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 Design Pattern

The MVC pattern is a software architecture pattern that divides an application into three interconnected components:

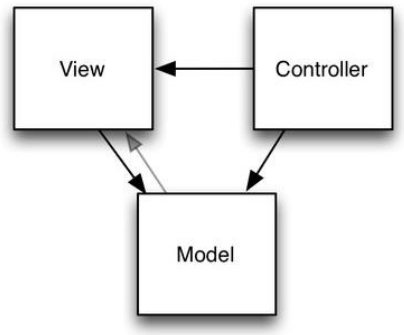
Model: Represents the data and business logic. It manages application data, processes business rules, and responds to requests from other components.

View: Displays data from the Model to users. It's passive and doesn't directly interact with the Model. Instead, it receives data from the Model and forwards user inputs to the Controller.

Controller: Acts as an intermediary between the Model and the View. It handles user input, updates the Model, and ensures the View reflects changes in the Model. The Controller contains application logic.

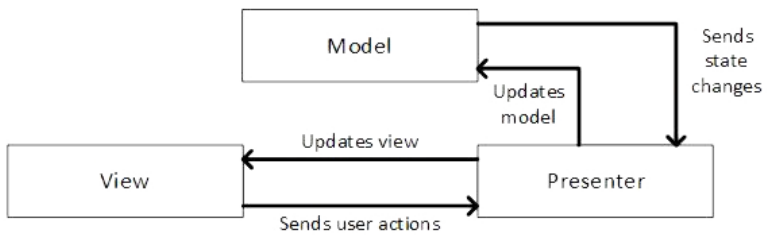
Variants of MVC

Classic MVC:



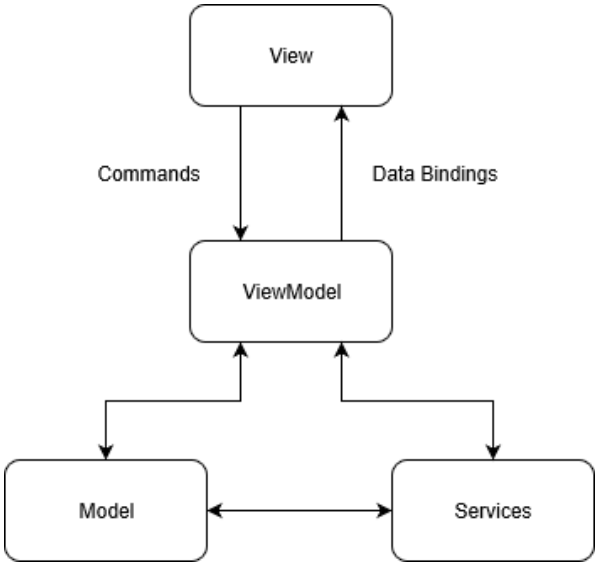
- Each component has a distinct role.
- Suitable for desktop applications.
- Clear separation of concerns.

Model-View-Presenter (MVP):



- Presenter replaces the Controller.
- View interacts directly with the Presenter.
- Common in GUI-based applications.

Model-View-View Model (MVVM):



- View Model replaces the Controller.
- Designed for data-binding frameworks (e.g., WPF, Angular).
- Ideal for rich client applications.

<u>When to Choose Each Variant</u>

Classic MVC:

- Use for traditional desktop applications.
- Strong separation of concerns.
- Well-suited for complex business logic.

MVP:

- Choose for GUI-based applications.
- Simplified testing (due to direct interaction with the Presenter).
- Good for maintaining legacy codebases.

MVVM:

- Select for data-binding frameworks.
- Ideal for rich client applications (e.g., mobile apps).
- Enables seamless UI updates.

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.

Microservices Architecture:

Decomposition into Microservices:

- We break down ShopifyMart into smaller, independent services (microservices). Each microservice handles a specific domain, such as product catalog, inventory management, order processing, and user authentication.
- Benefits:
  - Scalability: We can scale individual services based on demand.
  - Isolation: Failures in one microservice will not affect others.
  - Technology Diversity: Each service can use the most suitable technology stack.

