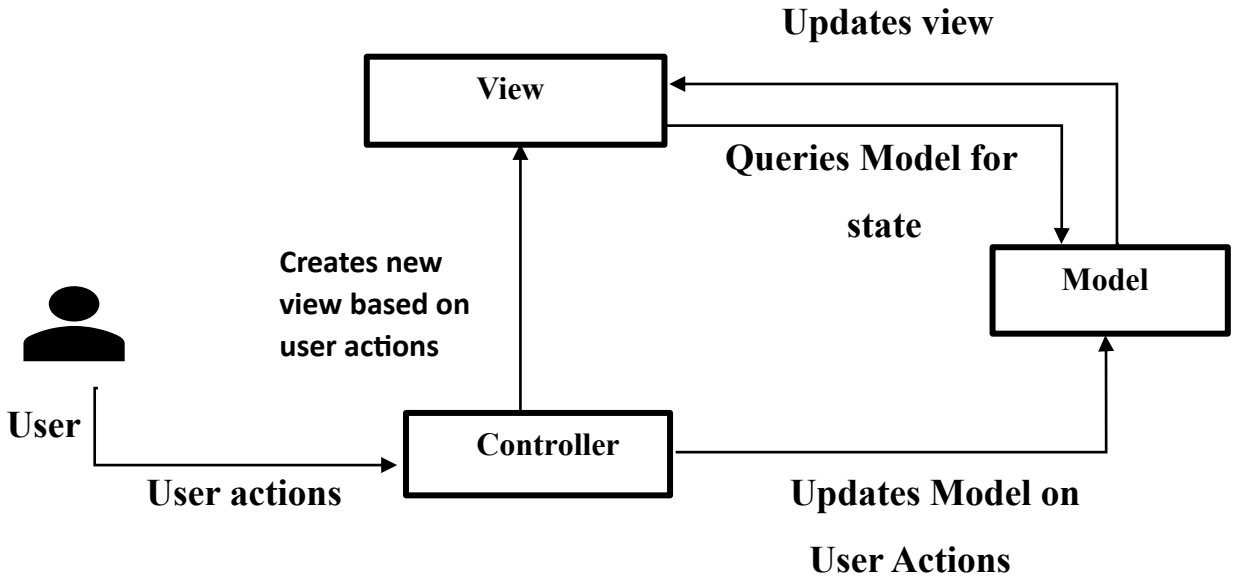
**Day 2-30 April 2024**

**Assignment 1:** Design Pattern Explanation - Prepare a one-page summary explaining the MVC (Model-View-Controller) design pattern and its two variants. Use diagrams to illustrate their structures and briefly discuss when each variant might be more appropriate to use than the others.

## Model View Controller (MVC):




MVC is a commonly used architectural pattern in Software Engineering that divides the application into 3 components. This separation of concerns promotes modularity, maintainability, and scalability in software development.

**1.Model:** The logic function of the system, to carry out computations and data handling.

- Represents the application's data and business logic.
- Manages the state of the application and responds to requests for data manipulation.
- Independent of the user interface (UI) and presentation logic.

**2.View:** Shows output to users.

- Presents the application's data to the user.
- Renders the UI based on the state of the Model.
- Passively observes changes in the Model and updates the UI accordingly.
- Multiple Views can interact with the same Model.

**3.Controller:** Handles input by the user into the system.

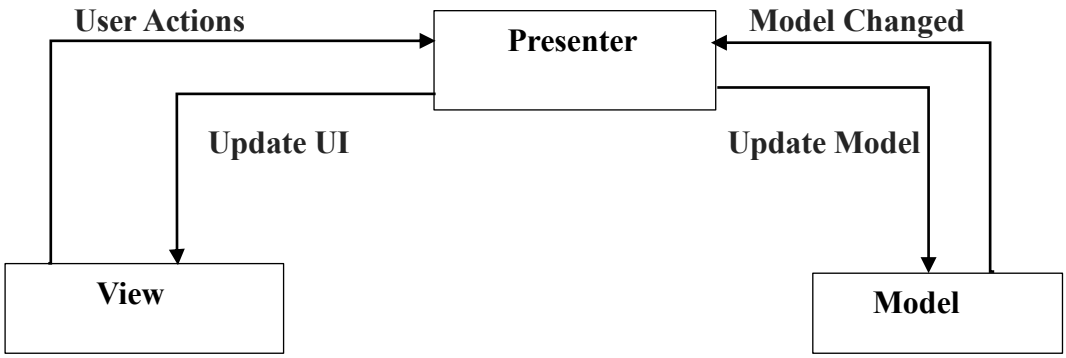
- Interprets user inputs and translates them into actions to be performed by the Model or View.
- Coordinating the flow of data and application logic.

- Updates the Model in response to user interactions and updates the View based on changes in the Model.

