1. MVC (Model–View–Controller) Design Pattern

Definition: MVC is a widely used software architectural pattern that separates an application into three main components — Model, View, and Controller — to enhance scalability, maintainability, and testability.

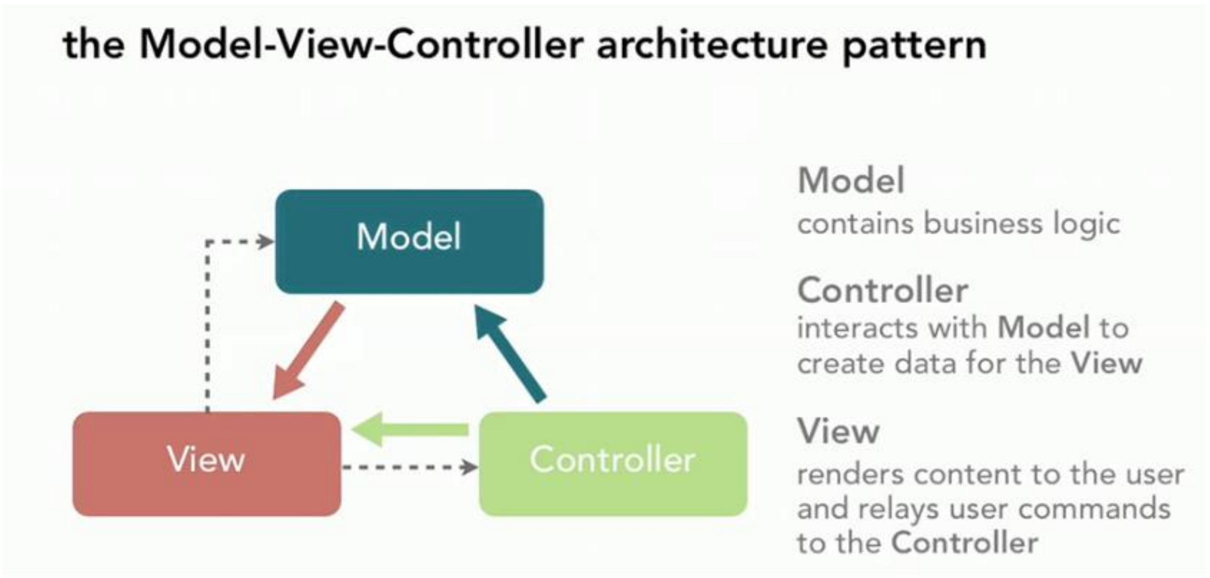
Components:

- **Model**: Manages application data and business logic.
- **View**: Handles the UI and displays data from the Model.
- **Controller**: Responds to user inputs, processes them, and updates the Model or View accordingly.

Flow:
User → View → Controller → Model → View (updated)

Example:
In a blog web app, the View shows posts, the Controller handles post submissions, and the Model stores the post data.

Diagram:


the Model-View-Controller architecture pattern
Model
View
Controller
Model
contains business logic
Controller
interacts with Model to create data for the View
View
renders content to the user and relays user commands to the Controller

2. MVP (Model–View–Presenter)

Definition:
An evolution of MVC, MVP introduces a **Presenter** that contains presentation logic and communicates directly with both the Model and View.

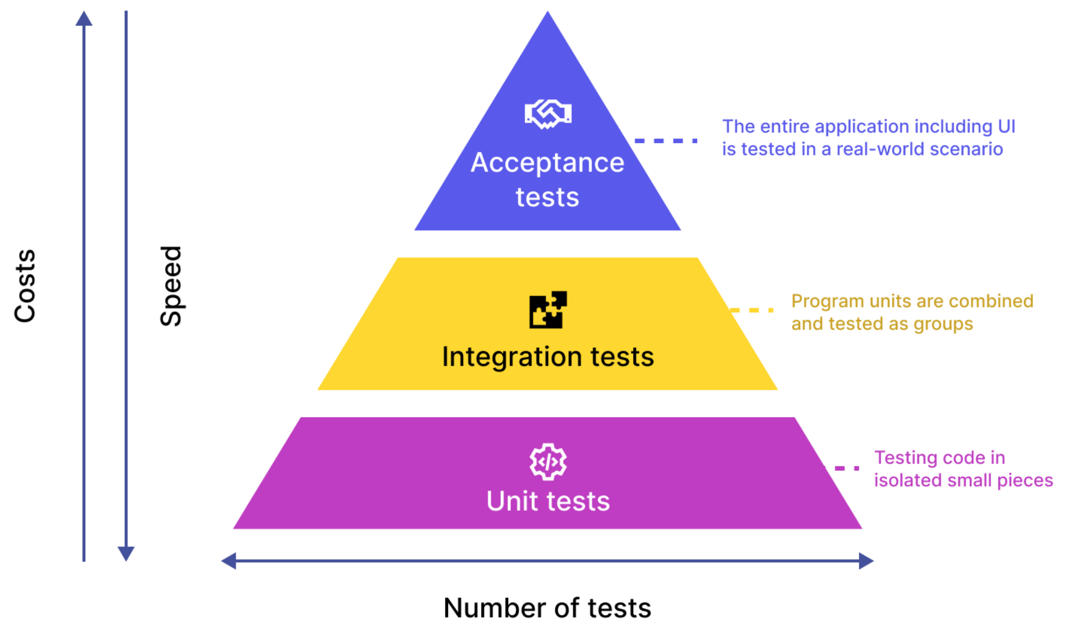
Key Points:

- The View is **passive** — it only displays data and delegates events to the Presenter.
- The Presenter fetches/updates data via the Model and updates the View.

Flow:
User → View → Presenter → Model
Presenter → View (updated)

Use Case:
Great for applications where **unit testing and separation of concerns** are crucial, like in **Android or desktop apps**.

Diagram:


Costs
Speed
Acceptance tests
The entire application including UI is tested in a real-world scenario
Integration tests
Program units are combined and tested as groups
Unit tests
Testing code in isolated small pieces
Number of tests

3. MVVM (Model–View–ViewModel)

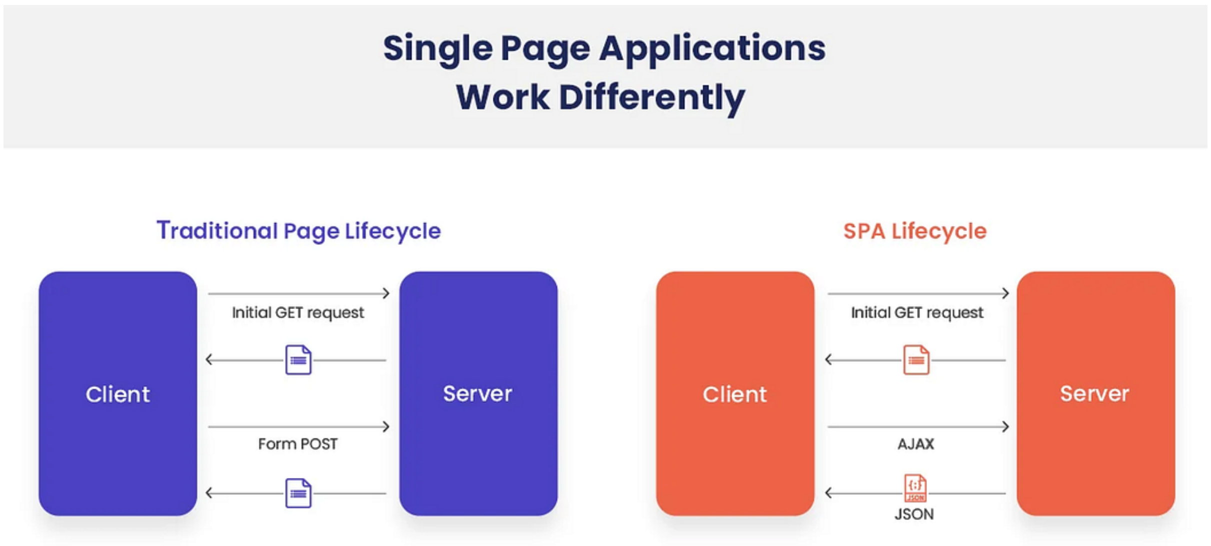
Definition:
MVVM is ideal for reactive applications. It introduces a **ViewModel** that exposes observable data objects for the View to bind to, enabling automatic updates.

Key Points:

- The **ViewModel** acts as a mediator between the View and Model.
- Uses **two-way data binding** to auto-update the View when data changes.

Flow:
View ⇄ ViewModel ⇄ Model
Use Case:
Perfect for **Single Page Applications (SPAs)** and **frameworks with data binding**, like **Angular, React, or WPF**.

Diagram:



Comparison and When to Use

Pattern	**Best For**	**View Type**	**Binding**	**Testability**
MVC	Traditional web apps	Active/Passive	Manual	Moderate
MVP	Desktop/mobile apps	Passive	Manual	High
MVVM	Reactive UIs (SPAs)	Passive	Automatic (two-way)	High

Name:- Anil Gupta
Reg. No.:- 2141007073