Event-Driven Architecture

Event Sourcing:

- We store all changes to the system's state as a sequence of events.
- **Benefits:**
  - Auditability: We can reconstruct the system's state at any point.
  - Consistency: Events ensure data consistency across services.

CQRS (Command Query Responsibility Segregation):

- Separate read and write models:
  - Command side (writes): Handles updates (e.g., placing an order).
  - Query side (reads): Optimized for querying (e.g., product search).
- Benefits:
  - Scalability: Read and write models can scale independently.
  - Performance: Query side can be optimized for specific use cases.

SOLID Principles

Single Responsibility Principle (SRP):

- Each microservice adheres to SRP by having a single responsibility (e.g., product catalog service manages product data only).
- Benefits:
  - Maintainability: Easier to modify and extend.
  - Testability: Smaller units are easier to test.

Open/Closed Principle (OCP):

- Microservices are open for extension (new features) but closed for modification (existing behaviour remains unchanged).
- Benefits:
  - Avoids breaking existing functionality.
  - Encourages adding new features through extensions.

Liskov Substitution Principle (LSP):

- Microservices' interfaces are designed to be interchangeable without affecting the system's behaviour.
- Benefits:
  - Interoperability: Services can be replaced or upgraded seamlessly.

Interface Segregation Principle (ISP):

- Microservices expose minimal, focused APIs.
- Benefits:
  - Clients only depend on what they need.
  - Avoids bloated interfaces.

DRY and KISS Principles

DRY (Don't Repeat Yourself):

- We avoid duplicating code or logic across microservices.
- Benefits:
  - Maintainability: Changes in one place propagate everywhere.
  - Consistency: Ensures uniform behaviour.

KISS (Keep It Simple, Stupid):

- We prioritize simplicity over complexity.
- Benefits:
  - Easier maintenance.
  - Reduced chances of bugs.

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

The benefits of serverless architecture: In recent years, serverless architecture has gained popularity due to its numerous benefits. By abstracting away infrastructure management, developers can focus solely on writing code, leading to increased productivity and faster time-to-market. Serverless also offers auto-scaling and pay-per-use pricing models, ensuring optimal resource utilization and cost efficiency. Additionally, it promotes event-driven programming paradigms, enabling highly scalable and responsive applications that can effortlessly handle varying workloads.

Progressive Web Apps (PWAs): represent a significant shift in web development, offering a hybrid experience between traditional web pages and native mobile applications. PWAs leverage modern web technologies to deliver fast, reliable, and engaging user experiences across various devices and platforms. With features like offline access, push notifications, and home screen installation, PWAs bridge the gap between web and mobile apps, providing users with seamless interactions and enhancing overall user satisfaction.

The role of AI and Machine Learning in software architecture: Artificial Intelligence (AI) and Machine Learning (ML) are revolutionizing software architecture by enabling intelligent, data-driven applications. AI-powered systems can analyse vast amounts of data, uncover valuable insights, and make informed decisions in real-time. ML algorithms are integrated into various aspects of software architecture, from predictive analytics to natural language processing, enhancing personalization, automation, and efficiency. AI and ML technologies empower developers to create smarter, more adaptive applications that continuously learn and improve over time.

Cloud Computing Service Models

Cloud computing offers three primary service models: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). SaaS provides ready-to-use applications hosted and maintained by a third-party provider, making it ideal for organizations seeking to offload software management tasks. PaaS offers development and deployment platforms with tools and services for building, testing, and managing applications, empowering developers to focus on coding without worrying about infrastructure management. IaaS provides virtualized computing resources, including servers, storage, and networking, allowing organizations to build and manage their own IT infrastructure in the cloud, providing maximum control and flexibility.