## Variants of MVC:

MVC, MVP, and MVVM are three popular design patterns in software development. Let's have a look on Model View Controller (MVC), Model View Presenter (MVP) and Model View View-model (MVVM) one by one. All these design patterns by and large help in developing applications that are loosely combined, easy to test and maintain. All discussion about the pattern is made in context of Android as a platform.

**Model View Presenter (MVP)**




The MVP pattern is similar to the MVC pattern. It is derived from MVC pattern, wherein the controller is replaced by the presenter. This pattern divides an application into three major aspects: Model, View, and Presenter.

**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.

**View**

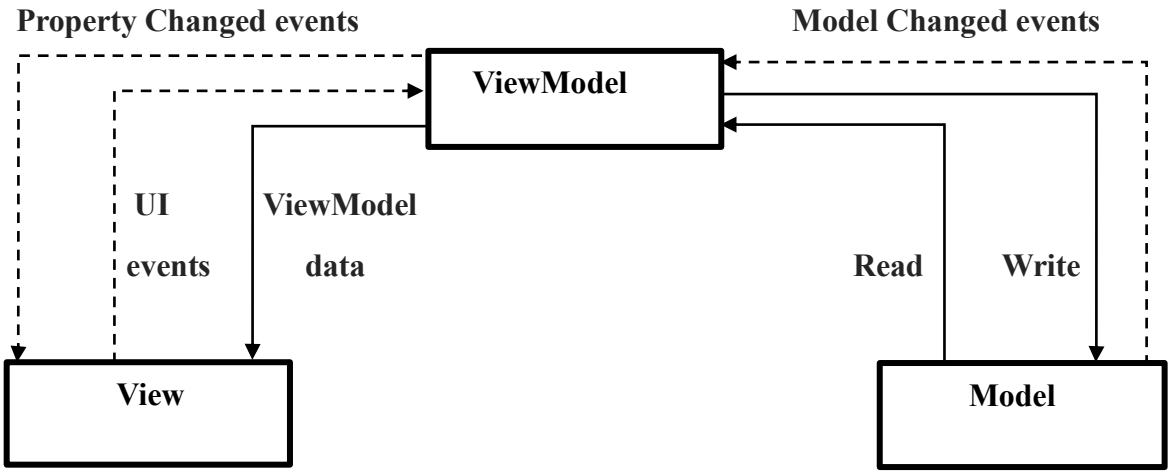
View is a component which is directly interacts with user like XML, Activity, fragments. It does not contain any logic implemented.

**Presenter**

The Presenter receives the input from users via View, then process the user's data with the help of Model and passing the results back to the View. Presenter communicates with view through interface. Interface is defined in presenter class, to which it pass the required data. Activity/fragment or any other view component implement this interface and renders the data in a way they want.

In the MVP design pattern, the presenter manipulates the model and also updates the view. In MVP View and Presenter are completely decoupled from each other's and communicate to each other's by an interface. Because if decoupling mocking of the view is easier and unit testing of applications that leverage the MVP design pattern over the MVC design pattern are much easier.

**Model View** View-model **(MVVM)**




MVVM pattern supports two-way data binding between View and View-Model. This allows automatic propagation of changes, inside the state of View-Model to the View. Generally, the View-Model utilizes the observer pattern to inform changes in the View-Model to the Model.

**View-Model**

It is responsible for exposing methods, commands, and other properties that help to maintain the state of the view, manipulate the model as the result of actions on the view, and trigger events in the view itself. View has a reference to View-Model but View-Model has no information about the View. There is many-to-one relationship between View and View-Model means many Views can be mapped to one View-Model. It is completely independent of Views.

The bi-directional data binding or the two way data binding between the view and the View-Model ensures that the models and properties in the View-Model is in sync with the view. The MVVM design pattern is well suited in applications that need support for bi-directional data binding.

**Assignment 2:** Principles in Practice - Draft a one-page scenario where you apply Microservices Architecture and Event-Driven Architecture to a hypothetical e-commerce platform. Outline how SOLID principles could enhance the design. Use bullet points to indicate how DRY and KISS principles can be observed in this context.

## Applying Microservices and Event-Driven Architecture to an E-commerce Platform

In our hypothetical scenario, we're developing an e-commerce platform called "EcoMart" that sells eco-friendly products. We'll utilize Microservices Architecture and Event-Driven Architecture to build a scalable and resilient system that can handle high traffic and frequent updates.

**Microservices Architecture:**

- **Customer Service:** Manages customer registration, authentication, and profile management.
- **Product Service:** Handles product catalog management, including adding, updating, and deleting products.
- **Order Service:** Manages order creation, processing, and fulfillment.
- **Payment Service:** Handles payment processing and integrates with payment gateways.
- **Inventory Service:** Tracks inventory levels and updates product availability in real-time.

**Event-Driven Architecture:**

- **Customer Events:** Triggered when a new customer registers, updates their profile, or logs in.
- **Product Events:** Generated when new products are added, existing products are updated, or products are deleted.
- **Order Events:** Created when a new order is placed, updated when the order status changes, and finalized when the order is fulfilled.
- **Payment Events:** Generated when a payment is processed successfully or fails.

