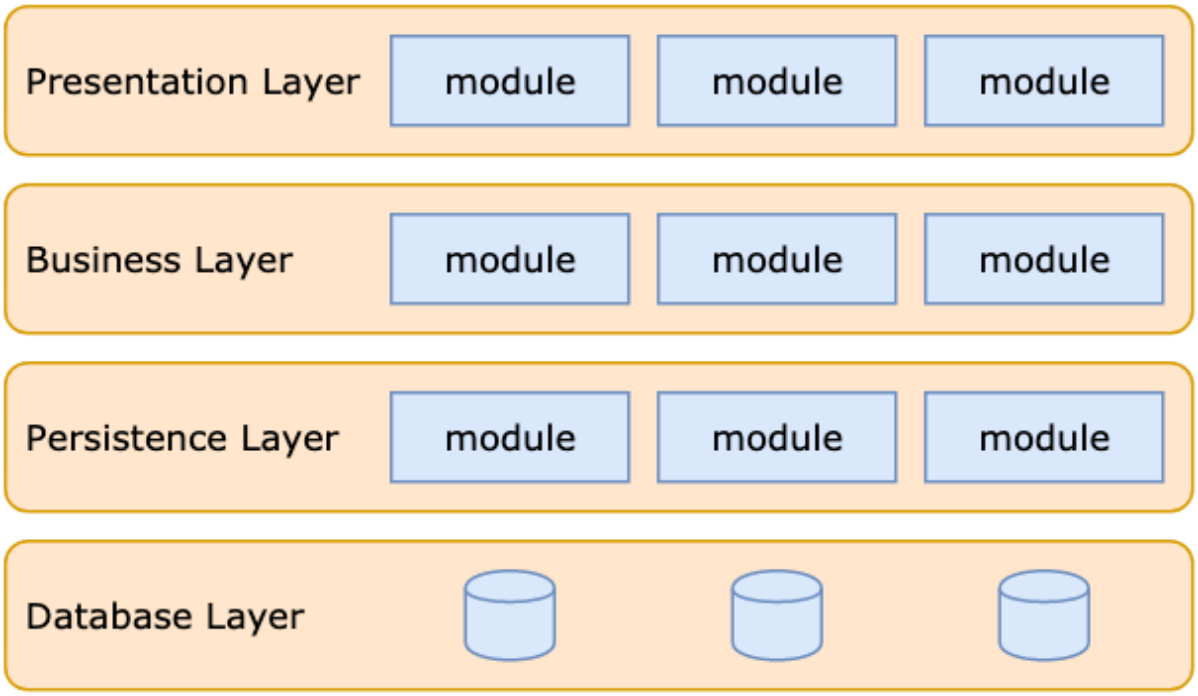
News App:Bharat News

Github Link:-https://github.com/PraveenAade15/BharatNew/tree/master
Tech Stack:
MVVm
Kotlin
Retrofit Api
Room Database
Ui

1. Layered Architecture

The layered architecture is the most common in adoption, well-known among developers, and hence the de facto standard for applications. If you do not know what architecture to use, use it.



Examples

- TCP / IP Model: Application layer > transport layer > internet layer > network access layer
- Retrofit: web layer > cache layer (follower + leader) > database layer

