Q) 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.

Sol)

Model-View-Controller:-Model–view–controller (MVC) is a software design patterncommonly used for developing user interface that divides the related program logic into three interconnected elements. These elements are the internal representations of information the interface that presents information to and accepts it from the user, and the controller software linking the two.

MVC Components:-

* Model
* View
* Controller

Model:-The Model component corresponds to all the data-related logic that the user works with. This can represent either the data that is being transferred between the View and Controller components or any other business logic-related data.

View:-The View component is used for all the UI logic of the application

Controller:-Controllers act as an interface between Model and View components to process all the business logic and incoming requests, manipulate data using the Model component and interact with the Views to render the final output.

The variations of MVC:-MVC is highly versatile and can be adapted to different scenarios and requirements. MVP (Model-View-Presenter) is a popular variation that sees the presenter as a mediator between the model and the view. The presenter updates the view with data from the model and handles user events and action.

* MVVM (Model-View-View Model),
* HMVC (Hierarchical Model-View-Controller).

MVVM (Model-View-View Model):-

Model-View-View Model (MVVM) is a design pattern that helps you separate your application logic from the user interface.

It is an architectural software design pattern in which the model represents the data, the view is what the user sees and interacts with, and the view model acts as a mediator between the model and view.

MVVM is a great pattern to use when building applications. It helps to separate the business logic from the UI, which makes it much easier to test your application**.**

Components of MVVM:-The Model-View-View Model (MVVM) pattern is a separation of concerns that helps you cleanly separate your application into three main components: 1) Model,

2) View,

3) View Model.

Model:-The model represents the data in your application. The model is generally a domain model, which can be an object oriented representation of your business domain. This would normally be a collection of objects, properties and methods.

View:-The view represents what the user sees on the screen. View controls and implements visual behaviour of application. It takes keyboard input from the user and provides feedback to them.

View model:-The view model is responsible for providing data from the model to the view (the user interface). It may also be responsible for handling any interactions between controls on different pages of your application.

The view model is the brain of your application. It’s responsible for providing data from the model (a collection of objects, properties and methods) to the view (the user interface)

Benefits of mvvm:-

It allows you to focus on one aspect of a problem at a time, which makes it easier to reason about complex systems. MVVM helps you separate concerns by keeping your UI code separate from your business logic code.

You can replace one piece of code without affecting another. For example, if you want to update

MVVM helps you write reusable code because you can break down your app into smaller chunks that are easier to maintain

Limitations of mvvm:-

The complexity of using MVVM can be one of its biggest drawbacks. It’s important to understand how the pattern works, and this takes time.

When developers add features or functionality to their applications that are unnecessary or don’t meet user needs. This can lead to applications that are bloated, slow and difficult to use.

Applications of mvvm:-

Angular: [Angular](https://angular.io/) is a popular front-end framework that implements MVVM architecture.

React: [React](https://react.dev/) is a popular JavaScript library that can also implement MVVM architecture using libraries such as MobX or Redux

HMVC (Hierarchical Model-View-Controller):-

This is the MVC model repeated over and over again for the parts of your application. And this allows is for parts of the application to be used again elsewhere as each controller can communicate with another controller.

This style of architecture is to modulate the application into smaller more manageable parts that can work independently.

It will also be easier to expand the work to be added to the application due to the capability of controllers calling to each other when a new part is created.

HMVC adds another layer of abstraction by allowing each controller to have its own sub-controllers and sub-views, forming a hierarchy of modules

Benefits of HMVC:-

Modularity: HMVC enables programmers to write testable, maintainable code that is modular. Each module may be created individually, and modifications to one module do not affect the other modules.

Code reuse: HMVC makes it simple to reuse code across many modules. Model, View, and Controller components that developers may reuse in other modules can be created.

Easy maintenance: By segmenting the code into modules, HMVC makes application maintenance simple. Each module may be tested individually, and modifications to one module do not affect the other modules.

