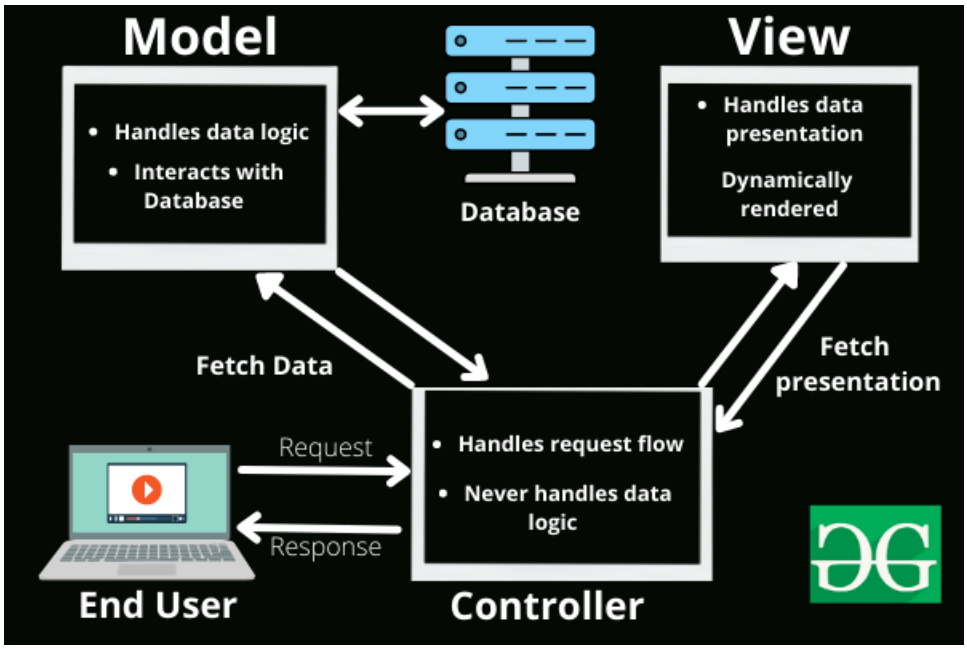
System Architecture

Notes by Mannan Ul Haq

MVC Architecture

The **Model-View-Controller (MVC)** framework is an architectural/design pattern that separates an application into three main logical components **Model**, **View**, and **Controller**. Each architectural component is built to handle specific development aspects of an application. It isolates the business logic and presentation layer from each other. It was traditionally used for desktop **graphical user interfaces (GUIs)**. Nowadays, MVC is one of the most frequently used industry-standard web development frameworks to create scalable and extensible projects. It is also used for designing mobile apps.



1. Model

The Model component in the MVC (Model-View-Controller) design pattern represents the data and business logic of an application. It is responsible for managing the application's data, processing business rules, and responding to requests for information from other components, such as the View and the Controller.

2. View

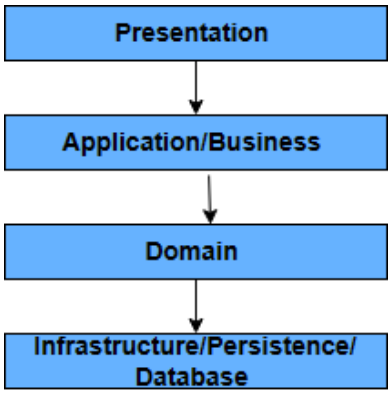
Displays the data from the Model to the user and sends user inputs to the Controller. It is passive and does not directly interact with the Model. Instead, it receives data from the Model and sends user inputs to the Controller for processing.

3. Controller

Controller acts as an intermediary between the Model and the View. It handles user input and updates the Model accordingly and updates the View to reflect changes in the Model. It contains application logic, such as input validation and data transformation.

Layered Architecture

Layered architectures are said to be the most common and widely used architectural framework in software development. It **describes an architectural pattern composed of several separate horizontal layers that function together as a single unit of software**.



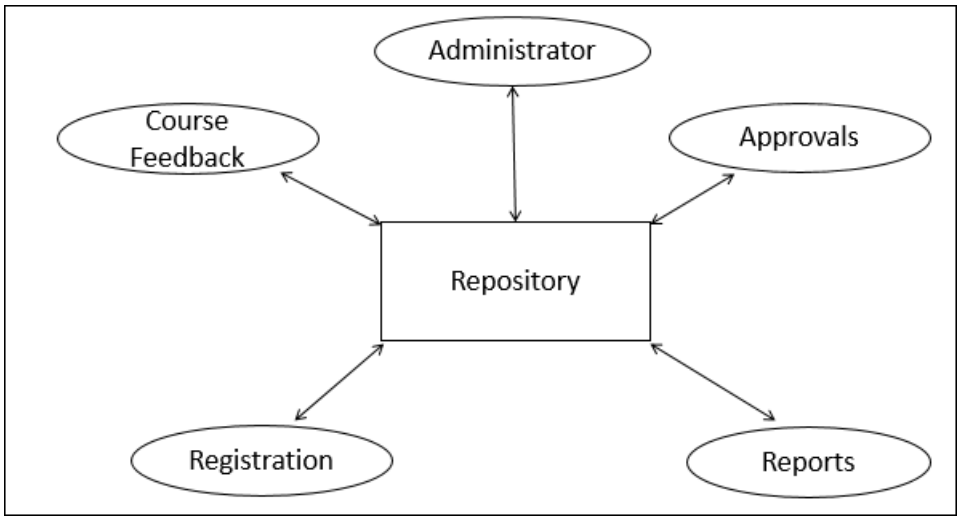
- **Presentation Layer** – responsible for user interactions with the software system
- **Application/Business Layer** – handles aspects related to accomplishing functional requirements
- **Domain Layer** – responsible for algorithms, and programming components
- **Infrastructure/Persistence/Database Layer** – responsible for handling data, databases

A major characteristic of this framework is that layers are only connected to the layers directly below them.