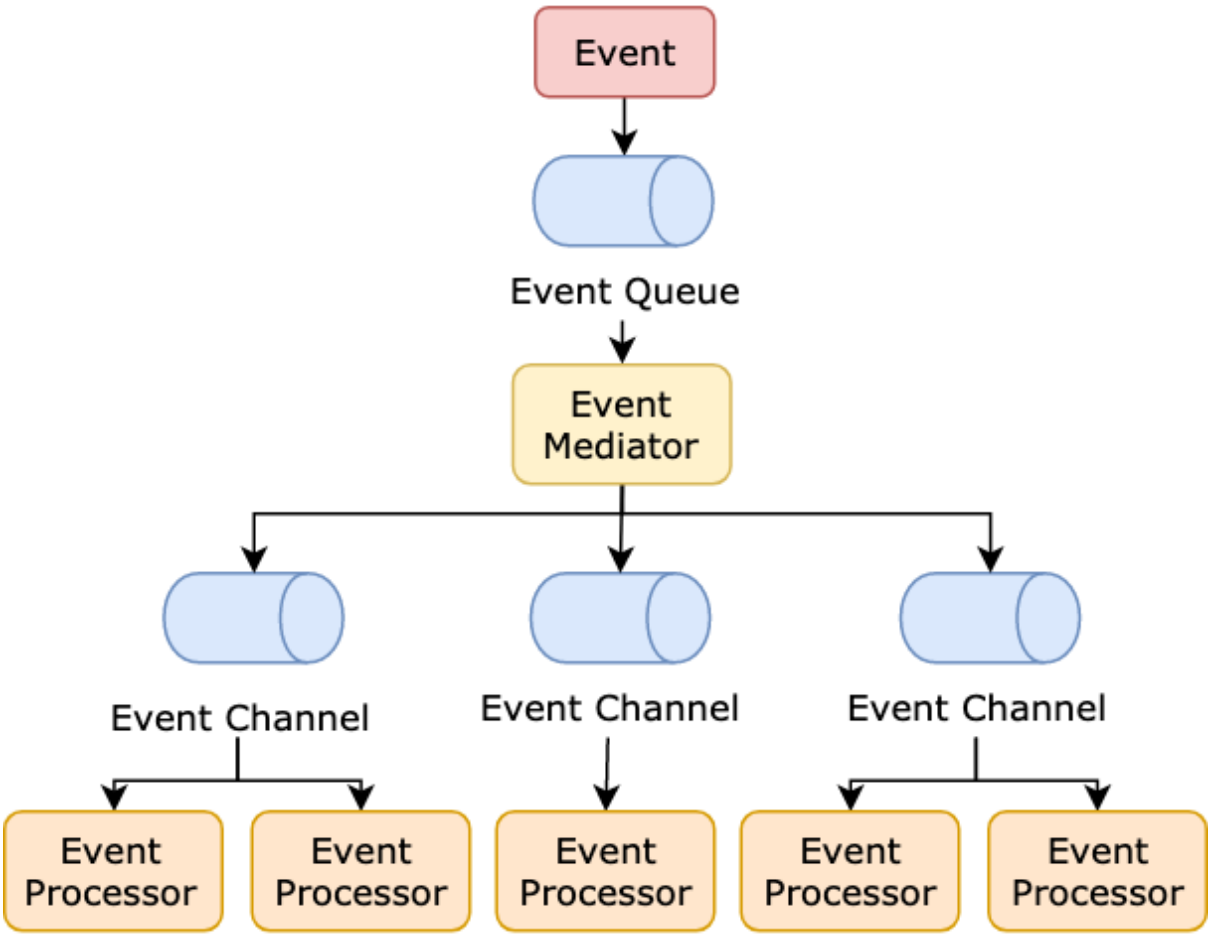
Pros and Cons

- Pros

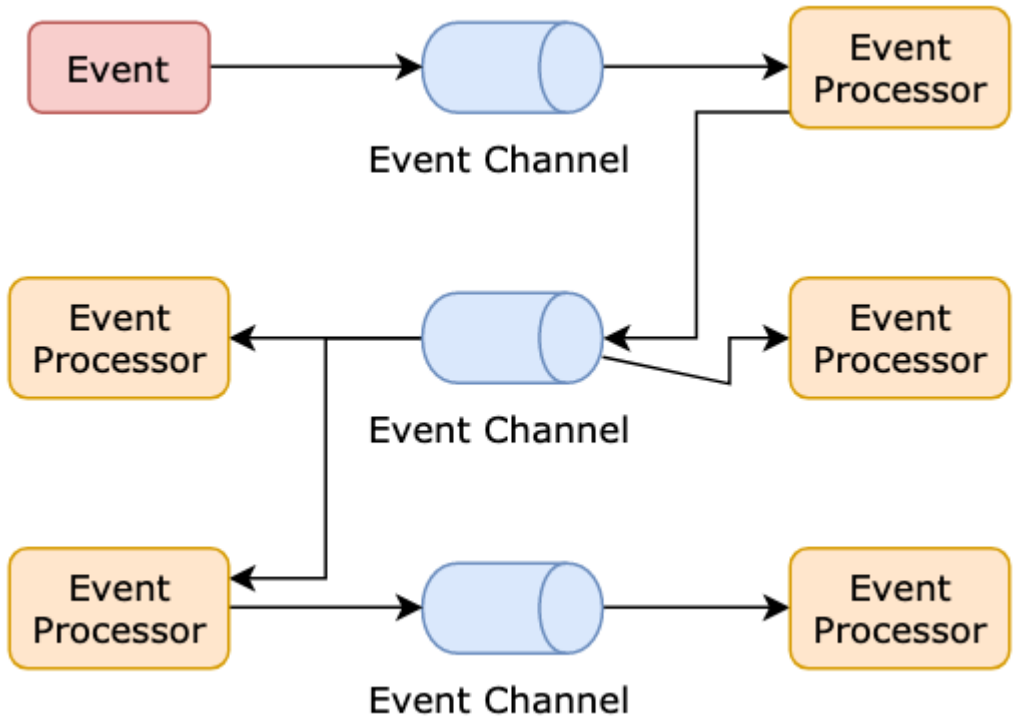
    - ease of use
    - separation of responsibility
    - testability
- Cons
    - monolithic
        - hard to adjust, extend or update. You have to make changes to all the layers.