Limitations of HMVC:-

One of the challenges is that HMVC can increase the learning curve and the code complexity, especially for beginners or developers who are not familiar with the pattern.

Another challenge is that HMVC can introduce some overhead and inefficiency, especially for simple or small applications that do not need the extra layer of abstraction

And third challenge is that HMVC can make testing and debugging more difficult, as you need to isolate and verify the behaviour of each module and its interactions with other modules or the main application

Applications of HMVC:-

Dashboard page with various widgets such as charts, tables, and notifications can be a module with its own controller, model, and view.

 Blog site with sections like posts, comments, categories, and tags can be a module with its own controller, model, and view that can be reused and integrated into different pages or contexts by using sub-requests and partial views.

Shopping cart system with features like products, orders, payments, or shipping can be a module with its own controller, model, and view that can communicate and coordinate with other modules or the main controller by using sub-requests and partial views.

Q) Draft a one-page scenario where you apply Micro services 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.

A microservices architecture is a type of application architecture where the application is developed as a collection of services. It provides the framework to develop, deploy, and maintain microservices architecture diagrams and services independently.

Within a microservices architecture, each microservice is a single service built to accommodate an application feature and handle discrete tasks. Each microservice communicates with other services through simple interfaces to solve business problems.

It is an architectural style that structures an application as a collection of services that are:

independently deployed

loosely coupled

Benefits of Micro services Architecture:-

Agility. Because microservices are deployed independently, it's easier to manage bug fixes and feature releases. You can update a service without redeploying the entire application, and roll back an update if something goes wrong

Small, focused teams. A microservice should be small enough that a single feature team can build, test, and deploy it

Mix of technologies. Teams can pick the technology that best fits their service, using a mix of technology stacks as appropriate.

Challenges of Micro services Architecture:-

Complexity. A microservices application has more moving parts than the equivalent monolithic application. Each service is simpler, but the entire system as a whole is more complex.

Data integrity. With each microservice responsible for its own data persistence. As a result, data consistency can be a challenge. Embrace eventual consistency where possible.

Management. To be successful with microservices requires a mature DevOps culture. Correlated logging across services can be challenging. Typically, logging must correlate multiple service calls for a single user operation.

* DRY is a principle that aims to avoid duplication of code. Duplication can lead to inconsistencies, errors, and extra work when you need to change or update your code
* DRY encourages you to extract common logic, data, or functionality into reusable modules, functions, or variables
* To apply dry you need to identify and eliminate any repetition in your code. For example, if you have multiple functions that perform the same or similar tasks, you can refactor them into one generic function that takes different parameters
* KISS is a principle that aims to avoid complexity in code. Complexity can make your code harder to understand, test, and modify
* KISS encourages you to write code that is clear, concise, and self-explanatory
* To apply kiss, you need to simplify and clarify your code. For example, if you have a long or nested function, you can break it down into smaller and simpler functions that have descriptive names

 A micro service architecture must follow these five principles:-

* Single concern
* Discrete
* Transportable
* Carries its own data
* Inherently ephemeral

Event-Driven Architecture to a hypothetical e-commerce platform:-

Event driven architecture (EDA) builds on composable commerce’s ability to add features or swap out applications seamlessly, and further accelerates and automates commerce processes by supporting truly real-time, event-driven actions.

With EDA, a time-sensitive ‘event’ (such as a customer placing an order, stock-level notifications, delivery updates, or payments) is used as the trigger for applications and services in your commerce architecture to talk to each other.

EDA enables business flows to be dynamically orchestrated in real time, based on each specific event and its requirements.

With an EDA, services such as the E-Commerce site, order validation, stock management, payment processing, and fulfillment can be from multiple different vendors to suit the organization’s needs.  Each service produces and consumes data on new events (such as Jon placing his order), and the event data is filtered and pushed to the relevant services automatically by an event router.

