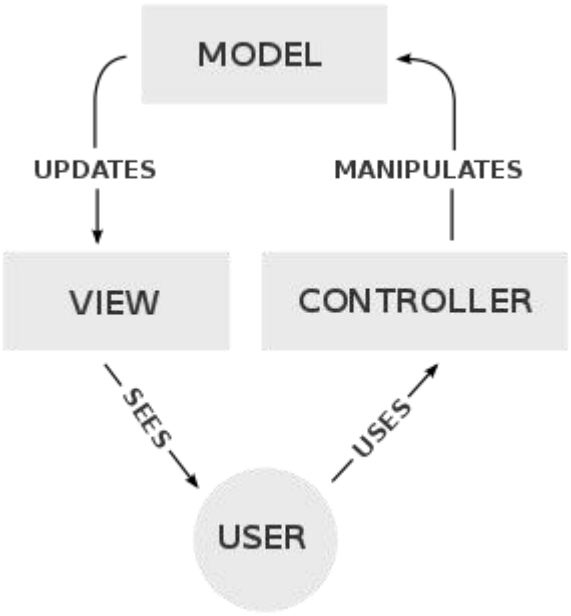
DAY 2
Assignment 1 :

MVC (Model view controller)



The Model-View-Controller (MVC) design pattern is a software architectural pattern used to separate an application into three interconnected components: the Model, the View, and the Controller. This separation enhances the modularity and maintainability of the application by isolating each component's responsibilities.

In the Classic MVC variant, the Model represents the application's data and business logic. The View is responsible for presenting the data to the user, typically through a user interface. The Controller acts as an intermediary between the Model and the View, handling user inputs and updating the Model accordingly.

- When to use:

Suitable for desktop applications or web applications with complex user interfaces. Provides a clear separation of concerns, making it easier to maintain and scale the application.

Two variants:

1. Model-View-Presenter (MVP):

In the MVP variant, the Presenter replaces the Controller found in Classic MVC. The Presenter is responsible for handling user inputs, updating the Model, and updating the View. Unlike the Controller, the Presenter does not directly manipulate the View; instead, it communicates with the View through an interface.

- When to use:

Suitable for applications with complex user interfaces where the logic for presenting data might be intricate.
Facilitates unit testing as the Presenter can be easily tested in isolation from the View.

2. Model-View-ViewModel (MVVM):

MVVM is a variant of MVC commonly used in client-side applications, particularly in frameworks like Angular, React, and Vue.js. The ViewModel replaces both the Controller and the Presenter found in Classic MVC and MVP, respectively. The ViewModel encapsulates the View's state and behavior, exposing data and commands that the View can bind to.

- When to use:

Ideal for applications with rich client-side interactions, such as single-page web applications.
Promotes a more declarative and reactive programming style, enhancing code maintainability and readability.

Assignment 2:

Scenario: Implementing Microservices and Event-Driven Architecture in an E-commerce Platform

In our hypothetical e-commerce platform, we are transitioning from a monolithic architecture to a microservices-based architecture coupled with event-driven principles to improve scalability, maintainability, and agility.

- Microservices Architecture Implementation:

1. User Management Service: Manages user authentication, registration, and profile data.
2. Product Catalog Service: Handles product information, including CRUD operations.
3. Order Management Service: Responsible for processing orders, managing carts, and handling payments.
4. Inventory Service: Tracks inventory levels and updates product availability.
5. Shipping Service: Manages shipping logistics and updates order status.
6. Notification Service: Sends out email and SMS notifications to users regarding order status and promotions.

- Event-Driven Architecture Implementation:

1.User Management Events: Emit events for user registration, authentication, and profile updates.
2.Product Events: Triggered upon product creation, modification, or deletion.Order Events: Generated when orders are placed, updated, or canceled.
3.Inventory Events: Broadcasted when inventory levels change.
4.Shipping Events: Dispatched upon successful shipment of orders.
5.Notification Events: Fired to notify users about order status changes or promotional offers.

