**Day 2 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.

**MVC Design Pattern Overview:**

The Model-View-Controller (MVC) design pattern is a widely used architectural pattern in software engineering for implementing user interfaces. It separates an application into three interconnected components: Model, View, and Controller.

**1.MODEL:** Represents the data and business logic of the application. It manages the data, responds to queries about the state of the application, and manipulates that data when instructed to do so by the controller or view.

**2.VIEW:** Represents the presentation layer of the application. It is responsible for displaying the data provided by the model in a user-friendly format. Views can range from simple UI elements to complex graphical interfaces.

**3.CONTROLLER**: Acts as an intermediary between the model and the view. It handles user input, processes requests from the view, updates the model accordingly, and sends the updated data back to the view for display.

**MVC Variants:**

**1.Classic MVC :** In the classic MVC variant, the model notifies the view and controller whenever there is a change in its state. Both the view and controller are observers of the model. This variant emphasizes the unidirectional flow of data from the model to the view and controller.

**Classic MVC Diagram:**

**MODEL**

|

VIEW CONTROLLER

**2. Model-View-Presenter (MVP):** In the MVP variant, the presenter acts as an intermediary between the model and the view. Unlike classic MVC, the view is more passive and delegates user input handling to the presenter. The presenter updates the view directly with the data from the model.

**MVP Diagram:**

**MODEL**

**|**

**PRESENTER**

**|**

**VIEW**

**3. Model-View-View Model (MVVM):** In the MVVM variant, the view model acts as an abstraction of the view. It exposes data and commands from the model to the view through data binding. The view model allows for easy testing and separation of concerns by keeping the view logic separate from the UI.

**MVVM Diagram:**

**MODEL**

**|**

**VIEWMODEL**

**|**

**VIEW**

**Choosing the Right Variant:**

* **Classic MVC**: Suitable for smaller applications with straightforward user interfaces where data flow is predictable.
* **MVP**: Ideal for applications requiring extensive user input handling or complex UI logic. It facilitates easier testing and decouples the view from the business logic.
* **MVVM**: Best suited for applications with rich user interfaces that require frequent updates or dynamic content. It promotes separation of concerns and facilitates easier maintenance and testing.

**DAy 2 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.

**Scenario: Applying Microservices and Event-Driven Architecture to an E-commerce Platform**

**Background:**

You're tasked with redesigning an existing monolithic e-commerce platform to improve scalability, resilience, and flexibility. The goal is to adopt Microservices Architecture and Event-Driven Architecture to achieve these objectives.

**Architecture Overview:**

1. **Microservices Architecture:**

Breaking down the monolithic application into smaller, independent services that can be developed, deployed, and scaled independently.

1. **Event-Driven Architecture**:

Decoupling components by enabling communication through asynchronous events, allowing for greater flexibility and scalability.

Design Considerations:

**Microservices:** Each service focuses on a specific business domain, such as product catalogue, order management, payment processing, user authentication, etc.

**Event-Driven:** Services communicate via events, such as "order placed," "product added," or "payment processed," allowing for loosely coupled interactions.

**Application of SOLID Principles:**

* **Single Responsibility Principle (SRP)**:
  + Each microservice has a single responsibility related to its business domain.
  + For example, the order management service handles order creation, modification, and retrieval.
* **Open/Closed Principle (OCP)**:
  + Services are designed to be open for extension but closed for modification.
  + New functionalities are added by creating new services or extending existing ones without modifying the core functionality.
* **Liskov Substitution Principle (LSP)**:
  + Services adhere to common interfaces, allowing them to be substituted without affecting the behaviour of the system.
  + For instance, different payment gateway services can be seamlessly integrated as long as they adhere to the payment service interface.
* **Interface Segregation Principle (ISP)**:
  + Interfaces are tailored to the specific needs of the services that use them.
  + For instance, the product catalogue service may expose a different set of APIs compared to the user authentication service.
* **Dependency Inversion Principle (DIP)**:
  + Services depend on abstractions rather than concrete implementations, promoting loose coupling and easier testing.
  + For example, services interact with each other through event-driven messaging systems rather than direct API calls.

**Observing DRY and KISS Principles:**

* **Don't Repeat Yourself (DRY)**:
  + Reusable components, such as authentication services, are implemented once and shared across multiple microservices.
  + Common functionalities like logging, error handling, and database access are encapsulated into shared libraries.
* **Keep It Simple, Stupid (KISS)**:
  + Each microservice is designed to be simple and focused on its specific task.
  + Complex functionalities are broken down into smaller, manageable units, making the system easier to understand and maintain.

**Conclusion:** By adopting Microservices Architecture and Event-Driven Architecture principles while adhering to SOLID, DRY, and KISS principles, the e-commerce platform becomes more scalable, resilient, and maintainable. The modular and loosely coupled nature of the architecture allows for easier development, deployment, and evolution of the system over time.

Top of Form

**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.**

**Trends and Cloud Services Overview:**

**Server less** architecture offers significant benefits in terms of scalability, cost-efficiency, and reduced operational overhead. By abstracting away the infrastructure management, developers can focus more on writing code and delivering value to users. Serverless platforms automatically scale resources based on demand, ensuring optimal performance without the need for manual intervention. Additionally, since serverless services are billed based on usage, organizations can achieve cost savings by paying only for the resources consumed during execution.

Progressive Web Apps (PWAs) represent a new approach to delivering web applications that offer a native app-like experience across various devices and platforms. PWAs leverage modern web technologies like Service Workers and Web App Manifests to enable features such as offline access, push notifications, and installation to the home screen. This approach enhances user engagement and retention by providing fast, reliable, and immersive experiences, even in low-network conditions. Moreover, PWAs eliminate the need for app store distribution, simplifying the deployment process for developers.

AI and Machine Learning are increasingly playing a vital role in software architecture by enabling intelligent decision-making, automation, and personalization. These technologies empower applications to analyse vast amounts of data, extract valuable insights, and adapt to user behaviour in real-time. AI-driven systems can enhance user experiences by providing personalized recommendations, automating repetitive tasks, and detecting anomalies or security threats. Integrating AI and Machine Learning into software architecture enables organizations to unlock new opportunities, improve operational efficiency, and stay competitive in today's data-driven landscape.

**Cloud Computing Services Models Overview:**

Cloud computing offers three primary service models: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS), each catering to different use cases and requirements. SaaS delivers fully functional applications over the internet, allowing users to access software without the need for installation or maintenance. PaaS provides developers with a platform and tools to build, deploy, and manage applications without worrying about underlying infrastructure. IaaS offers virtualized computing resources, including servers, storage, and networking, enabling organizations to deploy and manage their own applications and workloads while retaining control over the underlying infrastructure.