2. Event-Driven Architecture

A state change will emit an event to the system. All the components communicate with each other through events.



A simple project can combine the mediator, event queue, and channel. Then we get a simplified architecture:

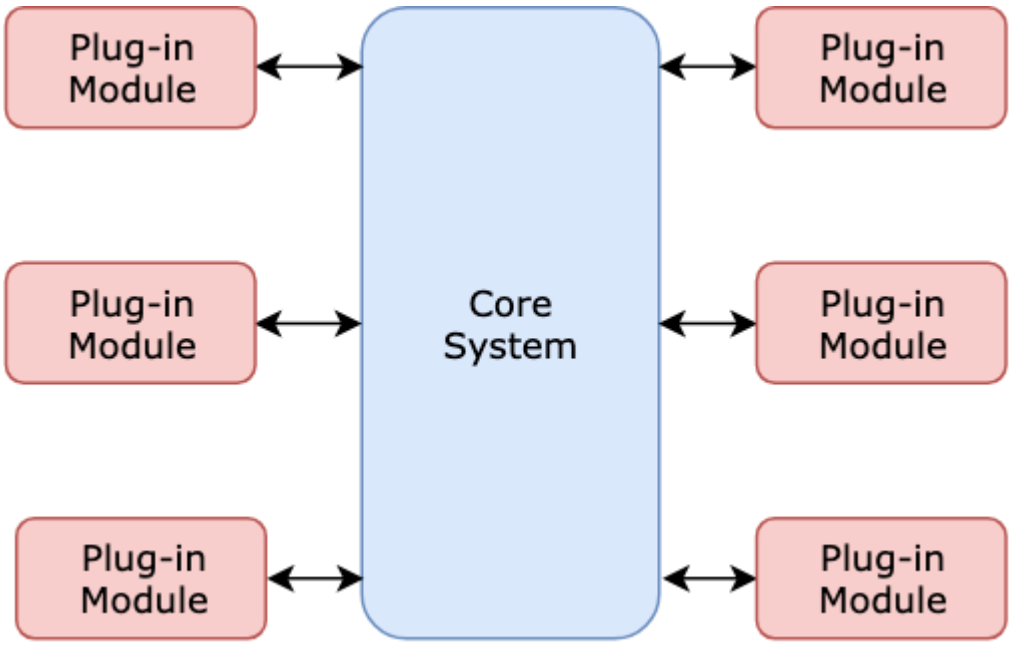


Examples

- QT: Signals and Slots
- Payment Infrastructure: Bank gateways usually have very high latencies, so they adopt async technologies in their architecture design.

3. Micro-kernel Architecture (aka Plug-in Architecture)

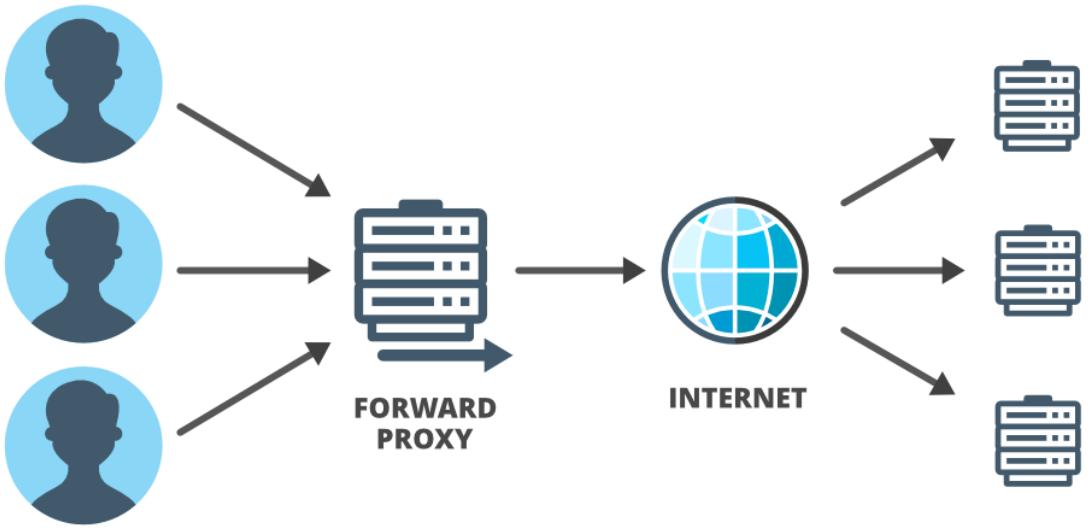
The software's responsibilities are divided into one "core" and multiple "plugins". The core contains the bare minimum functionality. Plugins are independent of each other and implement shared interfaces to achieve different goals.