Single Responsibility Principle (SRP):-

Each microservice has a single responsibility related to a specific business domain, ensuring that changes to one part of the system do not affect others. For example, the user management service handles user authentication, authorization, and profile management, adhering strictly to SRP.

Open/Closed Principle (OCP):-

Services are designed to be open for extension but closed for modification. New functionalities are added through the introduction of new services or by extending existing ones, rather than modifying the existing codebase. For instance, introducing a new payment gateway is facilitated by creating a new payment service that integrates seamlessly with the existing system without modifying the order processing service.

Liskov Substitution Principle (LSP):-

Interchangeability among microservices is ensured by adhering to common interfaces and contracts. Any service that adheres to the interface contract can seamlessly replace another service without impacting the overall system behaviour. For example, any recommendation engine service that adheres to the defined interface can be plugged into the system to provide personalized product recommendations.

Interface Segregation Principle (ISP):-

Services expose specific interfaces tailored to the needs of their clients, promoting loose coupling and ensuring that clients only depend on the interfaces they require. This prevents clients from being forced to depend on methods they do not use, reducing complexity and potential side effects. For instance, the product catalog service exposes separate interfaces for browsing products, retrieving product details, and managing inventory, catering to different client needs.

Dependency Inversion Principle (DIP):-

High-level modules such as the user interface and business logic depend on abstractions rather than concrete implementations of lower-level modules. This inversion of dependencies decouples high-level and low-level modules, allowing for easier unit testing, flexibility, and interchangeability. For instance, the order processing service depends on an abstract payment interface, enabling it to work with various payment service implementations without being tightly coupled to any specific one.

The benefits of implementing event driven architecture:-

* Decoupling allows for faster development cycles, as each service in the architecture can be developed independently from the other.
* Each service can be scaled independently to meet the organization’s specific needs.
* Response times for each service are faster, as they operate independently from each other
* .The DRY principle, an acronym for "Don't Repeat Yourself", emphasizes the importance of avoiding code duplication in our applications. The fundamental idea behind DRY is to have a single, authoritative representation of knowledge within our system. By eliminating duplicated code, we can enhance code maintainability, reduce complexity, and minimize the risk of introducing bugs.
* DRY extends beyond literal code duplication. It also encompasses the duplication of knowledge and intent. In other words, it's about expressing the same concept in multiple places, potentially in different ways.
* One way to implement the DRY principle is by utilizing methods. By extracting commonly repeated code into methods, we can centralize its logic and avoid duplicating it across our application.
* The KISS principle, which stands for "Keep It Simple, Stupid", emphasizes the importance of simplicity in software design and development. The goal is to prioritize straightforward solutions over complex ones. By keeping our code simple, we can enhance comprehensibility, usability, and maintainability.
* Adhering to the KISS principle offers several advantages in the context of digital product development.
* Simpler software structures facilitate testing, including automated testing. With less complexity, testing becomes easier and more effective, leading to higher code quality.

Q) 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?

Serverless architecture:-

Serverless computing is a cloud architecture that allows organizations to get on-demand access to computing resources. Customers only pay for the resources they use. Resources are not allocated to an application when it is not in use.

In a serverless computing architecture, a server’s code execution is fully managed by the cloud provider. Therefore, the provider’s customers do not need to develop and deploy the underlying infrastructure that would traditionally be required to run applications and programs. The primary objective of serverless computing is to make it easier for software developers to create code that is intended to run on cloud platforms and perform a clearly defined role.

Benefits of Serverless Architecture:-

Lower costs: Serverless computing architecture is a highly cost-effective process that removes the need for expensive underlying cloud architecture and hardware.