**SOLID Principles:**

**1.Single Responsibility Principle (SRP):**

Each microservice adheres to SRP by focusing on a single aspect of the system's functionality (e.g., customer management, product catalog, order processing).

**2. Open/Closed Principle (OCP):**

Microservices are designed to be open for extension but closed for modification. New features or changes are implemented by adding new microservices or extending existing ones without modifying the core functionality.

**3. LisKov Substitution Principle (LSP):**

Microservices within the system can be substituted for one another without affecting the correctness of the system's behavior. For example, a different payment service could be seamlessly integrated without impacting other services.

**4.Interface Segregation Principle (ISP):**

Interfaces are designed to be specific to the needs of each microservice, ensuring that clients only depend on the functionalities they require. This prevents clients from being forced to depend on interfaces they don't use.

**5.Dependency Inversion Principle (DIP):**

The Dependency Inversion Principle (DIP) states that high-level modules should not depend on low-level modules; both should depend on abstractions. Abstractions should not depend on details; details should depend on abstractions. By adhering to DIP, code becomes more flexible, reusable, and easier to maintain. DIP facilitates decoupling and promotes the use of interfaces or abstract classes to represent dependencies, enabling easier substitution of implementations.

**Observing DRY and KISS Principles:**

**1.Don't Repeat Yourself (DRY):**

- Data and business logic are encapsulated within microservices, avoiding duplication across the system.
- Shared functionalities, such as user authentication and payment processing, are implemented as separate microservices to prevent duplication of code.

**2.Keep It Simple, Stupid (KISS):**

- Each microservice is designed to be simple and focused, performing a specific set of tasks without unnecessary complexity.
- Event-driven communication simplifies the integration between microservices, allowing them to react to events and communicate asynchronously.

By adhering to SOLID principles and observing DRY and KISS principles, the EcoMart e-commerce platform can achieve modularity, scalability, and maintainability while embracing Microservices and Event-Driven Architecture.

**Assignment 3:** Trends and Cloud Services Overview - Write a three-paragraph report covering: 1) the benefits of serverless architecture, 2) the concept of Progressive Web Apps (PWAs), and 3) the role of AI and Machine Learning in software architecture. Then, in one paragraph, describe the cloud computing service models (SaaS, PaaS, IaaS) and their use cases.

**The benefits of serverless architecture:**

Serverless architecture has emerged as a paradigm shift in cloud computing, offering several benefits to developers and businesses. With serverless, developers can focus solely on writing code without managing underlying infrastructure, leading to faster development cycles and reduced operational overhead. Serverless also offers automatic scalability, where resources are provisioned dynamically based on demand, resulting in cost efficiency and improved performance. Additionally, serverless enables event-driven architectures, allowing applications to respond to events in real-time, leading to greater agility and responsiveness.

**The concept of Progressive Web Apps:**

Progressive Web Apps (PWAs) have gained traction as a modern approach to web development, combining the best features of web and mobile applications. PWAs leverage web technologies to deliver app-like experiences across different devices and platforms, offering features such as offline access, push notifications, and installation to the home screen. By providing a seamless user experience and eliminating the need for app store distribution, PWAs help businesses reach a wider audience and improve user engagement. PWAs also simplify development and maintenance, as they can be built using standard web development techniques and deployed like traditional websites.

**The role of AI and Machine Learning in software architecture:**

AI and Machine Learning (ML) are increasingly integrated into software architecture to enable intelligent and data-driven applications. AI and ML algorithms can analyze large datasets, extract insights, and make predictions, enhancing decision-making and automation. In software architecture, AI and ML are used for tasks such as natural language processing, image recognition, recommendation systems, and predictive analytics. By leveraging AI and ML capabilities, businesses can improve customer experiences, optimize operations, and unlock new opportunities for innovation and competitive advantage.

**Cloud Computing Service Models Overview**

Cloud computing offers three main service models: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS).

**SaaS** provides users with access to applications hosted and managed by a third-party provider over the internet. Common examples include email services, customer relationship management (CRM) software, and productivity suites. SaaS is ideal for organizations seeking to minimize IT infrastructure costs and simplify software deployment and maintenance.

**PaaS** provides developers with a platform and tools to build, deploy, and manage applications without worrying about underlying infrastructure. PaaS offerings include development frameworks, databases, and application hosting environments. PaaS is suitable for developers looking to streamline the application development lifecycle and focus on writing code rather than managing infrastructure.

**IaaS** offers virtualized computing resources, such as virtual machines, storage, and networking, on-demand over the internet. With IaaS, users have full control over the operating system, middleware, and applications, allowing for greater flexibility and customization. IaaS is commonly used for scenarios requiring scalability, resource isolation, and control, such as web hosting, development and testing environments, and disaster recovery.