to get general information about them. I have developed simple application to measure startup time and cpu usage for each pattern.

The **Model-View-Controller (MVC)** pattern is a widely used architectural design pattern in software development, including Android applications. It helps in organizing and separating the concerns of an application, making it easier to maintain, scale, and modify. In the context of Android development, the components of MVC are typically distributed as follows:

1. **Model:**

The Model represents the data and business logic of the application. It manages the data, enforces business rules, and responds to requests from the Controller. In Android, the Model could consist of classes that handle data storage, network communication, and data processing. This may include database interactions, API calls, and data manipulation.

1. **View:**

The View is responsible for the presentation layer and user interface of the application. It displays data from the Model and sends user input to the Controller for processing. In Android, Views are typically implemented using XML layout files that define the user interface elements, such as TextViews, Buttons, EditTexts, etc. However, it's essential to keep the View passive, meaning it should not contain any business logic.

1. **Controller:**

The Controller acts as an intermediary between the Model and the View. It receives user input from the View, processes it, interacts with the Model to fetch or update data, and then updates the View accordingly. In Android, the Controller is often represented by Activities or Fragments. These components handle user interactions, trigger data retrieval or modification, and update the UI.

However, since they all rely on each other, if one of the components fail, it may pose a problem for the whole application. Furthermore, part separability may create a problem of extreme maintenance and regulations. This pattern may not be appropriate for small real-world problems, and implementing it in our application may require an additional concept comprehension of problem definition.

A diagram of a system

Description automatically generated

**App startup time** with MVC pattern:



**Memory usage:**

A screenshot of a computer

Description automatically generated

**CPU Usage**

**A screenshot of a computer

Description automatically generated**

**Model–view–presenter (MVP)** is a derivation of the model–view–controller (MVC) architectural pattern. Since it is a derivation of MVC, it has many features similar to it. It appeared for the first time in IBM and then in Taligent during the 90s.

In MVP, the presenter assumes the functionality of the “middle-man”. In MVP, all presentation logic is pushed to the presenter.

Although MVP is a derivation of MVC, they do have their slight differences.

MVP consists of three components-

1. **Model** – The Model represents a set of classes that describes the business logic and data. It also defines business rules for data means how the data can be changed and manipulated.
2. **View** – View is used for making interactions with users like XML, Activity, fragments. It has got nothing to do with the logic that is to be implemented in the process.
3. **Presenter** – It is the processing unit of the system. The presenter gets the input from the View, processes the data with the help of Model, and passes the results back to View after the processing is done.

To simplify the MVP model, the presenter responds to the input data and returns it to View after the processing is done. View and Presenter are not related to each other and interact only through the interface.

However, if we look at it from another perspective, the pattern can suffer a disadvantage because the view would become passive to the model. In reality, applying this pattern would necessitate more experience and expertise. This means more time to put into development, deployment and unit testing.

A diagram of a data management system

Description automatically generated

**App startup time** with MVP pattern:



**Memory usage:A screenshot of a graph

Description automatically generated**

**CPU Usage**

**A screenshot of a computer

Description automatically generated**