Simplified backend code: Serverless computing enables developers to use simplified backend code that performs a single purpose, such as making a call to application programming interface

Improved security: Serverless computing constrains developers to only using code that works within a serverless context

Limitations of Serverless Computing:-

Serverless architecture can be difficult for organizations to secure because of its distributed nature. This makes the architecture more flexible and scalable, which renders traditional security solutions ineffective

Serverless computing also presents visibility issues as developers take the lead, which can result in applications being pushed to production before being addressed by security teams

Over-privileged functions: This is when a function has more permissions than it needs. In this way, it can be abused to attack key systems or exfiltrate or corrupt data.

Poisoning the well**:** This is when attackers try to incorporate malicious code into widely used projects.

Progressive Web App:-

A progressive web app (PWA) is a website that looks and behaves as if it is a mobile app.

PWA content is downloaded progressively, which provides the end user with a better user experience than a traditional website that uses responsive design.

A progressive web app (PWA) is an app that's built using web platform technologies, but that provides a user experience like that of a platform-specific app.

Like a website, a PWA can run on multiple platforms and devices from a single codebase., it can be installed on the device, and can integrate with the device and with other installed apps.

Benefits of PWA:-

They are responsive and work with many different screen sizes.

They work offline unlike other sites.

Low maintenance cost.

The updates are independent, you don't need to visit the play store for an update.

Limitations of Progressive Web Apps (PWA):-

Since progressive web apps are a relatively new concept, they are not compatible with legacy devices that run outdated browsers.

Despite their cross-platform capabilities, PWAs are essentially Android-centric. They cannot run well with other browsers such as Edge and Safari

Their novelty is one of the concerns for various issues. Many hardware and software platforms do not still support all functionalities of PWA.

The role of AI and Machine Learning in software architecture:

AI is a branch of computer science which is concerned with the study and creation of computer systems that exhibit some form of intelligence

AI is a broad area consisting of different fields, from machine vision, expert systems to the creation of machines that can "think".

Characteristics of AI systems:-

Understand a natural language or perceive and comprehend a visual scene.

Plan sequences of actions to complete a goal.

May not necessarily imitate human senses and thought processes.

Capable of performing intelligent tasks effectively and efficiently.

Ways AI & Ml is bound to change how architecture:-

AI can reduce the time needed to arrive at a design to mere hours or even minutes. Architects will soon be able to plug into a huge, constantly-updated database of designs and codes, letting technology generate design variations for the client in real-time.

Planning and managing cities, towns, and urban regions is a complex undertaking, and it’s one that’s greatly benefiting from advancements in AI technology. Machine learning can automatically identify the most optimal routes for utilities, which then get updated in real-time as the urban design goes through revisions.

s. Integrating AI and ML into software architecture empowers organizations to stay competitive in today's data-driven landscape, delivering intelligent solutions that adapt and evolve with changing requirements.

The cloud computing service models (SaaS, PaaS, IaaS) and their use cases:-

Cloud computing is the delivery of computing services—including servers, storage, databases, networking, software, analytics, and intelligence—over the Internet (“the cloud”) to offer faster innovation, flexible resources, and economies of scale.

**Types of cloud services: IaaS, PaaS, serverless, and SaaS:-**

Infrastructure as a Service (IaaS):-

IaaS contains the basic building blocks for cloud IT. It typically provides access to networking features, computers (virtual or on dedicated hardware), and data storage space. IaaS gives you the highest level of flexibility and management control over your IT resources. It is most similar to the existing IT resources with which many IT departments and developers are familiar.

Platform as a Service (PaaS):-

PaaS removes the need for you to manage underlying infrastructure (usually hardware and operating systems), and allows you to focus on the deployment and management of your applications. This helps you be more efficient as you don’t need to worry about resource procurement, capacity planning, software maintenance, patching, or any of the other undifferentiated heavy lifting involved in running your application.

Software as a Service (SaaS):-

SaaS provides you with a complete product that is run and managed by the service provider. In most cases, people referring to SaaS are referring to end-user applications (such as web-based email). With a SaaS offering, you don’t have to think about how the service is maintained or how the underlying infrastructure is managed. You only need to think about how you will use that particular software.