SOLID Principles Integration:

1. Single Responsibility Principle (SRP):
2. Each microservice has a single responsibility, such as user management or order processing, ensuring they are focused and maintainable.
3. Open/Closed Principle (OCP):
4. Services are designed to be open for extension (e.g., adding new features) but closed for modification, minimizing the risk of unintended side effects.
5. Liskov Substitution Principle (LSP):
6. Subservices within each microservice can be substituted interchangeably without affecting the correctness of the system, ensuring modularity and flexibility.
7. Interface Segregation Principle (ISP):
8. Service interfaces are tailored to the specific needs of clients, preventing unnecessary dependencies and coupling.
9. Dependency Inversion Principle (DIP):
10. High-level modules (e.g., controllers) depend on abstractions (interfaces) rather than concrete implementations, promoting decoupling and testability.

DRY (Don't Repeat Yourself) Principle:

- Reusable components such as authentication middleware or database connectors are implemented once and shared across microservices.
- Common functionalities like logging or error handling are abstracted into shared libraries to avoid duplication of code.

KISS (Keep It Simple, Stupid) Principle:

- Each microservice is designed to solve a specific business problem without unnecessary complexity.
- Service interactions are kept straightforward and asynchronous through the use of messaging queues or event brokers.

Assignment 3 :

Trends and Cloud Services Overview:

1. Serverless architecture offers numerous benefits, primarily by abstracting away the underlying infrastructure management tasks from developers. This results in reduced operational overhead, as cloud providers handle scaling, availability, and maintenance automatically. Additionally, serverless enables a pay-per-use pricing model, allowing organizations to optimize costs by only paying for the resources consumed during execution. Furthermore, serverless promotes rapid development and deployment cycles, fostering agility and innovation in software development practices.

2. Progressive Web Apps (PWAs) represent a modern approach to web development, combining the best features of web and native applications. PWAs provide a seamless user experience by leveraging technologies such as service workers, enabling offline functionality, push notifications, and fast load times. By bypassing the need for app store distribution, PWAs offer greater reach and accessibility across various devices and platforms. Moreover, PWAs improve engagement and retention rates through features like home screen installation and background sync capabilities.

3. AI and Machine Learning are increasingly shaping software architecture by enabling intelligent decision-making, automation, and personalized user experiences. AI-driven systems can analyze vast amounts of data to derive insights, predict trends, and optimize processes in real-time. Machine Learning algorithms empower applications to adapt and learn from user interactions, leading to more context-aware and intuitive experiences. From recommendation engines in e-commerce platforms to predictive maintenance in IoT devices, AI and Machine Learning are revolutionizing software architecture across diverse domains.

Cloud Computing Service Models

- Software as a Service (SaaS): SaaS delivers software applications over the internet on a subscription basis, eliminating the need for organizations to manage software installation, maintenance, and updates. Use cases include email services like Gmail, collaboration tools like Google Workspace, and customer relationship management (CRM) systems like Salesforce.

- Platform as a Service (PaaS): PaaS provides a platform allowing customers to develop, run, and manage applications without dealing with the underlying infrastructure. Developers can focus on building and deploying applications, while the cloud provider manages servers, storage, and networking. Use cases

include web application development platforms like Heroku, database services like Amazon RDS, and container orchestration platforms like Google Kubernetes Engine (GKE).

- Infrastructure as a Service (IaaS): IaaS offers virtualized computing resources over the internet, including virtual machines, storage, and networking. Users have full control over the operating systems and software running on the virtual machines. IaaS enables organizations to scale resources dynamically, paying only for what they use. Use cases include hosting websites and web applications, running development and testing environments, and disaster recovery solutions.

Reference : https://chatgpt.com/c/81e9a47e-f81f-4ad6-8343-805af5648000

- https://www.geeksforgeeks.org/solid-principle-in-programming-understand-with-real-life-examples