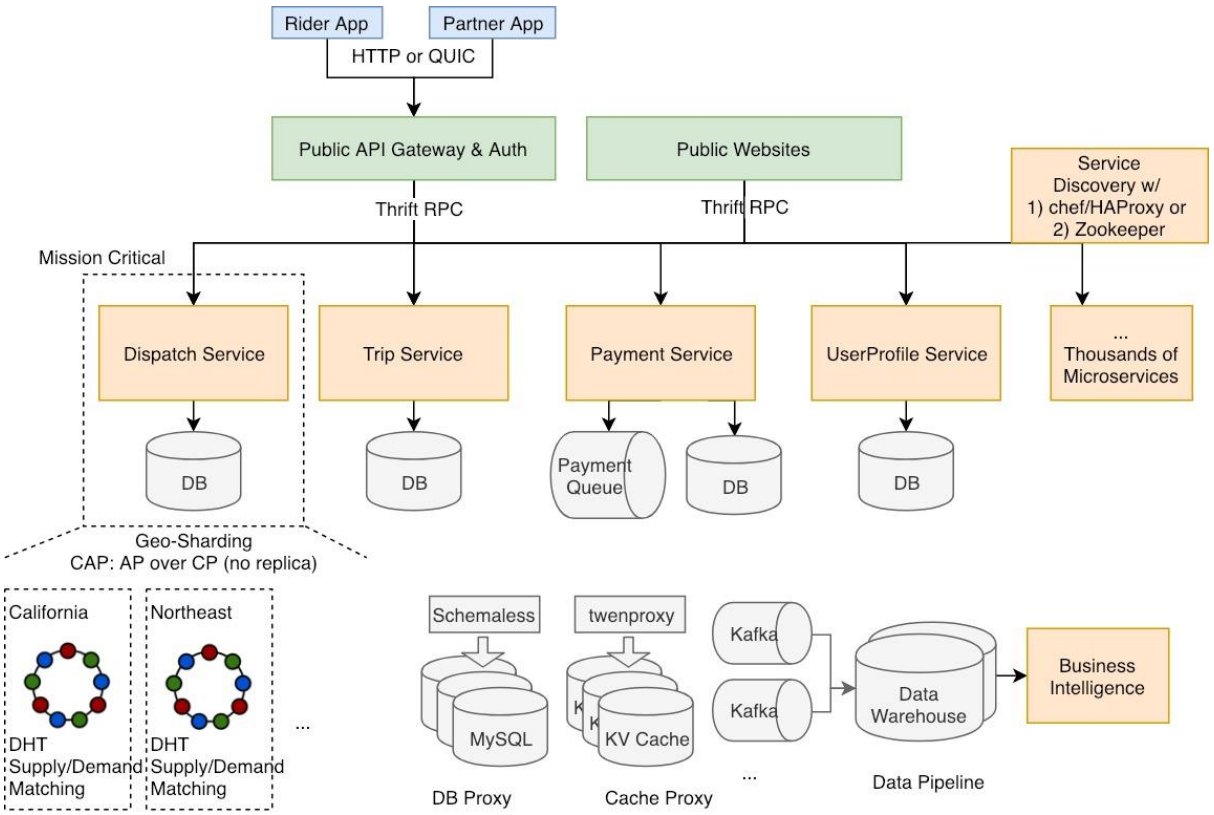


- android studio
- MINIX operating system

4. Microservices Architecture

A massive system is decoupled to multiple micro-services, each of which is a separately deployed unit, and t

