ASSIGNMENT-2.1

Given: 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.

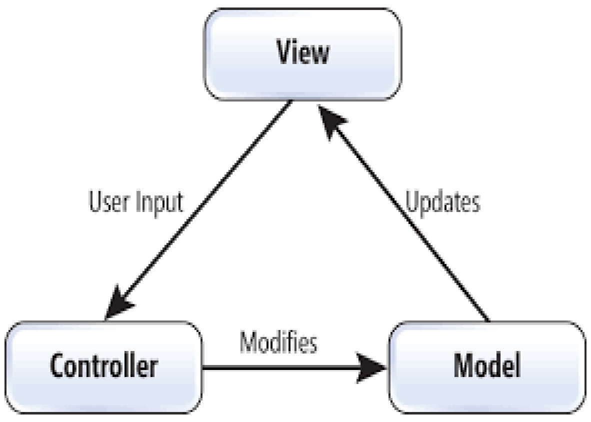
MVC (Model-View-Controller) Design Pattern and Its Variants:

The MVC design pattern separates an application into three main components to achieve separation of concerns and enhance maintainability:

- **Model**: Manages the application's data, business logic, and rules.
- **View**: Represents the presentation layer, rendering data from the Model to the user interface.
- **Controller**: Handles user input, processes requests, and updates both the Model and the View.

Variants of MVC:

1)classic MVC



Components:

Model: Represents the application's data and business logic. It manages the data and performs any necessary calculations or validations.

View: Responsible for rendering the user interface (UI) and displaying the data provided by the Model.

Controller: Acts as an intermediary between the Model and View. It receives input from the user, communicates with the Model to perform any necessary actions, and updates the View accordingly.

2) **Model-View-Presenter (MVP):**

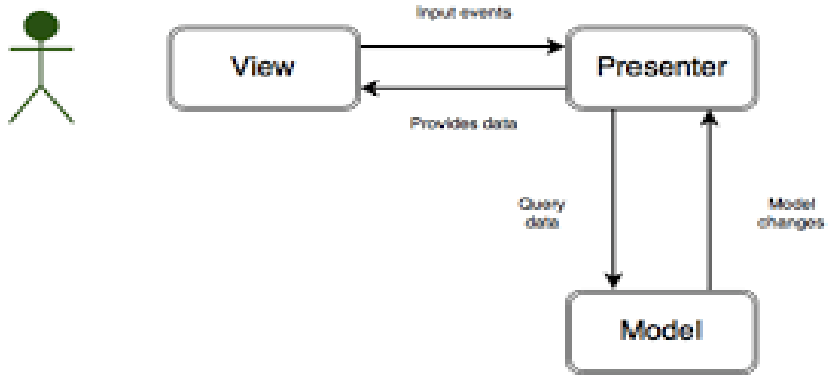


Diagram representing layer structure

MVP (Model-View-Presenter):

The MVP pattern is a variation of the MVC pattern, where the Controller is replaced by a Presenter. The MVP pattern is commonly used in Android and iOS app development, as well as in web applications.

How it works:

1.The user interacts with the application through the View.

2.The View sends a request to the Presenter.

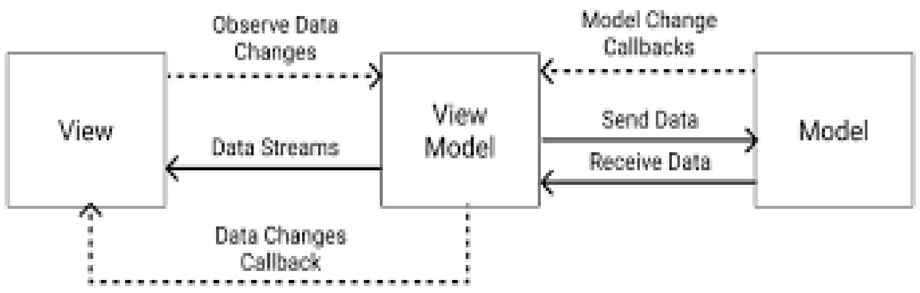
3.The Presenter processes the request and communicates with the Model to perform any necessary actions.

4.The Model updates its state and notifies the Presenter.

5.The Presenter updates the View with the new data from the Model.

6.The View renders the updated UI to the user.

3) **Model-View-View Model (MVVM):**



The MVVM pattern is a variation of the MVC pattern, where the Controller is replaced by a View Model. The MVVM pattern is commonly used in WPF, Silverlight, and Windows Store app development, as well as in web applications.

How it works:

1.The user interacts with the application through the View.

2.The View binds to the View Model, which exposes the data and functionality of the Model.

3.The View Model communicates with the Model to perform any necessary actions.