Another characteristic is the concept of **layers of isolation**. **This means that layers can be modified and the change won't affect other layers**. In short, changes are isolated to the specific layer that is altered.

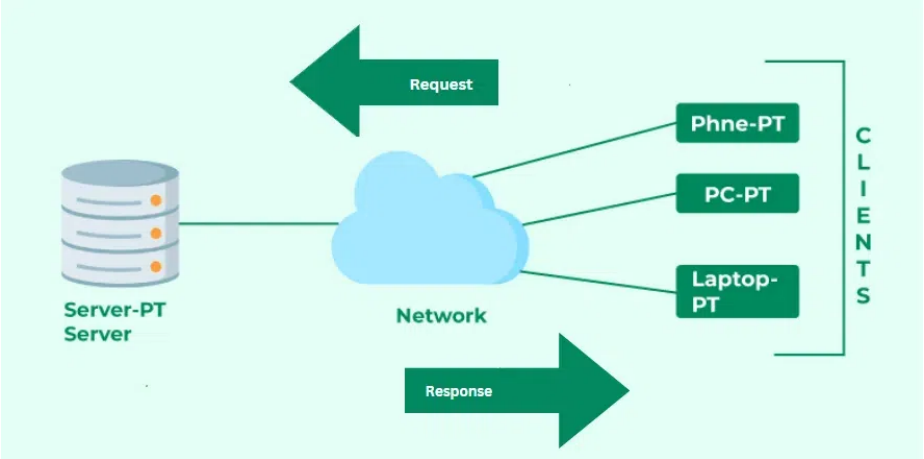
Repository Architecture

In repository architecture, the data is centralized and accessed frequently by other components, which modify data. The main purpose of this style is to achieve integrality of data. Data-centered architecture consists of different components that communicate through shared data repositories.



Client-Server Architecture

The Client-server model is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters called clients. In the client-server architecture, when the client computer sends a request for data to the server through the internet, the server accepts the requested process and delivers the data packets requested back to the client. Clients do not share any of their resources. Examples of the Client-Server Model are Email, World Wide Web, etc.



Pipe and filter Architecture

It consists of a sequence of m data-processing circuits, called stages or segments, which collectively perform a single operation on a stream of data operands passing through them

Main Differences and Use Cases of Software Architectures

MVC (Model-View-Controller)

Main Difference:

- **Separation of Concerns**: Divides the application into three interconnected components:
  - **Model**: Manages data and business logic.
  - **View**: Displays data to the user and sends user commands to the controller.
  - **Controller**: Handles user input, interacts with the model, and updates the view.

When to Use:

- **Web Applications**: Particularly useful for applications with a user interface, such as web apps.
- **Single Page Applications (SPAs)**: Helps manage complex user interactions and data binding.
- **Applications Requiring High Maintainability**: Facilitates easier updates and testing by keeping components separate.

Layered Architecture

Main Difference:

- **Hierarchical Organization**: Organizes the system into layers, each providing services to the layer above it and consuming services from the layer below.
  - Common layers include: Presentation, Business Logic, Data Access, and Database.

When to Use:

- **Enterprise Applications**: Suitable for large, complex systems with distinct functional areas.
- **Applications Needing Scalability**: Each layer can be developed and scaled independently.
- **Systems Requiring High Maintainability and Testability**: Each layer can be tested separately, improving maintainability.

Repository Architecture

Main Difference:

- **Centralized Data Management**: Uses a central repository that all components interact with to share data.
  - All data operations are performed through the repository.

When to Use:

- **Data-Centric Applications**: Ideal for systems where data consistency and central management are critical, such as databases and data warehouses.
- **Applications Requiring Centralized Data Control**: Ensures that all data operations are consistent and controlled from a central point.

Client-Server Architecture

Main Difference:

- **Distributed Computing**: Divides the system into two main components:
  - **Client**: Requests services and resources.
  - **Server**: Provides services and resources in response to client requests.

When to Use:

- **Web Services and APIs**: Commonly used in web applications and services.
- **Networked Applications**: Ideal for applications where multiple clients need to access shared resources, such as email servers, database servers, and file servers.
- **Applications Requiring Centralized Processing**: Centralizes processing and resource management on the server.

Pipe and Filter Architecture

Main Difference:

- **Data Flow Processing**: Organizes the system into a series of processing elements (filters) connected by pipes that pass data from one filter to the next.
  - Each filter performs a specific processing step on the data.

When to Use:

- **Data Transformation Applications**: Suitable for applications that process data in stages, such as compilers, data processing pipelines, and ETL (Extract, Transform, Load) systems.
- **Streaming Data Processing**: Ideal for real-time data processing where data flows continuously through the system.

Choosing the Right Architecture

1. **MVC**: Choose this when you need a clear separation of concerns, especially in web and user-interface-focused applications.

2. **Layered**: Ideal for complex systems with distinct functional areas that require scalability and maintainability.
3. **Repository**: Best for applications that require centralized data management and consistency.
4. **Client-Server**: Suitable for distributed applications where multiple clients need to access shared services or resources.
5. **Pipe and Filter**: Use this for applications that process data in sequential steps or require real-time data processing.