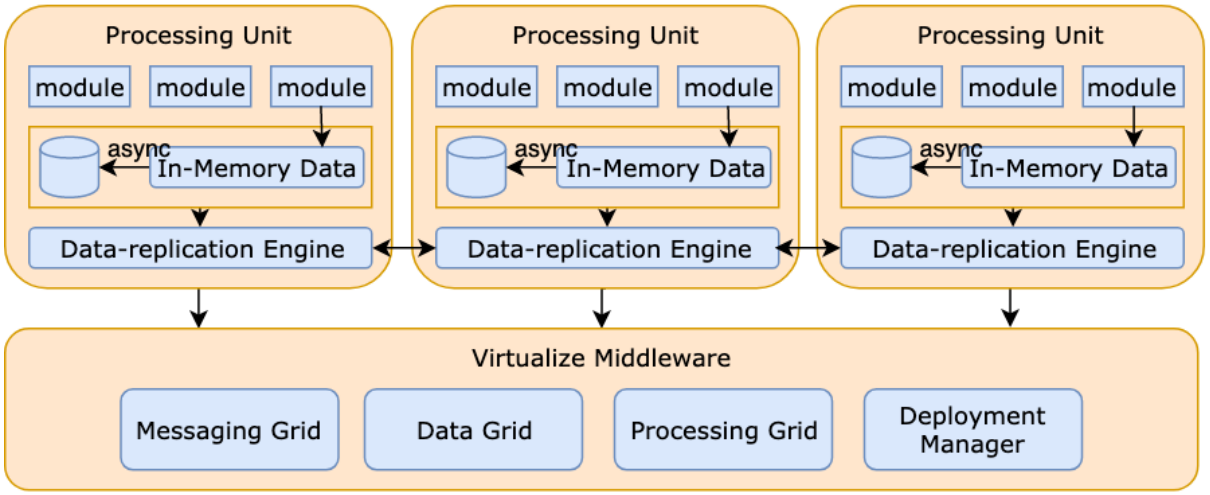




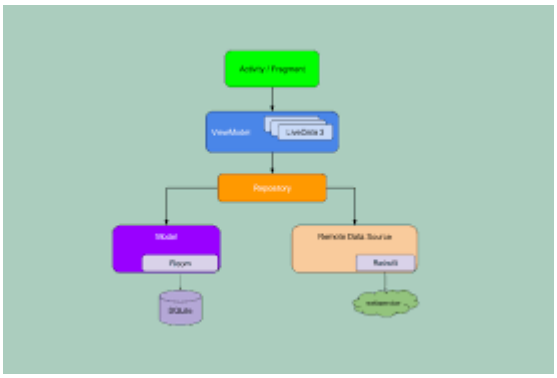
-

5. Space-based Architecture

This pattern gets its name from "tuple space", which means “distributed shared memory". There is no database or synchronous database access, and thus no database bottleneck. All the processing units share the replicated application data in memory. These processing units can be started up and shut down elastically.



mvvm architecture



View

This part of our architecture help us build our user interface and the only part our users can interact directly. I consist of Fragment object defined in “src” and a layout resource. There is a two-way binding between them, which allows easy data sharing.

ViewModel

ViewModel object acts as an intermediate between View and the Model, meaning it provides data for the UI components like fragments or activities. It also includes an observable data

holder called LiveData that allows ViewModel to inform or update the View whenever the data get updated. It is very crucial, mainly to keep our app from reloading on orientation changes. Which ultimately provides a great user experience.

Model

Model is responsible for fetching the data either from the local SQLite database or from a web service. So it is further divided into various components.

- **Room** — It is an ORM provided by Google, which provides an abstraction layer between the SQLite database and our data in the form of objects. It gives us errors in compile-time, which is much better than run-time error which difficult to track and debug.
  In order to use the room, it's very important to define our schema. We do that by creating a data model class

and add an @entity annotation. We also must add a @PrimaryKey annotation to our entity's id.

- **Web Service —** Now if we wish to access data from a REST API, we have a library called "Retrofit" at our disposal. And it helps us make our network calls. In short, it feeds us the JSON string; we convert that JSON response to java object and add as an entity to our app.

Conclusion

One an easily make android apps without following this architecture, but if we want to make apps that are robust, testable, maintainable and easy to read, then we must use this to our advantage. I will be writing more on MVVM architecture and how to follow the best practices.

-