4.The Model updates its state and notifies the View Model.

5.The View Model updates the View with the new data from the Model.

6.The View renders the updated UI to the user.

ASSIGNMENT-2.2

Given:
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.

E-Commerce Platform: "Shop Easy"

Shop Easy is a hypothetical e-commerce platform that allows users to browse and purchase products from various categories. The platform aims to provide a seamless user experience, scalability, and reliability.

Microservices Architecture:

To achieve scalability and reliability, Shop Easy is designed using a microservices architecture. The platform is divided into several independent services, each responsible for a specific business capability:

Product Service: Manages product information, including product details, pricing, and inventory.

Order Service: Handles order processing, including payment processing and order fulfilment.

User Service: Manages user accounts, including authentication, authorization, and user profiles.

Payment Gateway Service: Integrates with various payment gateways to process payments.

Inventory Service: Manages inventory levels and updates product availability.

Event-Driven Architecture:

To enable loose coupling and scalability, Shop Easy uses an event-driven architecture. Each microservice publishes events to a message broker (e.g., RabbitMQ), which notifies other services of changes. For example:

When a user places an order, the Order Service publishes an "Order Placed" event.

The Product Service listens to the "Order Placed" event and updates the product inventory accordingly.

The Inventory Service listens to the "Order Placed" event and updates the inventory levels.

SOLID Principles:

To enhance the design, SOLID principles are applied:

Single Responsibility Principle (SRP): Each microservice has a single responsibility, reducing complexity and improving maintainability.

Open/Closed Principle (OCP): Microservices are designed to be open for extension but closed for modification, allowing for easy addition of new features without modifying existing code.

Liskov Substitution Principle (LSP): Microservices are designed to be substitutable, ensuring that changes to one service do not affect other services.

Interface Segregation Principle (ISP): Microservices are designed to have minimal interfaces, reducing coupling and improving flexibility.

Dependency Inversion Principle (DIP): Microservices are designed to depend on abstractions, reducing coupling and improving testability.

DRY (Don't Repeat Yourself) Principle:

To observe the DRY principle, Shop Easy:

Uses a shared library for common functionality, such as logging and authentication.

Implements a consistent data model across microservices, reducing data duplication and inconsistencies.

Uses a centralized configuration management system to avoid duplicated configuration settings.

KISS (Keep It Simple, Stupid) Principle:

To observe the KISS principle, Shop Easy:

Uses simple, well-defined interfaces between microservices, reducing complexity and improving maintainability.

Avoids over-engineering and focuses on simplicity and ease of use.

Implements a minimal set of features in each microservice, reducing complexity and improving scalability.

ASSIGNMENT-2.3

Given:
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.

Trends and Cloud Services Overview

Benefits of Serverless Architecture

- **Cost Efficiency**: Pay only for the compute time used, eliminating server provisioning and maintenance costs.
- **Automatic Scaling**: Resources dynamically adjust based on workload, ensuring optimal performance without manual intervention.
- **Development Speed**: Focus on writing code and deploying applications without infrastructure management, leading to quicker release cycles.

The Concept of Progressive Web Apps (PWAs)

- **Web and Mobile Blend**: Combine the best features of web and mobile applications for app-like experiences in the browser.
- **Offline Access**: Utilize service workers and caching strategies for functionality without internet connectivity.
- **Accessibility**: No installation from app stores required, reducing friction for users and enhancing reach.

- **Responsiveness**: Designed to work seamlessly across various devices and screen sizes.

The Role of AI and Machine Learning in Software Architecture

- **Data-Driven Decisions**: Enable applications to learn from data, make predictions, and automate processes.
- **Personalization**: Improve user experience by tailoring content and interactions based on user behavior.
- **Security Enhancements**: Use anomaly detection to enhance security measures.
- **Operational Optimization**: Employ predictive analytics to optimize operations and provide insights from data.

Cloud Computing Service Models

- **SaaS (Software as a Service)**:
  - **Description**: Fully managed software applications delivered over the internet.
  - **Examples**: Google Workspace, Salesforce.
  - **Use Cases**: End-user software access without infrastructure concerns.
- **PaaS (Platform as a Service)**:
  - **Description**: Platform for building, deploying, and managing applications without server management.
  - **Examples**: Microsoft Azure, Google App Engine.
  - **Use Cases**: Development with tools, libraries, and frameworks.
- **IaaS (Infrastructure as a Service)**:
  - **Description**: Fundamental computing resources like virtual machines, storage, and networks.
  - **Examples**: Amazon Web Services (AWS), IBM Cloud.
  - **Use Cases**: Renting scalable infrastructure on a pay-as-you-go basis.